Game Semantics: Easy as Pi

Introducing Programming Game Semantics

SIMON CASTELLAN, Imperial College London, United Kingdom
LÉO STEFANESCO, IRIF, Université Paris Diderot, CNRS, France
NOBUKO YOSHIDA, Imperial College London, United Kingdom

Game semantics has proven to be a robust method to give compositional semantics for a variety of higher-order programming languages. However, due to the complexity of most game models, game semantics has remained unapproachable for non-experts.

In this paper, we aim at making game semantics more accessible by viewing it as a syntactic translation to a dialect of the π-calculus, referred to as *metalanguage*, followed by a semantic interpretation of the metalanguage into a particular game model. The semantic interpretation is done once and for all; while the syntactic translation can be defined for a wide range of programming languages without knowledge of the particular game model used. Reasoning on the interpretation (soundness and adequacy) can be done at the level of the metalanguage through a sound equational theory, escaping tedious technical proofs usually found in game semantics. We call this methodology *programming game semantics*.

We expose the methodology in three steps of increasing expressivity, building on concurrent game semantics based on event structures. By developing an extension of the existing models to deal with non-angelic nondeterminism, and nonlinear computation, we can give very accurate models of complex languages by a simple translation into a typed variant of the π-calculus inspired by Differential Linear Logic. We illustrate this expressivity on IPA, a higher-order programming language with shared-memory concurrency. By simply translating it into the metalanguage, we give the first model of IPA, which is (1) causal and (2) adequate for the usual operational notion of bisimulation — a novel result.

To make the development more concrete, we have built a simple prototype to compute the interpretation of the target programming language into the metalanguage and games strategies.

Additional Key Words and Phrases: Game Semantics, Pi-Calculus, Session Types, Metalanguage, Programming, Linear Logic, Event Structures, Denotational Semantics

1 INTRODUCTION

Background. Semantics of programming languages are usually *operational*, modelling the concrete execution on the metal by an abstract machine made of mathematical symbols and Greek letters. Operational semantics tend to be very flexible and model a wide range of programming features. However, traditional operational semantics struggle to express the behaviours of *open* programs, ie. programs with external parameters. This is where denotational semantics shines, by providing *compositional semantics*, where an open program is typically interpreted as a function from its external parameters to the result. The most common form of denotational semantics, based on domain theory, copes well with higher-order functions, but more laboriously with effectful computations, and almost not at all with concurrency.

At the dawn of the 1990s, a new form of denotational semantics appeared: *game semantics* [Abramsky et al. 2000; Hyland and Ong 2000]. There, an open program is modelled by its possible **interactions** with the context. Thanks to its interactivity, this methodology has proved to be

Authors' addresses: Simon Castellan, Imperial College London, London, United Kingdom, s.castellan@ic.ac.uk; Léo Stefanesco, IRIF, Université Paris Diderot, CNRS, Paris, France, leo.lveb@gmail.com; Nobuko Yoshida, Imperial College London, London, United Kingdom, n.yoshida@ic.ac.uk.

2019. XXXX-XXXX/2019/7-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

very extensible and support a wide range of programming features (references, control operators, probabilities, concurrency, quantum, etc.)

Moreover, as observed by Ghica and Tzevelekos [2012], game semantics reconciles denotational and operational semantics together: the interaction traces of a program can be computed either denotationally (by induction on the syntax) or operationally (by running an abstract machine).

While game semantics has been recognised as a powerful tool to build (fully abstract) denotational models, we believe that its most useful and promising feature is the simplicity with which one can *describe* the compositional behaviour of systems in general (eg. with its recent use for verifying operating systems [Gu et al. 2018] or compilers [Stewart et al. 2015]). So far, its simplicity has not been apparent, and game semantics is often considered inaccessible to non-experts who wish to define a denotational model based on games for their favourite language.

The Framework. The thesis of this paper is that, the complexity of game semantics interpretations can be cut down by introducing a simple message-passing intermediate language, between the source program and the model. Indeed, game semantics bundles two things together: the idea of interpreting a program as a process interacting with its environment, *and* a mathematical formalisation of this process. Most game semantics models of the same language only differ in the latter. Our main contribution in this paper is to make this separation explicit by factorising the interpretation of a language in game semantics as the following steps:

$$\begin{tikzcd}
\boxed{\begin{array}{c}\textbf{Program}\\ \text{(Source language)}\end{array}} \arrow[r, "\text{syntactic}", "\text{translation}"'] & \boxed{\begin{array}{c}\textbf{Process}\\ \text{(Metalanguage)}\end{array}} \arrow[r, "\text{interpretation}"] & \boxed{\begin{array}{c}\textbf{Strategy}\\ \text{(Game Model)}\end{array}}
\end{tikzcd}$$

This factorisation offers several advantages. First, it allows to decouple the interpretation of the source language from the details of the model. One can use the same translation to obtain different models (traces, event structures, ...), and, conversely, one can use the same model for different translations. Second, interpreting a language becomes a matter of writing a syntactic translation, an easier task than doing the interpretation directly, since one does not have to deal with the complexity of the model.

A Language for Strategies. What would a good programming language for strategies be? Strategies represent the interaction of the program with the environment in the form of messages that are exchanged between them. For that reason, a message-passing language such as the π-calculus [Milner et al. 1992] is an ideal candidate. Dating back to encodings of the call-by-name and call-by-value λ-calculi by Milner [1992], the π-calculus has been used to encode a wide range of programming languages, including functional, concurrent, and distributed languages. Its connection with game semantics has been studied ever since the introduction of game semantics [Hyland and Ong 1995]. However, interpretations in game semantics, viewed through the lenses of the π-calculus, do not use the *free name output* feature of the π-calculus (which creates semantic difficulties). This subset of the π-calculus, called the internal π-calculus, represents causalities of interactions by bound name passing, capturing greater expressiveness than (a core) CCS [Sangiorgi 1996].

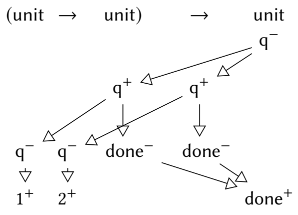
Some game semantics interpretations rely on the use of the categorical semantics of Linear Logic [Melliès 2009]. Through session types, there is a Curry-Howard correspondence between the internal π-calculus and Linear Logic, discovered by Caires and Pfenning [2010]; Wadler [2014]. More specifically, this paper uses an extension of the calculus in [Caires and Pfenning 2010; Wadler 2014] to Differential Linear Logic (DiLL) [Ehrhard 2018] where $\otimes$ and $\parr$ are identified to be able to

$$f : \mathbb{N} \to \mathsf{unit} \vdash f\,1 \parallel f\,2 : \mathsf{unit}$$

(a) Program (written in IPA)

$$\begin{array}{ll} (\nu xy; x'y')(o(). & \\ & f[a].\,(a().\,a \oplus 1 \parallel f().\,x \oplus \mathsf{done}) \\ \parallel_f & f[a].\,(a().\,a \oplus 2 \parallel f().\,x \oplus \mathsf{done}) \\ \parallel & y().\,y'().\,o \oplus \mathsf{done}) \\ \multicolumn{2}{l}{\triangleright\, f : (\!| \mathbb{N} \to \mathsf{unit} |\!)^{\perp}, o : (\!| \mathsf{unit} |\!)} \end{array}$$

(b) Process (written in π_{DiLL})



(c) Strategy (event structure)

Fig. 1. An overview of the methodology on an example

interpret languages with deadlocks. Session types are a natural fit here since their connection with game semantics have been recently discovered [Castellan and Yoshida 2019].

Contributions and Outline of the Paper. This paper proposes a framework where we view game semantics interpretations as syntactic translations to a process language, followed by a semantic interpretation of this process language into a game model. This allows us to carry out calculations on the model syntactically via an equational theory on the process language.

We present our framework in increasingly complex settings. **§ 2** introduces a model for *linear deterministic concurrent* strategies, due to Rideau and Winskel [2011]; a metalanguage, π_{MALL}, based on Multiplicative Additive Linear Logic (MALL) along with an equational theory; and a translation of the affine λ-calculus into π_{MALL}. **§ 3** extends the model to *nonlinear deterministic concurrent strategies*, presenting the model of [Castellan et al. 2015]; an extension of the metalanguage to π_{LL}, based on Linear Logic; and translations from call-by-name and call-by-value PCF into π_{LL}. Finally **§ 4** presents the new model of nondeterministic concurrent computation; extends the metalanguage to π_{DiLL} using ideas from Differential Linear Logic and shows how to give the first *causal*, *compositional* and *non-angelic* model of *Idealised Parallel Algol* (IPA), a higher-order programming language with shared memory concurrency.

Setting	Model	Calculus	Encoding
Linear (§ 2)	**LDStr** [Rideau and Winskel 2011]	π_{MALL}	Affine λ-calculus
Deterministic (§ 3)	**DStr** [Castellan et al. 2015]	π_{LL}	PCF
Nondeterministic (§ 4)	**Str** ($\star$)	$\pi_{\mathsf{DiLL}}(\star)$	IPA ($\star$)

The methodology is illustrated in Figure 1 on a parallel program calling twice an external function f. This program is translated to the π-calculus which is then interpreted as an event structure describing the causal relationships between the different actions of the program (see § 4.3 for the detailed description).

Apart from our proposal of this uniform framework for game semantics interpretations, together with its new metalanguage and the proof techniques developed for deterministic computations, this paper gives three fresh contributions on the game semantics side (denoted as **($\star$)** in the table above). The first contribution is a new game model for concurrent and nondeterministic higher-order computation, which combines the approaches of [Castellan et al. 2015] and [Castellan et al. 2018]. The second contribution is a metalanguage based on DiLL and its interpretation inside the model, as well as a sound (in)equational theory axiomatising the quotient induced by the interpretation. The third contribution is the first model for IPA based on event structures, which is adequate for weak bisimulation. Starting from the most basic game model, these new results are naturally derived as a consequence of our methodology. **§ 5** outlines our prototype implementation of the interpretation of the metalanguage and of IPA. **§ 6** provides the related and future work. Proofs can be found in ***Appendix***, and we shall submit our prototype implementation to ***Artefact Evaluation***.

We make use of the `knowledge` package. **Definitions** of mathematical concepts appear blue boldface, and their uses occur in blue and are linked to the original definition.

2 LINEAR AND DETERMINISTIC GAMES

This section introduces the framework in the simplest setting: *deterministic* and *linear*. We use here **LDStr**, the category of (forest-like, confusion-free and race-free) games, and linear and deterministic strategies from [Rideau and Winskel 2011]. We start by a brief introduction to prime event structures in § 2.1 followed by the description of deterministic strategies in § 2.2. The category **LDStr** and its structure are defined in § 2.3.

From § 2.4, the contributions of our paper starts. We propose a metalanguage, π_{MALL}, to describe the games and strategies of this model. π_{MALL} is based on an extension of MALL where $⅋$ and $\otimes$ are identified and become a unique connective written $\|$ (by analogy with categorical semantics of linear logic, we call this variant *compact-closed MALL*). π_{MALL} is similar syntactically to the linear fragment of binary session types [Honda et al. 1998], though the semantics are slightly different.

In § 2.5, π_{MALL} is interpreted in **LDStr**. This induces an equivalence relation on π_{MALL}. We propose a sound equational theory that allows us to reason about semantic equality. In § 2.6, we show the first use of our framework by translating an affine λ-calculus into π_{MALL} and show it is sound and complete with respect to $\beta\eta$-equivalence using the equational theory of π_{MALL}.

2.1 Event Structures

Event structures are a model of concurrent and nondeterministic computation based on the notion of causally ordered events [Winskel 1986]. We use here *prime event structures with binary conflict.*

Definition 2.1. An **event structure** is a triple $(E, \leq_E, \#)$ where $(E, \leq_E)$ is a partial order, and $\# \subseteq E^2$ is a binary irreflexive relation satisfying: (1) $[e] \overset{\text{def}}{=} \{e' \in E \mid e' \leq e\}$ is finite; and (2) if $e\#e'$ and $e' \leq e''$ then $e\#e''$.

Two events e, e' are in **conflict** when $e\#e'$ and are **compatible** otherwise. Two compatible events which are not ordered are called **concurrent**. We write $e \rightarrowtriangle e'$ (**immediate causal dependency**) when $e <_E e'$ with no events in between, and $e \sim\!\sim e'$ (**minimal conflict**) when (e, e') is the only conflicting pair in $[e] \cup [e']$. From $\rightarrowtriangle$ and $\sim\!\sim$, we can recover $\leq$ and $\#$ via axiom (2). Depictions of event structures will use $\rightarrowtriangle$ and $\sim\!\sim$.

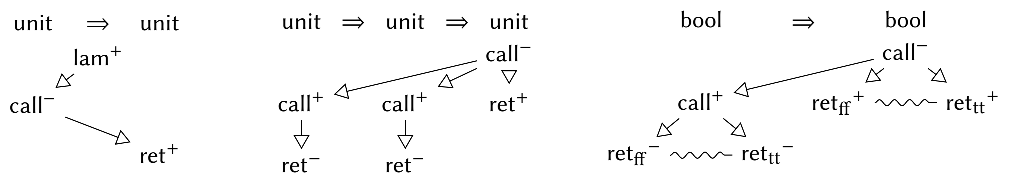
Given an event structure E, a **configuration** of E is a subset $x \subseteq E$ down-closed for $\leq_E$ and conflict-free. We write $\mathscr{C}(E)$ for the set of finite configurations of E. For $x \in \mathscr{C}(E)$, an **extension** of x is an event $e \in E \setminus x$ such that $x \cup \{e\} \in \mathscr{C}(E)$; we write $x \overset{e}{\longrightarrow\!\!\subset}$.

An event structure E is **confusion-free** when (1) $e \sim\!\sim e'$ implies $[e) = [e')$ and (2) the relation $(\sim\!\sim_E \cup =_E)$ is an equivalence relation. Its equivalence classes are called **cells**.

Finally, given a set $V \subseteq E$, the **projection** of E to V is the event structure $(V, \leq_E \cap V^2, \# \cap V^2)$.

Constructions on Event Structures. Given a family of event structures $(E_{i\in I})$ we define their **parallel composition** $\|_{i\in I}\ E_i$ as follows. Its events are pairs $(i \in I, e \in E_i)$. Causality and conflict are obtained by lifting those from the E_i:

$$(i, e) \leq_{\|E_i} (j, e') \overset{\text{def}}{=} (i = j \wedge e \leq_{A_i} e') \qquad (i, e)\#_{\|E_i}(j, e') \overset{\text{def}}{=} (i = j \wedge e\#_{A_i}e')$$



(a) unit ⇒ unit (CBV) (b) unit ⇒ unit ⇒ unit (CBN) (c) bool ⇒ bool (CBN)

Fig. 2. Examples of Games

Similarly, the **nondeterministic sum** of the E_i, written $\sum_{i\in I} E_i$ has the same components as $\|_{i\in I} E_i$ except for conflict:

$$(i,e)\#_{\sum_{i\in I} E_i}(j,e') \overset{\text{def}}{=} (i = j \Rightarrow e\#_{E_i}e').$$

The empty event structure, written $\emptyset$ is the unit for both parallel composition and sum.

Maps of Event Structures. A **partial map of event structures** is a partial function $f : E \rightharpoonup F$ such that the direct image of a configuration is a configuration and which is **locally injective**: f restricted to any configuration of E is injective. We write $\mathrm{dom}(f)$ for the domain of f. When $\mathrm{dom}(f) = E$, f is said to be **total**. Event structures and total maps form a category ES. The following categorical property will be key to compose strategies later on.

LEMMA 2.2 ([WINSKEL 1986]). *The category* ES *has pullbacks.*

2.2 Games and Strategies on Event Structures

We recall the game model based on event structures [Rideau and Winskel 2011].

Games as Polarised Event Structures. Games arise as particular event structures, where events are equipped with a polarity:

Definition 2.3. A **game** is an event structure A along with a labelling $pol : A \to \{-,+\}$ such that:

(1) A is confusion-free and **race-free**: if $a \sim_A a'$, then $pol(a) = pol(a')$
(2) $(A, \leq_A)$ is a **forest**: elements of $[a]$ are totally ordered by $\leq_A$ for any $a \in A$.

These restrictions on games make the development simpler and is sufficiently expressive to represent the behaviour of types we are interested in. Parallel composition and nondeterministic sum extend to games. Similarly, the empty event structure extends to the empty game written **1**. Given a game A, its **dual** $A^\perp$ is the game obtained from A by reversing polarities. Given an element $e \notin A$ and a polarity $p \in \{-,+\}$, we write $e^p \cdot A$ (**lifting**) the game obtained from A by adjoining an event e with polarity p which is below and consistent with all events of A. If $x, y \in \mathcal{C}(A)$, we write $x \subseteq^p y$ for $x \subseteq y$ and $pol(y \setminus x) \subseteq \{p\}$. A game is **negative** when its minimal events are all negative; **positive** if they are all positive. A game is **polarised** if it is either negative or positive.

We give a few examples of games in Figure 2. Each game corresponds to a particular function type, under a particular calling convention (call-by-name or call-by-value). Remember that a game represents the allowed messages that Program and Context can exchange during the course of the execution. As depicted in Figure 2a, in unit → unit in call-by-value, there are three moves. The first move is a Player move telling the context that the program has evaluated to a closure (as it may diverge). Once this move has been received, Context can call the closure via call. When called, Program can return from the closure via ret. The only rules we have here are due to the causal ordering which constrains the order in which the moves can be played by the players. In call-by-name, however, the dynamic is reversed: Context starts the computation by calling the function[1]. Moreover, arguments are not values but *thunks*. Each thunk can be evaluated by Program via a call move, as illustrated in Figure 2b. Finally, our third example illustrates the interpretation of types with several values: each value becomes a different move. Because there can only be one value exchanged in one run of the system, the different moves are in conflict as evidenced in Figure 2c. Note that moves are drawn in columns, to ease readability.

Strategies as Labelled Event Structures. Let us introduce the strategies of [Rideau and Winskel 2011]. Strategies will be their own event structure S along with a labelling map $S \to A$. Often the labelling will be injective, allowing us to see S as a subset of A; but this is not always the case (cf. Figure 3b).

Definition 2.4. A **deterministic strategy** (in the rest of the text, simply strategy) on a game A is a total map of event structures $\sigma : S \to A$ such that

Receptivity For $x \in \mathscr{C}(S)$ and a a negative extension of $\sigma\, x$, there exists a unique $x \overset{s}{\longrightarrow\!\!\subset}$ with $\sigma\, s = a$.
Courtesy For any $s \rightarrowtriangle_S s'$ such that $\sigma\, s$ and $\sigma\, s'$ are concurrent, $pol(s) = -$ and $pol(s') = +$.
Determinism If $s \sim\!\sim_S s'$, then $\sigma\, s \sim\!\sim_A \sigma\, s'$ and both are negative.

Asking for σ to be a map of event structures amounts to asking that σ respects the rules of the game: configurations of S should correspond to configurations (ie. executions) of A, in particular σ should not be able to play a move before it has played the move unlocking it. Moreover, within a configuration, moves should be played at most once. The conditions of receptivity and courtesy are needed for strategies to form a category and imply a strong form of asynchrony (see § 2.3). Determinism amounts to saying that nondeterministic choices can only come from Context.

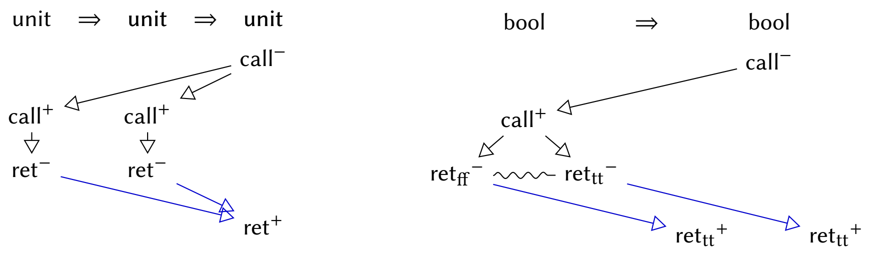
A strategy from a game A to a game B is a strategy on the composite game $A^\perp \parallel B$. We write $\sigma : A \,{\to\!\!\!\!+}\, B$ to distinguish them from maps. We will omit displaying both S and A, along with the labelling, but rather adopt a concise representation as in Figure 3 where we only display S with labels in A and use columns to disambiguate. Figure 3a depicts a simple example of a concurrent strategy, which given two thunks evaluates them in parallel and only returns when both have returned. We have highlighted in blue the causal links that the strategy adds onto the game, which must be from negative to positive by courtesy. Figure 3b depicts a strategy where the labelling σ is not injective. Indeed, the event $\mathrm{ret}_{\mathrm{tt}}$ occurs twice, but the two occurrences are in conflict (inherited from the conflict $\mathrm{ret}_{\mathrm{tt}}{}^- \sim\!\sim \mathrm{ret}_{\mathrm{ff}}{}^-$).

Given a game A, we form the **copycat strategy** on A, $\text{Æ}_A : \mathbb{C}_A \to A^\perp \parallel A$, as follows. The events of $\mathbb{C}_A$ are exactly those of $A^\perp \parallel A$, and $\leq_{\mathbb{C}_A}$ is obtained from the transitive closure of $\leq_{A^\perp \parallel A} \cup \{(a, \bar{a}) \mid a \in A^\perp \parallel A \text{ is negative}\}$ where we write $\bar{a} \in A^\perp \parallel A$ for the corresponding event to a on the other side. Then $a \#_{\mathbb{C}_A} a'$ holds when $[a]_{\mathbb{C}_A} \cup [a']_{\mathbb{C}_A} \in \mathscr{C}(A^\perp \parallel A)$. The labelling function Æ_A is simply the identity. A strategy is **negative** when its minimal events are all sent to negative events of the game. Copycat for example is a negative strategy.

The empty strategy $!_A$ on a game A is defined as the inclusion $A_0 \subseteq A$ where A_0 contains events $a \in A$ such that $[a]$ only contains negative events (ie. it is the minimal receptive strategy).

Because S is arbitrary, equality of strategies is not meaningful. Rather, we consider strategy up to isomorphism. Two strategies $\sigma : S \rightharpoonup A$ and $\tau : T \rightharpoonup A$ are **isomorphic** (written $\sigma \cong \tau$) when there exists an isomorphism of event structures $\varphi : S \cong T$ (that is a map $S \to T$ with an inverse) such that $\tau \circ \varphi = \sigma$.

[1]There is no need for Program to tell Context it has evaluated to a closure since $\lambda x.\,\bot \equiv \bot$ in call-by-name, while this equality does not hold in call-by-value



(a) Interpretation of $\lambda xy.\, x \parallel y$ (CBN) (b) Interpretation of $\lambda x.$ if x tt tt : **bool** (CBN)

Fig. 3. Examples of Strategies

2.3 The category LDStr

We now explore the algebraic properties of strategies, which have guided the choice for the metalanguage π_{MALL} (introduced in § 2.4).

Composition of Strategies. Let us first review the main operation on strategies: composition. Given a strategy $\sigma : A \mathrel{+\!\!\!\rightarrow} B$ from A to B, and a strategy $\tau : B \mathrel{+\!\!\!\rightarrow} C$ from B to C, we would like to combine them to form a strategy $\tau \odot \sigma$ from A to C. This is done by letting σ and τ interact on B, giving an event structure $T \circledast S$ (not a strategy) which projects to $A \parallel B \parallel C$. The composition is then obtained by hiding events projected to B to recover a strategy $\tau \odot \sigma : A^\perp \parallel C$.

In a few words, interaction amounts to merging the causal constraints of σ and τ (ie. do the union of the causal orderings), and pruning any resulting cycles. This rather complex construction can be elegantly formulated as a categorical pullback: the interaction $\tau \circledast \sigma : T \circledast S \to A \parallel B \parallel C$ of $\sigma : S \to A^\perp \parallel B$ and $\tau : T \to B^\perp \parallel C$ is obtained as the pullback of $\sigma \parallel C : S \parallel C \to A \parallel B \parallel C$ and of $A \parallel \tau : A \parallel T \to A \parallel B \parallel C$. Their **composition** is then obtained by projecting $T \circledast S$ to events that are sent to A or C:

$$T \odot S = (T \circledast S) \downarrow \{e \in T \circledast S \mid (\tau \circledast \sigma)(e) \in A \parallel C\} \qquad \tau \odot \sigma = \tau \circledast \sigma|_{T \odot S} : T \odot S \to A^\perp \parallel C.$$

Lemma 2.5 ([Rideau and Winskel 2011]). *Composition of strategies is associative, and copycat is an identity on strategies, both up to isomorphism.*

For copycat to be an identity, the conditions of receptivity and courtesy are necessary. Intuitively, this is because composing a strategy with copycat amounts to prefixing it with an asynchronous buffer; and if the strategy is too "synchronous", this buffer may have observable effects.

2.3.1 Categorical Structure. We define **LDStr**, the category whose objects are finite games and morphisms from A to B are strategies $\sigma : A \mathrel{+\!\!\!\rightarrow} B$. First, duality of games extends to an isomorphism of category **LDStr** ≅ **LDStr**$^{\mathrm{op}}$: any strategy $\sigma : A \mathrel{+\!\!\!\rightarrow} B$ can be regarded as a strategy $\sigma^\perp : B^\perp \mathrel{+\!\!\!\rightarrow} A^\perp$. Moreover, parallel composition of games forms a monoidal structure on **LDStr**. Therefore, **LDStr** is compact-closed:

Lemma 2.6 ([Castellan et al. 2017]). *(**LDStr**, $\parallel$, 1, $(-)^\perp$) is a compact-closed category.*

There is a way to lift certain maps of event structures between games to strategies that is useful to obtain structural maps. A **map of games** is a partial map $f : A \rightharpoonup B$ preserving polarities such that (1) f restricted to its domain is bijective, (2) f preserves causal ordering and conflict. The **lifting of f** is the strategy $\mathsf{lift}(f) \overset{\text{def}}{=} \mathbb{C}_A \downarrow A^\perp \parallel \mathsf{dom}(f) \xrightarrow{A^\perp \parallel f} A^\perp \parallel B$. Though **LDStr** does not have (co)products, it has weak (co)products, that are defined by lifting. Given two games A and B, let us define $A \,\&\, B \overset{\text{def}}{=} (\mathsf{L}^- \cdot A) + (\mathsf{R}^- \cdot B)$. There are obvious maps of games $p_1 : A \,\&\, B \rightharpoonup A$ and $p_2 : A \,\&\, B \rightharpoonup B$ and we let $\pi_1 = \mathsf{lift}(p_1) : A \,\&\, B \twoheadrightarrow A$ and $\pi_2 = \mathsf{lift}(p_2) : A \,\&\, B \twoheadrightarrow B$.

Lemma 2.7. *For every strategies $\sigma : A \twoheadrightarrow B$ and $\tau : A \twoheadrightarrow C$ there exists a strategy $\langle \sigma, \tau \rangle : A \twoheadrightarrow B \,\&\, C$ such that $\pi_1 \odot \langle \sigma, \tau \rangle \cong \sigma$ and $\pi_2 \odot \langle \sigma, \tau \rangle \cong \tau$. By duality, we obtain weak coproducts $A \oplus B = (A^\perp \,\&\, B^\perp)^\perp$.*

2.4 A Metalanguage for LDStr: π_{MALL}

Types and Processes. First, let us look at how to describe games *syntactically*. Since our games are finite forests, they can be built inductively from $\|$, $\oplus$ and $\&$. This decomposition crucially relies on the fact that games are confusion- and race-free. This suggests the following grammar for types:

$$T ::= \mathbf{1} \mid (T \parallel \ldots \parallel T) \mid \&_{i \in I} T_i \mid \oplus_{i \in I} T_i$$

In the grammar, k, i and j range over *labels* $\mathbb{L}$, to be thought of as values such as integers and booleans. The type grammar corresponds to Multiplicative Additive Linear Logic (without additive units and literals) except that $\otimes$ and $⅋$ have been collapsed into one operation, $\|$ (similarly for the multiplicative units). This is to be expected since the main asymmetry between $\otimes$ and $⅋$ arise as a way to avoid deadlocks during cut-elimination. In our setting, deadlocks are permitted so there is no need for a distinction. We call this variant *compact-closed MALL*, because identifying $\otimes$ with $⅋$ is a common feature in models that are compact-closed (eg. games or the relational model).

Types come equipped with a notion of *duality* stemming from the usual De Morgan laws: $\&$ and $\oplus$ are dual to each other and $\|$ and $\mathbf{1}$ are self-dual. We will use the notation $\ell^- \cdot T$ and $\ell^+ \cdot T$ for $\&_{i \in \{\ell\}} T$ and $\oplus_{i \in \{\ell\}} T$. We will also use the binary version $A \,\&\, B$ for $\&_{\ell \in \{\mathtt{inl}, \mathtt{inr}\}} T_\ell$ with $T_{\mathtt{inl}} = A$ and $T_{\mathtt{inr}} = B$; and dually for $A \oplus B$. For instance, let us introduce $\mathbb{B} = \mathsf{q}^- \cdot (\oplus_{b \in \{\mathrm{tt}, \mathrm{ff}\}} 1)$, the type of (call-by-name) booleans. The initial move represents the evaluation request from Context, as call-by-name booleans are thunks waiting for a request from the environment before starting to evaluate.

As for terms, we draw inspiration from the connection between Linear Logic and session types [Caires and Pfenning 2010; Wadler 2014]. The calculus turns out to be a subset of standard binary session types, the main difference being that our calculus has no free name output. The syntax for processes is as follows:

$$\begin{array}{rlllllll} P & ::= & \mathbf{0} & \text{(null)} & \mid \; P \parallel Q & \text{(parallel)} & \mid \; (\nu\, a\, b : A)P & \text{(hiding)} \\ & \mid & a \oplus k.\, P & \text{(selection)} & \mid \; a \,\&\, \{i.\, P_i\}_{i \in I} & \text{(branching)} & \mid \; a\{x_1, \ldots, x_n\}.\, P & \text{(split)} \end{array}$$

where $a, b, x, \ldots$ range over the set of channels $\mathbb{N}$ which transmit values. Process $(\nu\, a\, b : A)P$ is a restriction of two channels where type of a is A and type of b is its dual; Process $a \oplus k.\, P$ is an output of value k (a label) which selects the k-th branch of an input process $a \,\&\, \{i.\, P_i\}_{i \in I}$. Process $a\{x_1, \ldots, x_n\}$ (used to introduce parallel processes) is similar to the internal π-calculus [Sangiorgi 1996]. It is however not sending (or receiving) fresh channels $x_1, \ldots, x_n$. Instead, it is used to break down a channel of type $a : (T_1 \parallel \ldots \parallel T_n)$ into channels $x_i : T_i$; the idea being that several parallel protocols will be interleaved on a and an action on one of the x_i amounts to an action on a, paired with the natural number $i \in \mathbb{N}$ (thought to be a "port number"). For a process P, we write $\mathsf{fc}(P)$ for the set of free channels occurring in P, defined by $\mathsf{fc}((\nu\, a\, b : A)P) = \mathsf{fc}(P) \setminus \{a, b\}$, $\mathsf{fc}(a\{x_1, \ldots, x_n\}.P) = \mathsf{fc}(P) \setminus \{x_1, \ldots, x_n\}$ and other standard rules.

Typing Rules. The typing rules are given in Figure 4. Each typing rule corresponds to a rule of compact-closed MALL. Because of deadlocks, this typing system only ensures an *affine* channel usage. Γ denotes a mapping from channels to types. Mix ensures that two environments are disjoint;

$$\textsc{Weaken}\ \frac{}{\mathbf{0} \triangleright \Gamma} \qquad \textsc{Mix}\ \frac{P \triangleright \Gamma_1 \qquad Q \triangleright \Gamma_2}{P \parallel Q \triangleright \Gamma_1, \Gamma_2} \qquad \textsc{Cut}\ \frac{P \triangleright a : T, b : T^\perp, \Gamma}{(\nu\, a\, b : T) P \triangleright \Gamma} \qquad \oplus\text{-INTRO}\ \frac{k \in I \qquad P \triangleright \Gamma, a : T_k}{a \oplus k.\, P \triangleright \Gamma, a : \oplus_{i \in I} T_i}$$

$$\&\text{-INTRO}\ \frac{\forall i : I, P_i \triangleright \Gamma, a : T_i}{a \,\&\, \{i.\, P_i\}_{i \in I} \triangleright \Gamma, a : \&_{i \in I} T_i} \qquad \parallel\text{-INTRO}\ \frac{P \triangleright \Gamma, x_1 : T_1, \ldots, x_n : T_n}{a\{x_1, \ldots, x_n\}.\, P \triangleright \Gamma, a : (T_1 \parallel \ldots \parallel T_n)}$$

Fig. 4. Typing Rules for Compact-Closed MALL

Cut composes two channels with dual types and hides them; rules ⊕/&-Intro are standard; and ||-Intro types a split process with a corresponding parallel type.

An example of a useful process is the **forwarder** $[a{\leftrightarrow}b]_A \triangleright a : A, b : A^\perp$, the syntactic counterpart to the copycat strategy. It is defined by induction on A:

$$[a{\leftrightarrow}b]_{\mathbf{1}} \overset{\text{def}}{=} \mathbf{0} \qquad [a{\leftrightarrow}b]_{T_1 \parallel \ldots \parallel T_n} \overset{\text{def}}{=} a\{a_1 \ldots a_n\}.\, b\{b_1 \ldots b_n\}.\, ([a_1{\leftrightarrow}b_1]_{T_1} \parallel \cdots \parallel [a_n{\leftrightarrow}b_n]_{T_n})$$

$$[a{\leftrightarrow}b]_{T^\perp} \overset{\text{def}}{=} [b{\leftrightarrow}a]_T \qquad [a{\leftrightarrow}b]_{\&_{i \in I} T_i} \overset{\text{def}}{=} a \,\&\, \left\{i.\, b \oplus i.\, [a{\leftrightarrow}b]_{T_i}\right\}_{i \in I}$$

Another more concrete example is the process if for call-by-name conditional:

$$\text{if}^o\ c\ b_1\ b_2 \overset{\text{def}}{=} o \,\&\, \left\{ \mathsf{q} : c \oplus \mathsf{q}.\, c \,\&\, \begin{cases} \mathsf{tt} : b_1 \oplus \mathsf{q}.\, [o{\leftrightarrow}b_1]_{\mathbb{B}} \\ \mathsf{ff} : b_2 \oplus \mathsf{q}.\, [o{\leftrightarrow}b_2]_{\mathbb{B}} \end{cases} \right\} \triangleright c : (\mathbb{B})^\perp, b_i : (\mathbb{B})^\perp, o : \mathbb{B}$$

It has four channels: c for the condition, b_1 and b_2 for the values of the branches, and o for the output. When Context requests the outcome of i, the condition is evaluated, and, according to the value received, b_1 or b_2 is evaluated and its answer is forwarded to o.

2.5 Interpretation into LDStr

The interpretation of π_{MALL} inside **LDStr** maps session types to games; and processes to strategies.

Interpretation of Types. First, let us interpret types as finite games as follows:

$$[\![\mathbf{1}]\!] = \mathbf{1} \qquad [\![T_1 \parallel \ldots \parallel T_n]\!] = [\![T_1]\!] \parallel \ldots \parallel [\![T_n]\!] \qquad [\![\&_{i \in I} T_i]\!] = \&_{i \in I} [\![T_i]\!] \qquad [\![\oplus_{i \in I} T_i]\!] = \oplus_{i \in I} [\![T_i]\!].$$

We have $[\![T^\perp]\!] = [\![T]\!]^\perp$ as desired. Moreover, every game arises as the interpretation of a type:

Lemma 2.8 ([Castellan and Yoshida 2019]). *For every finite game A, there exists a type T such that $[\![T]\!] \cong A$.*

Interpretation of Terms. We now interpret typing derivations as morphisms in **LDStr**. The judgement $P \triangleright \Delta$ will be interpreted as a strategy on $[\![\Delta]\!]$. The interpretation is described in Figure 5 where we use abundantly the fact that a strategy on $[\![\Delta_1]\!] \parallel [\![\Delta_2]\!]$ can be viewed as a strategy $[\![\Delta_1]\!]^\perp \twoheadrightarrow [\![\Delta_2]\!]$. Notice that $[\![[a{\leftrightarrow}b]_T]\!] = \alpha_{[\![T]\!]}$ and that $[\![(\nu ab)(P \mid Q)]\!]$ where $a \in \mathsf{fc}(P)$ and $b \in \mathsf{fc}(Q)$ is equal to $[\![Q]\!] \odot [\![P]\!]$. The rule for $a\{x_1, \ldots, x_n\}.\, P$ is a simple "renaming" semantic-wise, since the games for $x_1 : T_1, \ldots, x_n : T_n$ and $a : (T_1 \parallel \ldots \parallel T_n)$ are isomorphic. The interpretation of outputs and inputs is done using the weak product structure. The interpretation of restriction uses the trace operator induced by the compact-closed structure, which consists in composing by copycat to connect a and b via an asynchronous forwarder.

Since strategies are finite, we have a definability result.

Theorem 2.9 (Definability, [Castellan and Yoshida 2019]). *Every strategy in $[\![\Delta]\!]$ is the interpretation of a process $P \triangleright \Delta$.*

$$\left\llbracket \frac{}{\mathbf{0} \triangleright \Delta} \right\rrbracket = !_\Delta \qquad \left\llbracket \frac{P \triangleright \Gamma_1 \qquad Q \triangleright \Gamma_2}{P \parallel Q \triangleright \Gamma_1, \Gamma_2} \right\rrbracket = \llbracket P \rrbracket \parallel \llbracket Q \rrbracket \qquad \left\llbracket \frac{P \triangleright \Gamma, x_1 : T_1, \ldots, x_n : T_n}{a\{x_1, \ldots, x_n\}.\, P \triangleright \Gamma, a : T_1 \parallel \ldots \parallel T_n} \right\rrbracket = \llbracket P \rrbracket$$

$$\left\llbracket \frac{k \in I \qquad P \triangleright \Gamma, a : T_k}{a \oplus k.\, P \triangleright \Gamma, a : \oplus_{i \in I} T_i} \right\rrbracket = (\pi_k^\perp : \llbracket T_k \rrbracket \mathrel{+\!\!\!\!\longrightarrow} \llbracket \oplus_{i \in I} T_i \rrbracket) \odot (\llbracket P \rrbracket : \llbracket \Gamma \rrbracket^\perp \mathrel{+\!\!\!\!\longrightarrow} \llbracket T_k \rrbracket)$$

$$\left\llbracket \frac{\forall i : I, P_i \triangleright \Gamma, a : T_i}{a \,\&\, \{i.\, P_i\}_{i \in I} \triangleright \Gamma, a : \&_{i \in I} T_i} \right\rrbracket = \langle \llbracket P_i \rrbracket : \llbracket \Gamma \rrbracket^\perp \mathrel{+\!\!\!\!\longrightarrow} \llbracket T_i \rrbracket \rangle_{i \in I}$$

$$\left\llbracket \frac{P \triangleright a : T, b : T^\perp, \Gamma}{(\nu\, a\, b : T) P \triangleright \Gamma} \right\rrbracket = (\llbracket P \rrbracket : \llbracket T \rrbracket^\perp \parallel \llbracket T \rrbracket \mathrel{+\!\!\!\!\longrightarrow} \llbracket \Gamma \rrbracket) \odot (\text{æ}_{\llbracket T \rrbracket} : \emptyset \mathrel{+\!\!\!\!\longrightarrow} \llbracket T \rrbracket \parallel \llbracket T \rrbracket^\perp)$$

Fig. 5. Interpretation of π_{MALL} into **LDStr**

Equational Theory. Rather than giving an operational semantics to our calculus and showing a correspondence with our game semantics, we give a semantics to our calculus by way of an equational theory axiomatising the natural congruence induced by the model: P and Q are equivalent when $\llbracket P \rrbracket \cong \llbracket Q \rrbracket$. This equational theory can be used to alleviate tedious reasoning on the game model, instead of doing the proof semantically.

To define the equational theory, we rely on open processes, which represent *evaluation contexts* (we avoid calling them context to avoid ambiguities with typing contexts). An **open process** is a process with free process variables (denoted by $X, Y, \ldots$) described by the following grammar:

$$\mathfrak{P} ::= X \mid (\mathfrak{P} \parallel P) \mid (P \parallel \mathfrak{P}) \mid a\{x_1, \ldots, x_n\}.\, \mathfrak{P} \mid a \oplus k.\, \mathfrak{P} \mid a \,\&\, \{i.\, \mathfrak{P}_i\}_{i \in I} \mid (\nu\, a\, b : T)\mathfrak{P}$$

Variables must always occur on one of the two sides of a parallel composition. We write $\mathsf{fc}(\mathfrak{P})$ for the free channels occurring in $\mathfrak{P}$. Open processes can be typed as follows: $X_i : \Delta_i \vdash \mathfrak{P} \triangleright \Gamma$ means that whenever $P_i \triangleright \Delta_i, \Delta$, we have $\mathfrak{P}[X_i := P_i] \triangleright \Gamma, \Delta$. The substitution is often simply written $\mathfrak{P}[P_i]_i$. We also consider particular open processes called **actions**:

$$\mathfrak{a} ::= a\{x_1, \ldots, x_n\}.\, X \mid a \oplus k.\, X \mid a \,\&\, \{i.\, X_i\}_{i \in I}$$

An action can be either a **positive action** (+) ($a \oplus k$), a **negative action** (-) ($a \,\&\, \{i.\, X_i\}_{i \in I}$) or a **naming action** (0) ($a\{x_1, \ldots, x_n\}.X$). To bind an action of the appropriate polarity, we write it in exponent, eg. $\mathfrak{a}^-$ denotes a negative action on channel a. We use such a generic formulation to accommodate the new constructions of § 3 and § 4.

We define the equivalence relation on typing derivations. Let $\equiv$ be the smallest congruence closed under the laws of Figure 6 (where we only consider the instances where both sides are typed in the same context). Let us detail the rules. They are split in several categories:

Compact-closed fragment rules deal with the parallel operation and the restriction. They arise from the compact-closed structure and state that ν commutes with all other constructs and $\parallel$ is a commutative monoid. The last equation states that the forwarder is an identity on processes.

Permutations rules deal with courtesy and receptivity. Courtesy and receptivity induce reorderings: for instance $a \oplus i.\, b \oplus j$ and $a \oplus i. \parallel b \oplus j$ denote the same strategy. The allowed reorderings are positive/positive and negative/negative. We also allow positive actions to permute with parallel composition, however $\mathfrak{a}^-[P \parallel Q]$ is not the same as $\mathfrak{a}^-[P] \parallel Q$ because in the former Q waits for $\mathfrak{a}$ but not in the latter. We use $\mathsf{na}(\mathfrak{a})$ to denote the names occurring in $\mathfrak{a}$, *bound or free*.

Communications define the main equations for communications.

Cut elimination rules allow to reduce restrictions. The first rule represents deadlocks: $(\nu\, a\, b : T)\mathfrak{a}^-[P_i]$ must be empty because any potential output on b is delayed by the input on a, hence

Compact-Closed Structure

$$P \equiv Q \ \text{(if } P =_\alpha Q) \quad P \parallel Q \equiv Q \parallel P \quad (P \parallel Q) \parallel R \equiv P \parallel (Q \parallel R) \quad P \parallel \mathbf{0} \equiv P$$

$$(\nu\, a\, b : T)\mathbf{0} \equiv \mathbf{0} \quad (\nu\, a\, b : T)P \equiv (\nu\, b\, a : T^\perp)P \quad (\nu\, a\, b : T)(\nu\, c\, d : T')P \equiv (\nu\, c\, d : T')(\nu\, a\, b : T)P$$

$$(\nu\, a\, b : T)P \parallel Q \equiv (\nu\, a\, b : T)(P \parallel Q) \ \text{(if } \mathsf{fc}(Q) \cap \{a, b\} = \emptyset)$$

$$(\nu\, a\, b : T)\mathfrak{P}[P_i]_i \equiv \mathfrak{P}[(\nu\, a\, b : T)P_i]_i \ \text{(if } \mathsf{fc}(\mathfrak{P}) \cap \{a, b\} = \emptyset)$$

$$P \equiv (\nu\, b\, b' : T)(P[a := b] \parallel [b' \leftrightarrow a]_T) \ \text{(if } a \in \mathsf{fc}(P))$$

Permutations

$$\mathfrak{P}[\mathfrak{a}^0[P_i]]_i \equiv \mathfrak{a}^0[\mathfrak{P}[P_i]_i] \ \text{(if } \mathsf{na}(\mathfrak{a}) \cap \mathsf{fc}(\mathfrak{P}) = \emptyset) \qquad \mathfrak{a}^+[\mathfrak{b}^+[P]] \equiv \mathfrak{b}^+[\mathfrak{a}^+[P]] \ \text{(if } a \neq b)$$

$$\mathfrak{a}^+[P] \parallel Q \equiv \mathfrak{a}^+[P \parallel Q] \qquad \mathfrak{a}^-[\mathfrak{b}^-[P_{i,j}]_i]_j \equiv \mathfrak{b}^-[\mathfrak{a}^-[P_{i,j}]_j]_i \ \text{(if } a \neq b)$$

Communication

$$(\nu\, a\, b : \oplus_{i\in I} T_i)(a \oplus k.\, P \parallel b \,\&\, \{i.\, Q_i\}_{i\in I}) \equiv (\nu\, a\, b : T_k)(P \parallel Q_k)$$

$$(\nu\, a\, b : T_1 \parallel \ldots \parallel T_n)(a\{x_1, \ldots, x_n\}.\, P \parallel b\{y_1, \ldots, y_n\}.\, Q) \equiv (\nu\, x_1\, y_1 : T_1) \cdots (\nu\, x_n\, y_n : T_n)(P \parallel Q)$$

Cut Elimination

$$(\nu\, a\, b : T)\mathfrak{a}^-[P_i]_i \equiv \mathbf{0} \qquad (\nu\, a\, b : T)(\mathfrak{u}^-[P_i]_i \mid \mathfrak{a}^-[Q_j]_j) \equiv \mathfrak{u}^-[(\nu\, a\, b : T)(P_i \mid \mathfrak{a}^-[Q_j]_j)]_i \ \text{(if } u \notin \{a, b\})$$

Fig. 6. Equational Theory for π_{MALL}

will never occur. Because of the permutation rule, we can always move an initial output outside a restriction. For inputs, it is only allowed when the other parallel threads are receiving on an internal variable. For instance, we can eliminate deadlocks by combining the two rules:

$$\begin{aligned} &(\nu\, a\, b : T)(\nu\, c\, d : T)(a \,\&\, \ell.\, d \oplus \ell \parallel c \,\&\, \ell.\, b \oplus \ell) && \text{(Moving } a \text{ outside the restriction on } c \text{ and } d) \\ \equiv\, &(\nu\, a\, b : T)a \,\&\, \ell.\, (\nu\, c\, d : T)(d \oplus \ell \parallel c \,\&\, \ell.\, b \oplus \ell) && \text{(Restriction on } a/b \text{ now starts with input on } a) \\ \equiv\, &\mathbf{0} \end{aligned}$$

All these rules are sound.

Lemma 2.10 (Soundness). *If $P \equiv Q$, then $[\![P]\!] \cong [\![Q]\!]$.*

2.6 Interpreting an Affine λ-Calculus in LDStr

As the first example of our methodology, we show how to translate an affine λ-calculus into LDStr, through a syntactic translation into π_{MALL}. The resulting model characterises $\beta\eta$-equivalence. Types of affine λ-calculus are given by the grammar: $\tau ::= \mathbb{B} \mid \tau \multimap \tau$. The affine typing system has the following standard rules:

$$\frac{\Gamma, x : \tau_1 \vdash M : \tau_2}{\Gamma \vdash \lambda x.\, M : \tau_1 \multimap \tau_2} \qquad \frac{\Gamma \vdash M_1 : \tau_1 \multimap \tau_2 \quad \Delta \vdash M_2 : \tau_1}{\Gamma, \Delta \vdash M_1\, M_2 : \tau_2} \qquad \frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau} \qquad \frac{b \in \{\mathrm{tt}, \mathrm{ff}\}}{\Gamma \vdash b : \mathbb{B}} \qquad \frac{\Gamma_1 \vdash M : \mathbb{B} \quad \Gamma_2 \vdash N : \mathbb{B} \quad \Gamma_2 \vdash N' : \mathbb{B}}{\Gamma_1, \Gamma_2 \vdash \mathsf{if}\, M\, N\, N' : \mathbb{B}}$$

To translate types, we follow the standard tradition of call-by-name game semantics of interpreting types by negative games. Syntactically, we thus interpret a type τ by a session type of the form $\&_{i\in I} T_i$. Such types support an interpretation of the linear arrow:

$$S \multimap (\&_{i\in I} T_i) \overset{\text{def}}{=} \&_{i\in I}(S^\perp \parallel T_i)$$

Types of the λ-calculus are translated as follows: $(\!|\, \mathbb{B} \,|\!) \overset{\text{def}}{=} \mathsf{q}^- \cdot \oplus_{b\in\{\mathrm{tt},\mathrm{ff}\}} \mathbf{1}$ and $(\!|\, \tau \multimap \tau' \,|\!) \overset{\text{def}}{=} (\!|\, \tau \,|\!) \multimap (\!|\, \tau' \,|\!)$, and typing contexts as: $(\!|\, x_1 : \tau_1, \ldots, x_n : \tau_n \,|\!) \overset{\text{def}}{=} x_1 : (\!|\, \tau_1 \,|\!), \ldots, x_n : (\!|\, \tau_n \,|\!)$.

For example, the session type associated to linear functions from booleans to booleans is:

$$(\!|\, \mathbb{B} \multimap \mathbb{B} \,|\!) = (\!|\, \mathbb{B} \,|\!) \multimap (\!|\, \mathbb{B} \,|\!) = \mathsf{q}^- \cdot (\mathsf{q}^- \cdot (\mathrm{tt}^- \,\&\, \mathrm{ff}^-) \parallel \mathrm{tt}^+ \oplus \mathrm{ff}^+)$$

Translation of Terms

$$(\!| \lambda x.\, M : \tau_1 \multimap \tau_2 |\!)^o_i \overset{\text{def}}{=} o\{x, o\}.\, (\!| M |\!)^o_i \qquad (\!| x : \tau |\!)^o_i \overset{\text{def}}{=} x \oplus i.\, [x \leftrightarrow a]_{\tau/i} \qquad (\!| b : \mathbb{B} |\!)^o_{\mathsf{q}} \overset{\text{def}}{=} o \oplus b$$

$$(\!| M_1\, M_2 : \tau_2 |\!)^o_i \overset{\text{def}}{=} ((\!| M |\!)^o_i)\{o{:=}x, o\} \;\; {}_x\odot_y \;\; (\!| N |\!)^y$$

$$(\!| \mathsf{if}\ M\ N\ N' : \mathbb{B} |\!)^o_i \overset{\text{def}}{=} (\!| M |\!)^x_{\mathsf{q}} \;\; {}_x\odot_y \;\; y \,\&\, \{\mathrm{tt} : (\!| N |\!)^o_i ; \mathrm{ff} : (\!| N' |\!)^o_i\}$$

Macros

Name	Definition	Type	Given
$(P)\{a{:=}x, y\}$	$\nu a\, b.\, (P \parallel b\{x', y'\}.([x \leftrightarrow x'] \parallel [y \leftrightarrow y']))$	$x : T_1, y : T_2$	$P \triangleright a : T_1 \parallel T_2$
$P \;{}_a\odot_b\; Q$	$(\nu\ a\, b : \_)(P \parallel Q)$	Γ, Δ	$P \triangleright \Gamma, a : T \;\; ; Q \triangleright \Delta, b : T^\perp$

Fig. 7. Translation from the affine λ-calculus to π_{MALL}

The function awaits for the q signal from the environment, and then has a choice: either it returns directly a boolean — it is a non-strict function — or it can ask Context for its argument value before returning its result.

For terms, the idea is to translate a typing derivation with conclusion $\Gamma \vdash M : \tau$ into a typing derivation of π_{MALL} whose conclusion is: $(\!| t |\!)^o \triangleright (\!| \Gamma |\!)^\perp, o : (\!| \tau |\!)$, where o, the name used to communicate with Context about the return value, is a parameter of the translation.

However, building directly $(\!| t |\!)^o$ by induction proved tedious so we use a slightly stronger inductive invariant. Indeed, the processes that we will build are all *negative* as well, ie. of the form $o \,\&\, \{P_i\}_{i \in I}$ where $(\!| \tau |\!) = \&_{i \in i} T_i$. The set I is called the set of *computation types* of τ, written $\mathrm{ct}(\tau)$, and we write $\tau / i \overset{\text{def}}{=} T_i$ (for now $I = \{\mathsf{q}\}$ — but this property will not remain true in the extension in § 4.3). It turns out that constructing P_i is simpler, so from a term $\Gamma \vdash M : \tau$, we build a family of processes $(\!| \Gamma \vdash M : \tau |\!)^o_i \triangleright (\!| \Gamma |\!)^\perp, o : \tau / i$ where $i \in \mathrm{ct}(\tau)$. Those processes are put together as follows:

$$(\!| \Gamma \vdash M : \tau |\!)^o \overset{\text{def}}{=} o \,\&\, \{i.\, (\!| \Gamma \vdash M : \tau |\!)^o_i\}_{i \in \mathrm{ct}(\tau)}\,.$$

The definition of the translation of a linear λ-terms (Figure 7) is now straightforward, by induction on its structure. From this syntactic translation, we can derive a semantic interpretation of types and λ-terms as games and strategies as follows:

$$[\![\tau]\!] = [\![(\!| \tau |\!)]\!] \qquad\qquad [\![M]\!] = [\![(\!| M |\!)^a]\!] \quad \text{for some } a.$$

Correctness of the Encoding. We now show that our translation characterises $\beta\eta$-equivalence, defined as the smallest congruence on typed λ-terms containing the well-typed instances of the rules:

$$\Gamma \vdash (\lambda x.\, t)\, N =_{\beta\eta} t[x := N] : B \qquad\qquad \Gamma \vdash M =_{\beta\eta} (\lambda x.\, M\, x) : A \multimap B$$

$$\Gamma \vdash \mathsf{if}\ b\ a_{\mathrm{tt}}\ a_{\mathrm{ff}} =_{\beta\eta} a_b : \mathbb{B} \qquad\qquad \Gamma \vdash M =_{\beta\eta} \mathsf{if}\ M\ \mathrm{tt}\ \mathrm{ff} : \mathbb{B}$$

In the third rule, $b \in \{\mathrm{tt}, \mathrm{ff}\}$ and in the second x is not free in t.

Theorem 2.11 (Soundness and Completeness). *We have $M =_{\beta\eta} N$ if and only if $(\!| M |\!)^a \equiv (\!| N |\!)^a$ if and only if $[\![M]\!] \cong [\![N]\!]$.*

Proof. The forward direction is by induction on the rules defining $\beta\eta$. A key property to use is the negativity of $(\!| M |\!)^a$: it always starts by an input on a by construction. The converse direction relies on the confluence and normalisation of the calculus. Every term M is $\beta\eta$-equal to its η-long β-normal form. If M, N are in such a normal forms, then we prove: $M =_{\beta\eta} N$ iff $(\!| M |\!)^a \equiv (\!| N |\!)^a$ iff $[\![M]\!] \cong [\![N]\!]$. See Appendix A.3 for details. □

3 NONLINEAR AND DETERMINISTIC GAMES

Strategies presented in § 2.3 are linear: each move of the game can be played at most once in an execution. As moves correspond to function calls, we are restricted to interpreting linear languages.

In this section, we extend our framework to handle nonlinear computations, following the methodology of Castellan et al. [2015, 2019]. In § 3.1, we present the challenge of extending the previous setting to deal with non-linearity. In § 3.2, we recall event structures with symmetry, the technical tool used to solve this issue. In § 3.3, we present how to build a category out of these ideas. In § 3.4, we present π_{LL}, an extension of π_{MALL} with the exponentials of Linear Logic to define nonlinear strategies. In § 3.5 and § 3.6, we look at encodings of higher-order languages into **DStr** following game semantics methodology, both in call-by-name and call-by-value.

3.1 Non-linearity in Game Semantics

A standard procedure to interpret nonlinear computations inside a linear model is given by Linear Logic [Girard 1987]. The main insight is that for each type A, there should be two types $!A$ and $?A$ allowing repeated invocations of A. In this setting, an object of type $!A$ should be seen as producing infinitely many A's, while an object of type $?A$ as producing finitely (possibly zero) A's. This is well-behaved when it comes to interaction since a $!(A^\perp)$ is a morally a "server", able to consume infinitely many A's, while $?A$ is a client, issuing several requests of type A.

Seely [1989] defines the algebraic structure required on these new objects. By duality, it is enough to describe the structure for one of those two connectives. First $?A$ should be a monad conform to the intuition that $?A$ is a bundle of many A's; and moreover it must turn any object into a commutative comonoid, i.e. for every A, it must come equipped with a map $?A \parallel ?A \to ?A$ explaining how to join bags together (and a unit $1 \to ?A$, the empty bag).

Before building such structure in games and strategies, let us look at the category of event structures first (indeed, most of the structure can be lifted to strategies). In event structures, a candidate for "infinite many E's", is the following construction. Given an event structure E, we define $\sharp E$ as $\parallel_{i\in\mathbb{N}} E$. In particular, its events are pairs (i, e) of a move $e \in E$ and an integer $i \in \mathbb{N}$, referred to as a **copy index**. There are natural candidates for a monad structure on $\sharp(-)$:

$$\eta_E : (a \in E) \mapsto (0, a) \in \sharp E \qquad\qquad \mu_E : \big((i, (j, a)) \in \sharp\sharp E\big) \mapsto \big((\langle i, j\rangle, a) \in \sharp E\big)$$

where $\langle \cdot, \cdot \rangle : \mathbb{N}^2 \simeq \mathbb{N}$ is any bijection. To satisfy the monadic laws, we must have $\langle 0, i\rangle = \langle i, 0\rangle = i$ for all $i \in \mathbb{N}$, which is incompatible with $\langle \cdot, \cdot \rangle$ being a bijection. There is in fact no hope of realising these laws "on the nose", i.e. up to function equality. Instead, the equation holds *up to copy indices*. This is reasonable: the precise identity of copy indices should not affect the semantics since they are an artefact of the model. To formalise the notion of equality "up to copy indices", we equip $\sharp E$ with an isomorphism family, turning it into an event structure with symmetry.

3.2 Event Structures with Symmetry

The main idea of event structures with symmetry introduced by Winskel [2007] is not to look at events individually but at configurations instead.

Definition 3.1. An **isomorphism family** on an event structure E is a family $\tilde{E}$ of order-isomorphisms $\varphi : x \cong y$ with $x, y \in \mathscr{C}(E)$ such that

(1) $\tilde{E}$ contains all identities, and is stable under inverse and composition
(2) If $x \subseteq x'$ and $\varphi : x' \cong y' \in \tilde{E}$, then $\varphi|_x : x \cong \varphi x \in \tilde{E}$
(3) If $x \subseteq x'$ and $\varphi : x \cong y \in \tilde{E}$, then there exists a (non-necessarily unique) $y' \in \mathscr{C}(E)$ such that φ extends to $\varphi' : x' \cong y' \in \tilde{E}$.

An **event structure with symmetry** is a pair $(E, \tilde{E})$ of an event structure E and an isomorphism family $\tilde{E}$ on E. We write $\mathcal{E}, \mathcal{F}, \ldots$ for event structures with symmetry.

Most constructions on event structures (such as parallel composition, sum and lifting) can be extended seamlessly to event structures with symmetry [Winskel 2007]. Given an event structure with symmetry $\mathcal{E}$, define an isomorphism family $\sharp\tilde{E}$ on $\sharp E$ containing all the $\theta : x \cong y$ for which there exists $\pi : \mathbb{N} \to \mathbb{N}$ such that

(1) for $(i, e) \in x$, $\theta\,(i, e)$ is of the form $(\pi\, i, e')$ and (2) $\{(e, e') \mid \theta(i, e) = (\pi\, i, e')\} \in \tilde{E}$.

We define the event structure with symmetry $\sharp\mathcal{E}$ as $(\sharp E, \sharp\tilde{E})$.

A **map of event structures with symmetry** $f : \mathcal{E} \rightharpoonup \mathcal{F}$ is a map $f : E \rightharpoonup F$ such that for all $\theta \in \tilde{E}$, $f\,\theta \overset{\text{def}}{=} \{(f\,e, f\,e') \mid (e, e') \in \theta\}$ is in $\tilde{E}$. Two such maps $f, g : \mathcal{E} \to \mathcal{F}$ are said to be **similar** if for all $x \in \mathcal{C}(E)$, $\{(f\,s, g\,s) \mid s \in x\} \in \tilde{E}$, which we write $f \sim g$. Event structures with symmetry and total maps form a category **ESS**. This equivalence relation is enough to formalise this idea of equality up to copy indices.

Lemma 3.2. *$\sharp(-) :$ **ESS** $\to$ **ESS** is a monad up to $\sim$.*

3.3 The Category DStr

Games with Symmetry. To be able to identify strategies up to copy indices, we need more structure than a simple symmetry on A, but the structure of a thin concurrent game [Castellan et al. 2019].

Definition 3.3. A **thin concurrent game** (tcg) is a tuple $\mathcal{A} = (A, \tilde{A}, \tilde{A}_+, \tilde{A}_-)$ where A is a game, and $\tilde{A}, \tilde{A}_+, \tilde{A}_-$ are isomorphism families on A subject to axioms listed in [Castellan et al. 2019]; in particular $\tilde{A}_-$ and $\tilde{A}_+$ are sub-isomorphism families of $\tilde{A}$.

The intuition is that $\tilde{A}_-$ (resp. $\tilde{A}_+$) contain only isomorphisms that affect negative (resp. positive) events. This idea of polarised decomposition of the symmetry was first introduced by Melliès [2003] in a simpler setting. Games operations (parallel composition, dual) extend to tcgs.

Uniform Strategies. Symmetry on games induces naturally an equivalence relation on strategies, relaxing isomorphism by asking that the triangle commutes up to symmetry on the game. This equivalence, however, is not a congruence since nothing prevents a strategy to observe copy indices from Opponent. To recover a well-behaved compositional setting, we need to ensure that the strategy we consider behaves uniformly with respect to copy indices from Opponent. It turns out that we need to add extra structures on strategies, under the form of an isomorphism family.

Definition 3.4. Consider a tcg $\mathcal{A}$. A **uniformity witness** for $\sigma : S \to A$ is an isomorphism family $\tilde{S}$ on S such that:

(1) σ becomes a map of event structures with symmetry $(S, \tilde{S}) \to (A, \tilde{A})$;
(2) if $\theta : x \cong y \in \tilde{S}$ and $\sigma\,\theta$ extends to $\varphi : x' \cong y' \in \tilde{A}$ with $x \subseteq^- x'$, then θ extends to a θ' such that $\sigma\,\theta' = \varphi$;
(3) if $\theta : x \cong y \in \tilde{S}$ is the identity on negative elements of x, then θ is the identity on x.

A **$\sim$-strategy** on $\mathcal{A}$ is a strategy σ on A along with a uniformity witness for σ.

We can lift the equivalence relation $\sim$ on maps to a weak isomorphism on strategies: two $\sim$-strategies $\sigma : S \to A$ and $\tau : T \to A$ are **weakly isomorphic** (written $\sigma \approxeq \tau$) when there exists an isomorphism of event structures with symmetry $\varphi : \mathcal{S} \cong \mathcal{T}$ such that $\tau \circ \varphi \sim \sigma$.

Theorem 3.5 ([Castellan et al. 2019]). *Uniformity witnesses compose and weak isomorphism is a congruence on $\sim$-strategies. As result, tcgs and $\sim$-strategies up to weak isomorphism form a compact-closed category* **DStr**.

The Lifting Monad. Unfortunately, if $\mathcal{A}$ is a tcg, $\sharp\mathcal{A}$ is not a tcg is general. However, if $\mathcal{A}$ is polarised then $\sharp\mathcal{A}$ is also tcg [Castellan et al. 2019]. To work around this issue, we lift games before taking $\sharp(-)$ which leads to the two natural exponentials of Linear Logic: $!\mathcal{A}$ (resp. $?\mathcal{A}$) when lifted with a negative (resp. positive) move.

Formally, liftings of games are defined as follows. Given a tcg $\mathcal{A}$, we define $\uparrow^-\mathcal{A}$ to be the tcg $\mathbf{box}^- \cdot \mathcal{A}$. This operation on tcgs extends to a functor $\uparrow^- - : \mathbf{DStr} \to \mathbf{DStr}$: given $\sigma : \mathcal{A} \to\!\!\!\!\!+\, \mathcal{B}$, $\uparrow^-\sigma$ starts by acknowledging the negative box on the right, before playing box on the left, and continuing as σ. As a result, $\uparrow^-\sigma$ is a negative strategy. Writing $\mathbf{DStr}^-$ for the subcategory of $\mathbf{DStr}$ whose games are negative with negative strategies on it, we have:

Lemma 3.6. *The functor $\uparrow^-(-)$ is the right adjoint to the inclusion $\mathbf{DStr}^- \to \mathbf{DStr}$. In particular, $\uparrow^-$ transports comonads on $\mathbf{DStr}^-$ to comonads on $\mathbf{DStr}$.*

By duality, we define similarly $\uparrow^+\mathcal{A}$ which has a dual adjunction result.

Exponentials on $\mathbf{DStr}$. On $\mathbf{DStr}^-$, $\sharp(-)$ has the desired structure:

Lemma 3.7 ([Castellan et al. 2019]). *$\sharp(-) : \mathbf{DStr}^- \to \mathbf{DStr}^-$ is an* ***exponential comonad****, i.e. in this affine setting, a comonad equipped with a natural transformation, the* ***contraction****, $c_{\mathcal{A}} : \sharp\mathcal{A} \to\!\!\!\!\!+\, \sharp\mathcal{A} \parallel \sharp\mathcal{A}$ which turns $\sharp\mathcal{A}$ into a comonoid.*

The exponential comonad $\sharp(-)$ induces, through the lifting adjunction, an exponential comonad $!(-)$: $!A = \sharp\uparrow^- A$. By duality, we get the exponential monad $?\mathcal{A} \overset{\text{def}}{=} \sharp\uparrow^+\mathcal{A}$. The unit of the monad $?(-)$ gives the **dereliction** strategy $d_{\mathcal{A}} : \mathcal{A} \to\!\!\!\!\!+\, ?\mathcal{A}$, forwarding $\mathcal{A}$ onto the zeroth copy of $\mathcal{A}$ in $!\mathcal{A}$; and the multiplication of the comonad $!(-)$ gives the **digging** strategy $\mu_{\mathcal{A}} : !\mathcal{A} \to\!\!\!\!\!+\, !!\mathcal{A}$, forwarding $(\langle i, j\rangle, a)$ onto $(i, (j, a))$.

3.4 A Metalanguage for DStr: π_{LL}

To build a calculus for $\mathbf{DStr}$, we extend the calculus of § 2.4 with the exponentials of Linear Logic:

$$T ::= \ldots \mid\ !T \mid\ ?T \qquad P ::= \cdots \mid a!(x).\,P \mid a?[x].\,P \mid a?\{x, y\}.\,P$$

The type $!T$ represents a *server* waiting for clients following protocol T and the type $?T$ is for a *client* connecting to a server that follows protocol T. The process $a!(x).\,P$ creates a *server* listening on a; spawning a copy of P for every request x made; $a?[x].\,P$ is a *client* which connects to server a and continues as P with channel x meant to perform the request. The process $a?\{x, y\}.\,P$ is a *duplication*: it duplicates the name a into two names x and y and executes P: the two names will be connected to the same server but can issue requests in parallel to the server. This construction allows the non-affine use of channels of type $?T$. Note that $\mathrm{fc}(a?\{x, y\}.\,P) = \mathrm{fc}(P) \setminus \{x, y\}$. Typing judgements are extended with the corresponding rules of Linear Logic:

$$\textsc{Server}\ \frac{P \triangleright ?\Gamma, x : T}{a!(x).\,P \triangleright ?\Gamma, a : !T} \qquad \textsc{Client}\ \frac{P \triangleright \Gamma, x : T}{a?[x].\,P \triangleright \Gamma, a : ?T} \qquad \textsc{Dup}\ \frac{P \triangleright \Gamma, x : ?T, y : ?T}{a?\{x, y\}.\,P \triangleright \Gamma, a : ?T}$$

Rule Server ensures that all the free names of an exponential input are client names (denoted by $?\Gamma$); rule Client types a request (once) to a server; and Rule Dup types a duplicator where the type of x, y is the same as a. The forwarder can naturally be extended:

$$[a \leftrightarrow b]_{!T} = a!(x).\,b?[y].\,[x \leftrightarrow y]_T.$$

Communication

$$(\nu\, a\, b : !T)(a!(x).\, P \parallel b?[y].\, Q) \equiv (\nu\, x\, y : T)(P \parallel Q)$$

$$(\nu\, a\, b : !T)(a!(x).\, P \parallel b?\{b_1, b_2\}.\, Q) \equiv (\nu\, a_1\, b_1 : !T)(\nu\, a_2\, b_2 : !T)((a_1!(x).\, P \parallel_\Delta a_2!(x).\, P) \parallel Q)$$
$$(\text{if } \mathsf{fc}(P) \setminus \{x\} = \Delta, (\mathsf{fc}(P) \cup \mathsf{fc}(Q)) \cap \{a_1, a_2\} = \emptyset)$$

Contractions

$$a?\{a_1, a_2\}.\, P \equiv a?\{a_2, a_1\}.\, P \qquad a?\{b, a'\}.\, a'?\{c, d\}.\, P \equiv a?\{a', d\}.\, a'?\{b, c\}.\, P$$
$$a?\{b, c\}.P \equiv P[a/b] \ (\text{if } c \notin \mathsf{fc}(P))$$

Fig. 8. Equational Theory for π_{LL}

Infinitary terms. Types now denote infinite games, and as a result the model contains infinite strategies. The promotion construction allows to define some infinite strategies, but it is not enough. To allow arbitrary infinite behaviours, we consider *infinite terms* with *infinite typing derivations*. Formally, we consider the free ω-CPO over the set of typing derivations, ordered by prefix. This will be useful to interpret the fixpoint operator of PCF and the memory allocation of IPA.

Interpretation of the Calculus. Types are interpreted by the corresponding operations in **DStr** homomorphically: $[\![!T]\!] = ![\![T]\!]$ and $[\![?T]\!] = ?[\![T]\!]$. The interpretation of the new process constructions follows the standard categorical semantics of Linear Logic:

$$\left[\!\!\left[\frac{P \triangleright ?\Gamma, x : T}{a!(x).\, P \triangleright ?\Gamma, a : !T}\right]\!\!\right] = (![\![P]\!] : !![\![\Gamma]\!]^\perp \twoheadrightarrow ![\![T]\!]) \odot \mu_{[\![\Gamma]\!]^\perp} \quad \left[\!\!\left[\frac{P \triangleright \Gamma, x : T}{a?[x].\, P \triangleright \Gamma, a : ?T}\right]\!\!\right] = d_{[\![T]\!]} \odot ([\![P]\!] : [\![\Gamma]\!]^\perp \twoheadrightarrow [\![T]\!])$$
$$\left[\!\!\left[\frac{P \triangleright \Gamma, x : ?T, y : ?T}{a?\{x, y\}.\, P \triangleright \Gamma, a : ?T}\right]\!\!\right] = c_{[\![T]\!]} \odot ([\![P]\!] : [\![\Gamma]\!]^\perp \twoheadrightarrow ?[\![T]\!] \parallel ?[\![T]\!])$$

The interpretation of infinitary terms follows from this result:

Lemma 3.8 ([Castellan et al. 2015]). *For any tcg A, $\sim$-strategies over A have a natural order which turns* **DStr**(A) *into an ω-CPO.*

The interpretation of finite terms of π_{MALL} into **DStr** is monotonic for this order, and as a result, can be uniquely extended by continuity to infinite terms.

Equational Theory. We extend open processes as follows:

$$\mathfrak{P} ::= \cdots \mid a!(x).\, \mathfrak{P} \mid a?\{a_1, a_2\}.\, \mathfrak{P} \mid a?[x].\, \mathfrak{P}$$

The action $a?\{a_1, a_2\}.\, X$ is a naming action; $a!(x).\, X$ is a negative one; and $a?[x].\, X$ is positive. We use the following macro to manage context duplication, where both processes have access to the same channels

$$P \parallel_\Gamma Q \overset{\text{def}}{=} \vec{x}?\{\vec{a}, \vec{b}\}.\ \left(P[\vec{x} := \vec{a}] \parallel Q[\vec{x} := \vec{b}]\right) \quad \text{such that} \quad \frac{P \triangleright ?\Gamma, \Delta_1 \quad Q \triangleright ?\Gamma, \Delta_2}{P \parallel_\Gamma Q \triangleright ?\Gamma, \Delta_1, \Delta_2}$$

We also use $P[a^?]$ for $a?[x].\, P[x]$ to limit the noise added by derelictions.

In **Communication**, the first rule is defined as a usual reduction for the πI-calculus [Sangiorgi 1996]; the second rule creates two fresh servers which are accessed by Q. In **Contractions**, the rules are there to reflect the fact that contraction induces a comonoid structure.

Note that the communication rule for $!/?$ makes the server disappear, unlike the usual replication rules in the π-calculus. The typing rules ensure that there are no occurrences of a in Q: the server will never be contacted again and hence can safely be shut down. The only way to contact a server twice is via the use of the contraction rule, which will duplicate the server accordingly.

Lemma 3.9 (Soundness). *If $P \equiv Q$, then $[\![P]\!] \cong [\![Q]\!]$.*

Translation of Terms

$$\begin{aligned}
(\!| \Gamma, x : \tau \vdash x : \tau |\!)_i^o &\overset{\text{def}}{=} x?[u].\, x \oplus i.\, [u \leftrightarrow o]_{\tau/i} \\
(\!| \Gamma \vdash \lambda x.\, M : \tau_1 \longrightarrow \tau_2 |\!)_i^o &\overset{\text{def}}{=} o\{x, o\}.(\!| M |\!)_i^o \\
(\!| M\, N : \tau_2 |\!)_i^o &\overset{\text{def}}{=} ((\!| M |\!)_i^o)\{o{:=}x, o\} \ {}_x\!\odot_y^{(\!| \Gamma |\!)} \ x!(r).\, (\!| N |\!)^r \\
(\!| \mathsf{if}\ M\, N\, N' : \tau |\!)_i^o &\overset{\text{def}}{=} (\!| M |\!)_{\mathrm{q}}^x \ {}_x\!\odot_y^{(\!| \Gamma |\!)} \ y \,\&\, \left\{ \mathrm{tt} : (\!| N |\!)_i^o ; \mathrm{ff} : (\!| N' |\!)_i^o \right\} \\
(\!| \mathbf{Y} : (\tau \longrightarrow \tau) \longrightarrow \tau |\!)_i^o &\overset{\text{def}}{=} o\{f, u\}.\, \mathsf{fix}_i^{f,u} \\
(\!| \mathsf{iszero} : \mathbb{N} \longrightarrow \mathbb{B} |\!)_{\mathrm{q}}^o &\overset{\text{def}}{=} o\{a, o\}.\, a \oplus \mathrm{q}.\, a \,\&\, \begin{cases} 0 : o \oplus \mathrm{tt} \\ n+1 : o \oplus \mathrm{ff} \end{cases} \\
(\!| \mathsf{succ} : \mathbb{N} \longrightarrow \mathbb{N} |\!)^o &\overset{\text{def}}{=} o\{a, o\}.\, a \oplus \mathrm{q}.\, a \,\&\, \{ n : o \oplus n+1 \}_{n \in \mathbb{N}}
\end{aligned}$$

Macros

Name	Definition	Type	
$P \ {}_a\!\odot_b^\Gamma \ Q$	$(\nu\, a\, b : \_)(P \parallel_\Gamma Q)$	$?\Gamma, \Delta_1, \Delta_2$	given $P \triangleright ?\Gamma, \Delta_1, a : T$ $\quad Q \triangleright ?\Gamma, \Delta_2, b : T^\perp$
$\mathsf{fix}_i^{u,f}$	$f?\{f_1, f_2\}.\, f_1^? \oplus i.\, f_1\{a, u'\}.$ $([u \leftrightarrow u'] \parallel a!(x).\, x \,\&\, \left\{ j.\, \mathsf{fix}_j^{x, f_2} \right\})$	$f : ?((\!| \sigma \rightarrow \sigma |\!)^\perp), u : \sigma/i$	

Fig. 9. Translation from call-by-name PCF to π_{LL}

3.5 Interpretation of Call-by-Name PCF in DStr

We show how to translate a non-linear language, PCF, into π_{LL}. The grammar of types and terms extends that of the affine λ-calculus.

$$\tau ::= \cdots \mid \mathbb{N} \qquad M ::= \cdots \mid \mathtt{n} \in \mathbb{N} \mid \mathbf{Y} \mid \mathtt{iszero}(M) \mid \mathtt{succ}(M)$$

The typing rules of PCF are standard. We write $\rightarrow_{\mathsf{PCF}}$ for the standard call-by-name reduction on terms of PCF. Due to the non-linearity, we change the interpretation of the arrow to use an exponential on the argument:

$$(\!| \tau_1 \longrightarrow \tau_2 |\!) = !(\!| \tau_1 |\!) \multimap (\!| \tau_2 |\!) \qquad (\!| \mathbb{N} |\!) = \mathrm{q}^- \cdot \oplus_{n \in \mathbb{N}} \mathbf{1} \qquad (\!| \mathbb{B} |\!) = \mathrm{q}^- \cdot \oplus_{b \in \{\mathrm{tt}, \mathrm{ff}\}} \mathbf{1}.$$

The translation on terms is then updated to take non-linearity into account, following the usual encoding of call-by-name computation into Linear Logic. It is described in Figure 9. This translation is similar to the one for the affine λ-calculus: apart from adding a promotion in the definition of application, and a dereliction for the variable case, we simply replaced instances of ${}_x\!\odot_{\bar{x}}$ with its non-linear counterpart.

Correctness. As before, we have the substitution lemma:

Lemma 3.10. $(\!| M_1[x := M_2] |\!)^o \equiv (\!| M_1 |\!)^o \ {}_x\!\odot_y^\Gamma \ y!(r).\, (\!| M_2 |\!)^r$.

Proposition 3.11 (Soundness and adequacy). *Given two terms* $\Gamma \vdash M, N : \tau$, *if* $M \rightarrow_{PCF} N$ *then* $(\!| M |\!)^o \equiv (\!| N |\!)^o$. *Moreover if* $(\!| M |\!)^o \equiv (\!| n |\!)^o$ *for* $\vdash M : \mathbb{N}$, *then* $M \rightarrow_{PCF} n$.

Proof. The proof of soundness is essentially the same as the affine case, using the fact that programs are interpreted by negative strategies. Adequacy is first proved on finite terms (terms without Y but with immediate divergence), and then lifted to the infinite terms by continuity, following standard methods. The proof is in Appendix B.2 $\square$

Our methodology inherits the flexibility of game semantics. For instance, for PCF, we know that implementing `if` concurrently, by evaluating the condition and the branches in parallel leads also to a sound and adequate model, due to the absence of state. We can describe such an interpretation here by simply changing the interpretation of conditionals as follows:

$$(\!| \mathsf{if}\ M\ N_1\ N_2 |\!)^o_i = (\nu u\, v; x_1\, y_1; x_2\, y_2) \begin{pmatrix} (\!| M |\!)^u_{\mathsf{q}} \parallel_{(\!|\Gamma|\!)} (\!| N_1 |\!)^{x_1}_i \parallel_{(\!|\Gamma|\!)} (\!| N_2 |\!)^{x_2}_i \\ \parallel v \,\&\, \{\mathsf{tt} : [y_1 \leftrightarrow o]; \mathsf{ff} : [y_2 \leftrightarrow o]\} \end{pmatrix}$$

The return value of M is only used to know what value to return on o: that of N_1 or that of N_2.

3.6 Interpretation of Call-by-Value Languages in DStr

As an aside, to show that different encoding of types can lead to different calling conventions, we show how to interpret a call-by-value simply-typed λ-calculus. For simplicity and space reason, we consider here PCF where the fixpoint combinator has been replaced by a formal divergence $\bot : A$. The fixpoint combinator in call-by-value is more involved than in call-by-name and this simplification is sufficient for demonstrating our framework. We use a standard call-by-value reduction whose main rules are:

$$\frac{V \text{ is a value}}{(\lambda x.\, M)\, V \to_v M[x := V]} \qquad \frac{b \in \{\mathsf{tt}, \mathsf{ff}\}}{\mathsf{if}\ b\ M_{\mathsf{tt}}\ M_{\mathsf{ff}} \to_v M_b} \qquad \frac{M \to_v N}{E[M] \to_v E[N]} \qquad + \text{ rules for } \mathtt{iszero} \text{ and } \mathsf{succ}$$

where values and evaluation contexts are defined as follows:

$$V ::= n \mid b \mid \lambda x.\, M \qquad E ::= [] \mid E\, V \mid M\, E \mid \mathsf{if}\ E\ M\ N \mid \mathtt{iszero}(E) \mid \mathsf{succ}(E).$$

The main difference between the game semantics interpretation of call-by-name and call-by-value computations is the polarity of types. In call-by-name, types become negative games waiting for a message from Context before starting to evaluate; while in call-by-value, types become positive games which can directly produce a result.

Types of this language are interpreted as follows:

$$(\!| \mathbb{B} |\!) = \oplus_{b \in \{\mathsf{tt},\mathsf{ff}\}} \mathbf{1} \qquad (\!| \mathbb{N} |\!) = \oplus_{n \in \mathbb{N}} \mathbf{1} \qquad (\!| \oplus_{i \in I} A_i \to B |\!) = \mathsf{lam}^+ \cdot !\&_{i \in I}(A_i^\perp \parallel B).$$

The translation of arrow types is more involved than in call-by-name. The first move is played when the term has evaluated to a closure. This is used to distinguish $\lambda x.\bot$ and $\bot$; only the first term will play lam^+. We then allow Context to call this closure as many times as it wants (via the exponential !). To call the closure, Context has to provide an argument value ($\&_{i \in I}$), and then Program can ask further questions about its argument (playing in A_i) or provide information about the return value (playing in B). For instance, the type of functions from integers to Booleans is interpreted as the game $\mathsf{lam}^+ \cdot !(\&_{n \in \mathbb{N}} \cdot (\mathsf{tt}^+ \oplus \mathsf{ff}^+))$.

Remark that for all type τ, we have $(\!| \tau |\!) = \oplus_{i \in i} T_i$. The set I is the set of *value types* of τ, written $\mathsf{vt}(\tau)$. Given $i \in \mathsf{vt}(\tau)$, we write τ/i for the type T_i. This notation extends to contexts: we write $\mathsf{vt}(\Gamma)$ for the set of families $(k_x)_{x \in \mathsf{dom}(\Gamma)}$ where $k_x \in \mathsf{vt}((\!| \Gamma(x) |\!))$. Given such a family (k_x), we write $\Gamma/(k_x)$ for the *metalanguage context* $(x : \Gamma(x)/k_x)_{x \in \mathsf{dom}(\Gamma)}$.

We need this machinery because in call-by-value, terms receive values from the context before starting to execute. As a result, the translation works as follows: From a term $\Gamma \vdash M : \tau$, we build by induction a family of processes $(\!| \Gamma \vdash M : \tau |\!)^o_{\vec{k}} \triangleright (\Gamma/\vec{k})^\perp, o : (\!| \tau |\!)$ indexed over value types of Γ. The translation is described in Figure 10.

We can put these processes together using a specific tensor product defined as follows: $(\oplus_{i \in I} A_i) \otimes (\oplus_{j \in J} B_j) = \oplus_{i,j \in I \times J}(A_k \parallel B_l)$. For instance, $\mathbb{B} \otimes \mathbb{B}$ is really a pair of booleans. We write $\otimes\Gamma$ for the

Translation of Terms

$$\begin{aligned}
\llparenthesis \Gamma \vdash \bot : A \rrparenthesis^{o}_{\vec{k}} &\overset{\text{def}}{=} \mathbf{0} \\
\llparenthesis \Gamma, x \vdash x : \sigma \rrparenthesis^{o}_{\vec{k}} &\overset{\text{def}}{=} o \oplus k_x .\, [x \leftrightarrow o]_{\sigma / k_x} \\
\llparenthesis \Gamma \vdash \lambda x.\, M : \sigma \to \tau \rrparenthesis^{o}_{\vec{k}} &\overset{\text{def}}{=} \lambda^{\sigma}_{x \to o}.\, (\llparenthesis \Gamma, x : \sigma \vdash M : \tau \rrparenthesis^{o}_{\vec{k}, x : v})_{v \in \mathrm{vt}(\sigma)} \\
\llparenthesis \Gamma : M\, N : \tau \rrparenthesis^{o}_{\vec{k}} &\overset{\text{def}}{=} \mathbf{app}^{\text{cbv}}_{x : \sigma \to o} \left(\llparenthesis M \rrparenthesis^{x}_{\vec{k}}, \llparenthesis N \rrparenthesis^{o}_{\vec{k}} \right) \qquad (\Gamma \vdash M : \sigma) \\
\llparenthesis \Gamma \vdash \mathsf{if}\, M\, N_1\, N_2 : \tau \rrparenthesis^{o}_{\vec{k}} &\overset{\text{def}}{=} \llparenthesis M \rrparenthesis^{x}_{\vec{k}} \;{}_{x}\odot_{y}^{\llparenthesis \Gamma \rrparenthesis / \vec{k}}\; y \mathbin{\&} \left\{ \mathrm{tt} : \llparenthesis N_1 \rrparenthesis^{o}_{\vec{k}} ; \mathrm{ff} : \llparenthesis N_2 \rrparenthesis^{o}_{\vec{k}} \right\} \\
\llparenthesis \Gamma \vdash \mathsf{iszero} : \mathbb{N} \to \mathbb{B} \rrparenthesis^{o}_{\vec{k}} &\overset{\text{def}}{=} \lambda^{\mathbb{N}}_{x \to o}. \begin{pmatrix} o \oplus \mathrm{tt} & \text{if n = 0} \\ o \oplus \mathrm{ff} & \text{if n > 0} \end{pmatrix}_{n \in \mathbb{N}} \\
\llparenthesis \Gamma \vdash \mathsf{succ} : \mathbb{N} \to \mathbb{N} \rrparenthesis^{o}_{\vec{k}} &\overset{\text{def}}{=} \lambda^{\mathbb{N}}_{x \to o}.\, (o \oplus n = 1)_{n \in \mathbb{N}}
\end{aligned}$$

Macros

Name	Definition	Given
$\lambda^{\sigma}_{x \to o}.\, (P_k)$	$o \oplus \mathsf{lam}.\, o!(r).\, r \mathbin{\&} \{k : r\{x, o\}.\, P_k\} \rhd \Delta, o : \llparenthesis \sigma \to \tau \rrparenthesis$	$\forall k, P_k \rhd \Delta, x : (\llparenthesis \sigma \rrparenthesis / k)^{\perp}, o : \llparenthesis \tau \rrparenthesis$
$\mathsf{unlam}^{v}_{x \to o}(P)$	$\begin{pmatrix} P[o := x] \;{}_{x}\odot_{y} \\ y \mathbin{\&} \left\{ \mathsf{lam} : y^{?} \oplus v.\, [y \leftrightarrow o] \right\} \end{pmatrix} \{o{:=}x, o\} \rhd \Gamma, x : (\llparenthesis \sigma \rrparenthesis / v)^{\perp}, o : \llparenthesis \tau \rrparenthesis$	$P \rhd \Gamma, o : \llparenthesis \sigma \to \tau \rrparenthesis, v \in \mathrm{vt}(\sigma)$
$\mathbf{app}^{\text{cbv}}_{x : \sigma \to o}(P, Q)$	$Q \;{}_{x}\odot^{\Delta}_{y}\; y \mathbin{\&} \left\{ v.\, \mathsf{unlam}^{v}_{y \to o}(P) \right\}_{v \in \mathrm{vt}(\sigma)} \rhd ?\Delta, o : \llparenthesis \tau \rrparenthesis$	$Q \rhd o : \llparenthesis \sigma \to \tau \rrparenthesis, ?\Delta,$ $P \rhd x : \llparenthesis \sigma \rrparenthesis, ?\Delta$

Fig. 10. Translation from call-by-value λ-calculus to π_{LL}

tensor product of the interpretations of the types in Γ: we have $\mathrm{vt}(\otimes\Gamma) = \mathrm{vt}(\Gamma)$. The final translation is then defined as follows:

$$\llparenthesis \Gamma \vdash M : \tau \rrparenthesis^{c,o} := c \mathbin{\&} \left\{ c\{\vec{x}\}.\, \llparenthesis \Gamma \vdash M : \tau \rrparenthesis^{o}_{\vec{k}} \right\}_{\vec{k} \in \mathrm{vt}(\Gamma)} \rhd c : \otimes\Gamma, o : \llparenthesis \tau \rrparenthesis.$$

The translation of terms is presented in Figure 10. Because the types in the context are ensured to implement call-by-value, this translation is more involved than the previous ones, and has to be defined on the typing relation instead of the terms themselves.

3.6.1 Soundness. In the semantics, values can be recognised by the fact that they start by sending an initial move on o.

Lemma 3.12. *If a term $\Gamma \vdash M : \tau$ is a value, then its translation is of the form: $\llparenthesis M \rrparenthesis^{o}_{\vec{k}} \equiv o \oplus \ell.P_{\vec{k}}$ for some $\ell \in vt(\tau)$.*

For example, closed values of type $\mathbb{N}$ are in bijection with natural numbers.

Lemma 3.13. *The effect of substitution of values on the translation is the following:*

$$\llparenthesis \Gamma \vdash M[x := N] \rrparenthesis^{o}_{\vec{k}} = P_{\vec{k}} \;{}_{y}\odot_{x}^{\llparenthesis \Gamma \rrparenthesis / \vec{k}}\; \llparenthesis \Gamma \vdash M \rrparenthesis^{o}_{\vec{k}, x : \ell}.$$

where N is assumed to be a value with $\llparenthesis N \rrparenthesis^{y}_{\vec{k}} = y \oplus \ell.P_{\vec{k}}$.

We then deduce soundness and adequacy (however, in this setting without recursion, adequacy is a much weaker result):

Proposition 3.14 (Soundness and adequacy). *Given two terms $\Gamma \vdash M, N : \tau$, if $M \to_v N$, then $(\!| M |\!)^{c,o} \equiv (\!| N |\!)^{c,o}$. Moreover, if $[\![M]\!] \equiv [\![n]\!]$ for $\vdash M : \mathbb{N}$, then $M \to_v n$.*

4 NONLINEAR AND NON-DETERMINISTIC GAMES

This section extends our framework to handle *nondeterminism.* In § 4.1, we introduce a new model that combines [Castellan et al. 2015] and [Castellan et al. 2018] to have a non-angelic model of nonlinear computations. In § 4.2, we propose our metalanguage for **Str**, denoted as π_{DiLL}, based on Differential Linear Logic (DiLL). In § 4.3, we give the interpretation of a higher-order language with shared-memory concurrency.

4.1 The category Str

Traditional models of game semantics tend to have an *angelic* treatment of nondeterminism: if M is a nondeterministic program, then $[\![M]\!]$ records all the possible maximal executions of M. In particular, it does not record the points where M might get stuck. Such models equalise $\mathrm{tt} + \bot$ and tt (where $+$ represents nondeterministic choice), which shows a loss of information. In models based on event structures, a similar problem occurs: moving to nondeterministic $\sim$-strategies does not give an accurate model of nondeterminism. In [Castellan et al. 2018], this problem was solved by allowing strategies to record internal transitions. Such internal transitions give rise to *internal events* which can be used to record *hidden divergences.*

4.1.1 Essential Strategies. Formally, such strategies are represented as *partial* maps $S \rightharpoonup A$, and events outside the domain are viewed as internal.

Definition 4.1 ([Castellan et al. 2018]). An **essential strategy** on a game A is a partial map $\sigma : S \rightharpoonup A$ such that:

- if $s \rightarrowtriangle_S s'$ and $\sigma\, s, \sigma\, s'$ are incomparable in A, then s is non-positive and s' non-negative;
- if $x \in \mathcal{C}(S)$ and $\sigma\, x$ can be extended by a negative move $a \in A$, then x can be extended by a unique $s \in S$ with $\sigma\, s = a$;
- if $s \sim s'$, then either both $\sigma\, s$ and $\sigma\, s'$ are defined and negative, or both are undefined;
- if $s \in S \setminus \mathrm{dom}(\sigma)$, then there exists s' such that $s \sim s'$.

In the first axiom, by non-positive (resp. non-negative) , we mean that the event is not mapped to a positive event (resp. negative). An event of $s \in S$ is **internal** (or invisible) if $\sigma\, s$ is not defined, and **external** or visible otherwise.

The usual copycat strategy is an essential strategy (which happens to be total). Moreover, essential strategies can be composed in a similar way as strategies.

Theorem 4.2 ([Castellan et al. 2018]). *There is a category of games and essential strategies.*

4.1.2 Essential $\sim$-Strategies. To get essential $\sim$-strategies, we simply merge the definitions as there are very little interactions between the extensions:

Definition 4.3. Given a tcg $\mathcal{A}$, a **uniformity witness** for an essential strategy $\sigma : S \rightharpoonup \mathcal{A}$ is an isomorphism family $\tilde{S}$ such that:

(1) σ extends into a map of event structures with symmetry $(S, \tilde{S}) \rightharpoonup (A, \tilde{A})$;
(2) if $\theta : x \cong y \in \tilde{S}$ and $\sigma\,\theta$ extends to $\varphi : x' \cong y'$ with $x \subseteq^- x'$, then θ extends to θ' such that $\sigma\,\theta' = \varphi$; and
(3) If $\theta : x \cong y \in \tilde{S}$ is the identity on negative elements of x, then θ is the identity on x.

A **$\sim$-essential strategy** on $\mathcal{A}$ is an essential strategy σ on A with a uniformity witness for σ.

As before, uniformity witnesses compose, and most of the structure can be lifted from **DStr**:

Theorem 4.4. *Thin concurrent games and $\sim$-essential strategies form a compact-closed category* **Str**, *which has weak (co)products and an exponential comonad.*

4.1.3 Transitions and weak bisimulation. Event structures come with a natural notion of transition system, given by configurations. This can be lifted to the level of strategies. Given an essential strategy $\sigma : S \rightharpoonup A$, and a configuration $x \in \mathcal{C}(S)$, we build the essential strategy $\sigma/x : S/x \rightharpoonup A/\sigma\, x$ (read σ after x) where S/x consists of the events in S which (1) are not in x, but (2) are compatible in x, and similarly for $A/\sigma\, x$. This construction lifts to $\sim$-essential strategies.

We say that $\sigma : A$ can do a transition to τ with visible actions $y \in \mathcal{C}(A)$, written $\sigma \xrightarrow{y} \tau$, when there exists a configuration $x \in \mathcal{C}(\sigma)$ such that $\sigma/x \approxeq \tau$ and $\sigma\, x \cong y$ in $\tilde{A}$. The particular case where $y = \emptyset$ corresponds to an internal transition and is simply written $\sigma \longrightarrow \tau$. Using this notion of transition, we can define weak bisimulation on strategies:

Definition 4.5. A *weak bisimulation* is an equivalence relation $\mathcal{R}$ between $\sim$-essential strategies on the same tcg such that if $\sigma \mathcal{R} \tau$, for all configuration $y \in \mathcal{C}(A)$, if $\sigma \xrightarrow{y} \sigma'$, then there exists τ' such that $\tau \xrightarrow{y} \tau'$ with $\sigma' \mathcal{R} \tau'$. We write $\approx$ for the largest weak bisimulation.

Note that $\sigma \approxeq \tau$ implies $\sigma \approx \tau$.

4.2 A Metalanguage for Str: π_{DiLL}

We present a calculus for **Str** that extends π_{LL} in § 3.4. For this, we draw inspiration from Differential Linear Logic which extends Linear Logic with nondeterministic primitives. DiLL was introduced to talk about differentiation of programs, but here we use intuitions coming from DiLL in terms of *non-uniform servers*. Earlier, we said that a program of type $!A$ can be seen as a server, able to handle infinitely many requests. However, the only rule in Linear Logic to build such servers is the promotion rule, which can represent only *uniform* servers which handle all requests the same way, without internal state. DiLL allows for the possibility of *stateful* servers, and therefore the possibility of races. This is done by adding two constructs:

- *Codereliction* builds a server which will handle a unique request. If there are several concurrent requests, one of them will be handled and the other ignored (leading to races); and
- *Cocontraction* puts two (possibly partial) servers together. Requests are then routed to one of the two nondeterministically.

There is a problem however with this presentation: if we co-contract two *one-time* servers (obtained by codereliction) and there are two concurrent requests, then there are several possibilities. The best case scenario is that each request goes to a different server, and hence are both handled. The worst case scenario is that they both go to the same server, and then only one of them will be handled. In an angelic model, we only see the best case scenario, but in a non-angelic model we see all cases. To overcome this issue, we add a single construct, *generalised codereliction*, written $a\#(x).\, P$, where x *and* a can be free in P:

$$P ::= \cdots \mid a\#(x).\, P \qquad \mathfrak{P} ::= \cdots \mid a\#(x).\, \mathfrak{P} \qquad \mathfrak{a}^- ::= \cdots \mid a\#(x).\, X.$$

This construct is a mix of codereliction and co-contraction, which sets up a one-time server on a, waits for a request x, and execute P. Unlike the usual codereliction, a is still available in P, so that when P has finished handling the request x, P can listen again on a. A stateful server can be written by $a\#(x).\, P$, and its typing rule is:

$$\text{ND}\ \frac{P \vartriangleright \Delta, a : !A, x : A}{a\#(x).\, P \vartriangleright \Delta, a : !A}$$

This construct can be used to define the usual nondeterministic sum. Given $P, Q \triangleright \Delta$ we define:

$$P + Q \overset{\text{def}}{=} (\nu\, a\, b : !(1 \oplus 1))\big(a\#(x).\, x \,\&\, \{\texttt{inl} : P; \texttt{inr} : Q\}_{i \in I} \parallel (b?[r].\, r \oplus \texttt{inl} \parallel_b b?[r].\, r \oplus \texttt{inr})\big)$$

Interpretation in **Str**. To interpret this extension in **Str**, we use a morphism $c_A : !A \parallel A \nrightarrow !A$. The intuition is that requests on the right side race to get forwarded to A; requests that did not win the race are then forwarded to the left $!A$. Since this strategy has a complex operational behaviour, let us start by an informal description, based on the example where the game has a single positive move, ie. $A = a^+$. The desired strategy c_A for a single move game $A = \mathsf{a}^+$ is (partially) depicted in Figure 11. The strategy starts by waiting for a request on the rightmost $!A$. The different $\mathbf{box}_i^-$ are all racing together to be considered the **first** message acknowledged by c_A. Hence, one neutral event per $\mathbf{box}_i^-$ is triggered when the ith message wins the race. If the ith message has won, then the copy of A on the left is put in contact with the successor of $\mathbf{box}_i^-$, expressed by the links $a^- \rightarrowtriangle \langle a, i\rangle^+$ and $*_i \rightarrowtriangle \langle a, i\rangle^+$. Then, if i has won the race, it means that any other $j \neq i$ lost. In that case, messages on the j component are forwarded to the leftmost $!A$: $\mathbf{box}_j^- \rightarrowtriangle \mathbf{box}_{j,i}^+$ and $*_i \rightarrowtriangle \mathbf{box}_{j,i}^+$. From there, we have a copycat strategy between the ith copy of A on the left and on the right. Formally, the event structure C_A is defined as:

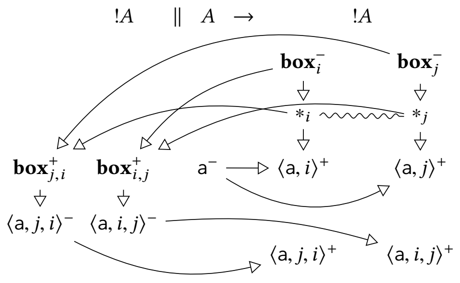


Fig. 11. Strategy c_A for $A = \mathsf{a}^+$

Events are of one of the following form:

- an event $\mathbf{box}_i^-$ for $i \in \mathbb{N}$ mapped to the initial move of the right $!A$;
- an internal event $*_i$, representing the fact that the ith request has won the race
- an event $\mathbf{box}_{i,j}^+$ for $i, j \in \mathbb{N}$, corresponding to the forwarding of $\mathbf{box}_i^-$ when j wins the race;
- an event $\langle e, i\rangle$ for every $e \in \mathbb{C}_A$ and $i \in \mathbb{N}$ representing the forwarding to A when the race is won by i;
- an event $\langle e, i, j\rangle$ for every distinct numbers $i, j \neq \mathbb{N}$ and $e \in \mathbb{C}_A$, for forwarding the ith copy to $!A$ when i loses the race to j.

Causality includes the usual causal order on the copies of $\mathbb{C}_A$, plus the following links:

- $\mathbf{box}_i^- \rightarrowtriangle *_i$ and $*_i \leq \langle e, i\rangle$ for all $e \in \mathbb{C}_A$;
- $\mathbf{box}_i^- \rightarrowtriangle \mathbf{box}_{i,j}^+$ and $*_j \rightarrowtriangle \mathbf{box}_{i,j}^+$ for any $i \neq j$: $\mathbf{box}_i^{+,j}$ is played when Opponent made the i-th request and j won the race; and
- $\mathbf{box}_{i,j}^+ \leq \langle e, i, j\rangle$ for $e \in \mathbb{C}_A$.

Conflict is generated by asking that the $*_i$ are all in mutual conflict.

C_A might not be receptive if A is positive (as in the example), so that the strategy c_A is obtained by precomposing with copycat to ensure receptivity. Using this strategy, we let:

$$[\![a\#(x).\, P \triangleright \Delta, a : !A]\!] = c_A \odot \big([\![P \triangleright \Delta, a : !A, x : A]\!] : [\![\Delta]\!]^\perp \nrightarrow [\![!A]\!] \parallel [\![A]\!]\big)\,.$$

Preorder. Instead of trying to axiomatise $\approxeq$ in this setting, in presence of nondeterminism, it is more fruitful to axiomatise the transition relation $\longrightarrow$. We define the reduction relation $P \longrightarrow Q$ as the smallest preorder closed under the rules in Figure 12. Note that it is not closed under all prefixes. By these rules, we can derive that $P + Q \longrightarrow P$ and $P + Q \longrightarrow Q$.

Lemma 4.6. *For $P, Q \triangleright \Delta$, $P \longrightarrow Q$ implies $[\![P]\!] \longrightarrow [\![Q]\!]$.*

$$\frac{P \longrightarrow Q}{(\nu\, a\, b : T)P \longrightarrow (\nu\, a\, b : T)Q} \qquad \frac{P' \equiv P \longrightarrow Q \equiv Q'}{P' \longrightarrow Q'} \qquad \frac{(\nu\, a\, b_1 : !T)P \longrightarrow (\nu\, a\, b_1 : !T)Q}{(\nu\, a\, b : !T)b?\{b_1, b_2\}.\, P \longrightarrow (\nu\, a\, b : !T)b?\{b_1, b_2\}.\, Q}$$

$$\frac{}{(\nu\, a\, b : !T)(a\#(x).\, P \parallel b?[y].\, Q) \longrightarrow (\nu\, a\, b : !T)(\nu\, x\, y : T)(P \parallel Q)} \qquad \frac{(\nu\, a\, b : T)P \longrightarrow (\nu\, a\, b : T)Q}{(\nu\, a\, b : T)(P \parallel R) \longrightarrow (\nu\, a\, b : T)(Q \parallel R)}$$

Fig. 12. Preorder on π_{DiLL}

$$\frac{}{\langle(\mathsf{skip}; M), \rho\rangle \to \langle M, \rho\rangle} \qquad \frac{M \to_{\mathrm{PCF}} M'}{\langle M, \rho\rangle \to \langle M', \rho\rangle} \qquad \frac{\rho(r) = k}{\langle !r, \rho\rangle \to \langle k, \rho\rangle} \qquad \frac{}{\langle r := k, \rho\rangle \to \langle \mathsf{skip}, \rho[r := k]\rangle}$$

$$\frac{\langle M, \rho\rangle \to \langle M', \rho'\rangle}{\langle E[M], \rho\rangle \to \langle E[M'], \rho'\rangle} \qquad \frac{\langle M, \rho[r := k]\rangle \to \langle M', \rho'\rangle}{\langle \mathsf{new}\, r := k\, \mathsf{in}\, M, \rho\rangle \to \mathsf{new}\, r := \rho'(r)\, \mathsf{in}\, M', \rho'}$$

Fig. 13. Operational Semantics for IPA

4.3 Interpretation of a Shared-Memory Concurrent Higher-Order Language

The Language IPA. Idealised Parallel Algol [Ghica and Murawski 2007], IPA, is an extension of call-by-name PCF with shared memory concurrency. Types of PCF are extended by unit, the type of effectful computation, and loc, the type of *locations*, ie. references to integers. We use $r, s, \ldots$ to range over variables of type loc. The terms are extended as follows:

$$M, N ::= \cdots \mid \mathsf{skip} \mid M; N \mid (M \parallel N) \mid !M \mid M := N \mid \mathsf{new}\, r := k\, \mathsf{in}\, M$$

with the following standard typing rules:

$$\frac{}{\Gamma \vdash \mathsf{skip} : \mathsf{unit}} \qquad \frac{\Gamma \vdash M : \mathsf{unit} \quad \Gamma \vdash N : A}{\Gamma \vdash M; N : A} \qquad \frac{\Gamma \vdash M : \mathsf{unit} \quad \Gamma \vdash N : \mathsf{unit}}{\Gamma \vdash M \parallel N : \mathsf{unit}}$$

$$\frac{\Gamma \vdash M : \mathsf{loc}}{\Gamma \vdash !M : \mathbb{N}} \qquad \frac{\Gamma \vdash M : \mathsf{loc} \quad \Gamma \vdash N : \mathbb{N}}{\Gamma \vdash M := N : \mathsf{unit}} \qquad \frac{\Gamma \vdash M : \mathsf{loc} \quad \Gamma \vdash N : \mathbb{N}}{\Gamma \vdash M := N : \mathsf{unit}} \qquad \frac{\Gamma, r : \mathsf{loc} \vdash M : \mathbb{N}}{\Gamma \vdash \mathsf{new}\, r := k\, \mathsf{in}\, M : \mathbb{N}}$$

In $\mathsf{new}\, r := k\, \mathsf{in}\, M$, k is an integer representing the initial value of r.

Operational Semantics. We recall the semantics of IPA as given in [Ghica and Murawski 2007]. The operational semantics considers **programs**, i.e. open terms of the form $r_1 : \mathsf{loc}, \ldots, r_n : \mathsf{loc} \vdash M : \mathbb{N}$. The operational semantics has states $\langle M, \rho\rangle$ where $\Gamma \vdash M : \mathbb{N}$ is a program and $\mu : \mathrm{dom}(\Gamma) \to \mathbb{N}$ is the memory state. It is given in Figure 13, using the following notion of evaluation contexts:

$$E[] := [] \mid (P \parallel E) \mid (E \parallel P) \mid []; P \mid !E \mid E := n \mid M := E.$$

This operational semantics induces a notion of program equivalence on closed terms: *weak bisimulation.* Two programs $\vdash M, N : \mathbb{N}$ are **weakly bisimilar** (written $M{\approx}N$) when:

- If $M = n$, then $\langle N, \emptyset\rangle \longrightarrow^* n$
- If $\langle M, \emptyset\rangle \longrightarrow \langle M', \emptyset\rangle$, then $\langle N, \emptyset\rangle \longrightarrow^* \langle N', \emptyset\rangle$ and $M{\approx}N$.

Translating IPA. We first extend the translation of types

$$(\!|\, \mathsf{unit} \,|\!) = \mathsf{q}^- \cdot \mathsf{done}^+ \qquad\qquad (\!|\, \mathsf{loc} \,|\!) = \mathsf{q}^- \cdot \&\{\mathsf{rd} : \oplus_{n\in\mathbb{N}}\mathbf{1},\ \mathsf{wr}(i) : \mathsf{done}^+\}$$

The type loc is interpreted as the type of a server waiting for requests that can be either read requests (answered by a number, the current value), or write requests, answered by a message acknowledging the write. The translation of terms is detailed in Figure 14, and follows closely the intuition of the interpretation in [Ghica and Murawski 2007]. $\mathsf{new}\, r := k\, \mathsf{in}\, M$ is interpreted by connecting M with a sequential memory server implemented using the generalised codereliction. The memory server waits for requests and processes them *in sequence.* If two concurrent requests

Translation of Terms

$$(\!|\Gamma \vdash \mathtt{skip} : \mathsf{unit}|\!)^o_i = o \oplus \mathsf{done} \qquad (\!|\Gamma \vdash \mathsf{new}\, r := k \,\mathsf{in}\, M : \mathbb{N}|\!)^o_i \overset{\text{def}}{=} (\!|M|\!)^o_i \;\; {}_r\odot_m \;\; \mathsf{mem}^k_m$$

$$(\!|\Gamma \vdash M \parallel N : \mathsf{unit}|\!)^o_{\mathsf{q}} \overset{\text{def}}{=} (\nu\, x_1\, y_1; x_2\, y_2)\left((\!|M|\!)^{x_1}_{\mathsf{q}} \parallel_\Gamma (\!|N|\!)^{x_2}_{\mathsf{q}} \parallel y_1().y_2().o \oplus \mathsf{done}\right)$$

$$(\!|\Gamma \vdash M; N : \mathsf{unit}|\!)^o_i \overset{\text{def}}{=} (\!|M|\!)^x_{\mathsf{q}} \;\; {}_x\odot^\Gamma_y \;\; y().(\!|N|\!)^o_i \qquad (\!|\Gamma \vdash !M : \mathbb{N}|\!)^o_{\mathsf{q}} \overset{\text{def}}{=} (\!|M|\!)^o_{\mathsf{rd}}$$

$$(\!|\Gamma \vdash M := N : \mathsf{unit}|\!)^o_{\mathsf{q}} \overset{\text{def}}{=} (\!|N|\!)^x_{\mathsf{q}} \;\; {}_x\odot^\Gamma_y \;\; y \,\&\, \left\{n.\, (\!|M|\!)^o_{\mathsf{wr}(n)}\right\}_{n\in\mathbb{N}}$$

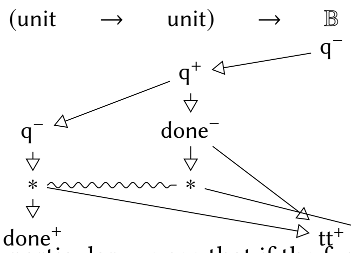
Macros

Name	Definition	Type	Given
mem^k_m	$m\#(x).\, x \,\&\, \left\{\mathsf{wr}(i) : x \oplus \mathsf{done}.\, \mathsf{mem}^i_m \,;\, \mathsf{rd} : x \oplus k.\, \mathsf{mem}^k_m\right\}$	$m : \mathsf{loc}$	$k \in \mathbb{N}$
$o().P$	$o \,\&\, \mathsf{done}.\, P$	$\Delta, o : \mathsf{done}^-$	$P \triangleright \Delta$

Fig. 14. Interpretation of IPA into π_{DiLL}

are made, then a sequentialisation is picked nondeterministically. (This behaviour is due to the implementation of $a\#(x)\,P$.)

A first example (without state) of the translation is depicted in Figure 1, where we write $f[a].\,P$ for $f \oplus \mathsf{q}.\, f\{a, f\}.\, P$ for readability. Another more interesting example is the sequential program $M = \lambda f.\,\mathsf{new}\; r := 0 \;\mathsf{in}\; f(r := 1); !r = 1$ whose interpretation is depicted on the right. It tests whether its argument is a strict function (ie. evaluates its argument before returning). Its interpretation shows what can happen when M is evaluated in a concurrent context. In particular, we see that if the function f provided by the context returns and evaluates its argument there is a race. This is not possible in IPA, but could be possible in extensions of IPA including control operators.



Our model is adequate with respect to weak bisimulation:

Theorem 4.7 (Adequacy). *For any closed program $\vdash M : \mathbb{N}$, we have*

$$M \approx N \qquad \textit{if and only if} \qquad [\![M]\!] \approx [\![N]\!]$$

5 IMPLEMENTATION

To illustrate the model and the metalanguage, we have built a simple prototype that can be used to compute the interpretation of a term of IPA or of π_{DiLL}. The prototype is available at:

http://programminggamesemantics.github.io/index.html

The prototype allows entering *finitary* IPA terms, that is, terms without fixpoints or natural numbers, and references store booleans. Such terms can then be converted to π_{DiLL} and finally the event structure is displayed. Or a term of π_{DiLL} can directly be entered and its interpretation is displayed. The implementation of the model relies on the compilation of the metalanguage to a sequential monadic language with a monad expressive enough to support the message-passing primitives of π_{DiLL}. The code is then run and every branch of the process is explored.

6 RELATED WORK

Metalanguage and Process Representation for Strategies and Games. Hyland and Ong [1995] first studied a relationship between game semantics and the π-calculus, where π-calculus processes are used to denote plays of innocent strategies (for PCF). This idea led to recast the traditional encoding of the call-by-value λ-calculus into the π-calculus [Milner 1992] into a game semantics model for call-by-value PCF [Honda and Yoshida 1999]; and to translate conditions on strategies (innocence, well-bracketing) into typing disciplines for the π-calculus [Berger et al. 2001; Yoshida et al. 2004]. In the sequential setting, the work by Longley [2009] proposed a programming language to describe sequential innocent strategies as a whole. Later, Goyet [2013] proposed an abstract calculus for sequential strategies, close to the πI-calculus. An abstract metalanguage based on the profunctors view of strategies has been proposed by Castellan et al. [2014] but its expressiveness is unclear; and no syntax is provided for the affine maps on which its primitives rely.

Dimovski and Lazic [2004]; Ghica and Murawski [2006] represent strategies as CSP terms for use with model checkers. However, their calculi are less canonical than our metalanguages which are based on Linear Logic, and are not connected to the model by an equational theory. In a similar vein, Disney and Flanagan [2015] argue for a reading of strategies in terms of processes in the sequential setting, for type soundness.

Game Semantics for Concurrency and Nondeterminism. The first concurrent game semantics model, based on traces, is due to Laird [2001] for a message-passing language, extended to IPA later by Ghica and Murawski [2007], model which is fully abstract for may-equivalence. The first causal model of a concurrent nonlinear language is due to [Sakayori and Tsukada 2017], based on sets of partial orders. This model, yet causal, is still angelic. The first causal model of (linear) IPA is due to [Castellan and Clairambault 2016], also angelic. Harmer and McCusker [1999] provide the first non-angelic model of game semantics in the sequential setting based on *stopping traces*. This approach is tailored to must-equivalence. Our approach using essential events gives the first accurate (adequate for weak bisimulation) model of IPA, capturing faithfully the nondeterministic branching behaviours of shared-memory concurrent programs. Another line of work uses presheaves to represent nondeterminism faithfully [Eberhart et al. 2015; Tsukada and Ong 2015], but do not give a causal interpretation of the languages studied.

In another line of the work, Sakayori and Tsukada [2019] give a sound and complete axiomatisation of the equational theory induced on the asynchronous π-calculus by the game semantics model of [Laird 2001], only fully abstract for may-equivalence. Note that obtaining a model adequate for bisimulation is usually harder than may-equivalence in nondeterministic languages.

Recently, Melliès [2019] gave a games semantics model for DiLL based on templates games. It differs from ours in two ways: (1) his model is synchronous, while our model is asynchronous (due to courtesy); and (2) it ignores deadlocks, preventing from modelling IPA adequately.

Extensions of the Linear-Logic and Session Types Correspondence. In this paper, we use a variation of Differential Linear Logic (DiLL) to extend our deterministic framework to nondeterminism. Beffara [2006] presents a model of Linear Logic in terms of processes of the πI-calculus, used to interpret concurrent extensions of the λ-calculus. A different approach exists which uses Linear Logic-based session π-calculi to investigate expressiveness of network topologies [Toninho and Yoshida 2018a] and encodability via various forms of the λ-calculi (see [Toninho and Yoshida 2018b, § 5]). These papers have proven that the expressiveness of the original Linear Logic-based calculi is limited to functional and strong normalising behaviours. To overcome these limitations, several extensions have been proposed, eg. the lock primitives for nondeterminism [Balzer and Pfenning 2017]; dynamic monitoring [Gommerstadt et al. 2018; Jia et al. 2016]; exceptional handling [Caires and

Pérez 2017]; multiparty interactions [Carbone et al. 2016, 2015]; and hyperenvironments to capture non-blocking I/O [Kokke et al. 2019]. Our approach differs from them, aiming to describe game semantics translations more accurately based on the metalanguages arisen from event structures, in particular using the memory cell, grounded on the rigorous DiLL logic.

7 CONCLUSION

We presented a syntactic understanding of game semantics interpretations of higher-order languages based on message-passing concurrency. This understanding allows for easy and flexible interpretations of higher-order concurrent programs, as well as equational reasoning on them. We believe it opens a large area of future research into using game semantics to precisely capture the semantics of sophisticated programming languages via their translations into the π-calculi. Our metalanguage allows the specification of complicated causal models by simple syntactic translations.

As future work, we plan to explore more advanced source programming languages, in particular relaxed shared memory, by combining our approach with the techniques presented in [Castellan 2016]. In a more theoretical direction, we will extend the metalanguage to support definability results, and the equational theory to support completeness results; on the categorical side, we will try to characterise **Str** as an initial category. We would also like to explore the operational aspects of the metalanguage: following Ghica and Tzevelekos [2012], we would like to describe operationally the construction of the event structure of a process. Another future work is to represent the event structures generated by processes in a finite way, using for instance Petri Nets generated by a geometry of interaction [Lago et al. 2017].

REFERENCES

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

Stephanie Balzer and Frank Pfenning. 2017. Manifest sharing with session types. *PACMPL* 1, ICFP (2017), 37:1–37:29. https://doi.org/10.1145/3110281

Emmanuel Beffara. 2006. A Concurrent Model for Linear Logic. *Electr. Notes Theor. Comput. Sci.* 155 (2006), 147–168. https://doi.org/10.1016/j.entcs.2005.11.055

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

Luís Caires and Jorge A. Pérez. 2017. Linearity, Control Effects, and Behavioral Types. In *ESOP*. 229–259. https://doi.org/10.1007/978-3-662-54434-1_9

Luís Caires and Frank Pfenning. 2010. Session Types as Intuitionistic Linear Propositions. In *CONCUR 2010 - Concurrency Theory, 21th International Conference, CONCUR 2010, Paris, France, August 31-September 3, 2010. Proceedings (Lecture Notes in Computer Science)*, Paul Gastin and François Laroussinie (Eds.), Vol. 6269. Springer, 222–236. https://doi.org/10.1007/978-3-642-15375-4_16

Marco Carbone, Sam Lindley, Fabrizio Montesi, Carsten Schuermann, and Philip Wadler. 2016. Coherence Generalises Duality: a logical explanation of multiparty session types. In *CONCUR'16 (Leibniz International Proceedings in Informatics (LIPIcs))*, Vol. 59. Schloss Dagstuhl, 33:1–33:15.

Marco Carbone, Fabrizio Montesi, Carsten Schormann, and Nobuko Yoshida. 2015. Multiparty Session Types as Coherence Proofs. In *CONCUR 2015 (LIPIcs)*, Vol. 42. Schloss Dagstuhl, 412–426.

Simon Castellan. 2016. Weak memory models using event structures. In *Vingt-septième Journées Francophones des Langages Applicatifs (JFLA 2016).*

Simon Castellan and Pierre Clairambault. 2016. Causality vs interleaving in game semantics. In *CONCUR 2016 - Concurrency Theory.*

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

Simon Castellan, Pierre Clairambault, and Glynn Winskel. 2019. Thin Games with Symmetry and Concurrent Hyland-Ong Games. *Logical Methods in Computer Science* 15, 1 (2019). https://lmcs.episciences.org/5248

Simon Castellan, Jonathan Hayman, Marc Lasson, and Glynn Winskel. 2014. Strategies as Concurrent Processes. *Electr. Notes Theor. Comput. Sci. (Special issue for MFPS XXX)* 308 (2014), 87–107. https://doi.org/10.1016/j.entcs.2014.10.006

Simon Castellan and Nobuko Yoshida. 2019. Two Sides of the Same Coin: Session Types and Game Semantics. In *Proceedings of the 46th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2019, Lisbon, Portugal.*

Aleksandar Dimovski and Ranko Lazic. 2004. CSP Representation of Game Semantics for Second-Order Idealized Algol. In *Formal Methods and Software Engineering, 6th International Conference on Formal Engineering Methods, ICFEM 2004, Seattle, WA, USA, November 8-12, 2004, Proceedings (Lecture Notes in Computer Science)*, Jim Davies, Wolfram Schulte, and Michael Barnett (Eds.), Vol. 3308. Springer, 146–161. https://doi.org/10.1007/978-3-540-30482-1_18

Tim Disney and Cormac Flanagan. 2015. Game Semantics for Type Soundness. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015.* 104–114. https://doi.org/10.1109/LICS.2015.20

Clovis Eberhart, Tom Hirschowitz, and Thomas Seiller. 2015. An Intensionally Fully-abstract Sheaf Model for pi. In *6th Conference on Algebra and Coalgebra in Computer Science, CALCO 2015, June 24-26, 2015, Nijmegen, The Netherlands (LIPIcs)*, Lawrence S. Moss and Pawel Sobocinski (Eds.), Vol. 35. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 86–100. https://doi.org/10.4230/LIPIcs.CALCO.2015.86

Thomas Ehrhard. 2018. An introduction to differential linear logic: proof-nets, models and antiderivatives. *Mathematical Structures in Computer Science* 28, 7 (2018), 995–1060. https://doi.org/10.1017/S0960129516000372

Dan R. Ghica and Andrzej S. Murawski. 2006. Compositional Model Extraction for Higher-Order Concurrent Programs. In *Tools and Algorithms for the Construction and Analysis of Systems, 12th International Conference, TACAS 2006 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2006, Vienna, Austria, March 25 - April 2, 2006, Proceedings (Lecture Notes in Computer Science)*, Holger Hermanns and Jens Palsberg (Eds.), Vol. 3920. Springer, 303–317. https://doi.org/10.1007/11691372_20

Dan R. Ghica and Andrzej S. Murawski. 2007. Angelic Semantics of Fine-Grained Concurrency.

Dan R. Ghica and Nikos Tzevelekos. 2012. A System-Level Game Semantics. *Electr. Notes Theor. Comput. Sci.* 286 (2012), 191–211. https://doi.org/10.1016/j.entcs.2012.08.013

J.Y. Girard. 1987. Linear logic. *Theoretical computer science* 50, 1 (1987), 1–101.

Hannah Gommerstadt, Limin Jia, and Frank Pfenning. 2018. Session-Typed Concurrent Contracts. In *ESOP*. 771–798. https://doi.org/10.1007/978-3-319-89884-1_27

Alexis Goyet. 2013. The Lambda Lambda-Bar calculus: a dual calculus for unconstrained strategies. In *The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '13, Rome, Italy - January 23 - 25, 2013*, Roberto Giacobazzi and Radhia Cousot (Eds.). ACM, 155–166. https://doi.org/10.1145/2429069.2429089

Ronghui Gu, Zhong Shao, Jieung Kim, Xiongnan (Newman) Wu, Jérémie Koenig, Vilhelm Sjöberg, Hao Chen, David Costanzo, and Tahina Ramananandro. 2018. Certified Concurrent Abstraction Layers. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2018).* ACM, New York, NY, USA, 646–661. https://doi.org/10.1145/3192366.3192381

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.* IEEE Computer Society, 422–430. https://doi.org/10.1109/LICS.1999.782637

Kohei Honda, Vasco Thudichum Vasconcelos, and Makoto Kubo. 1998. Language Primitives and Type Discipline for Structured Communication-Based Programming. In *Programming Languages and Systems - ESOP'98, 7th European Symposium on Programming, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS'98, Lisbon, Portugal, March 28 - April 4, 1998, Proceedings (Lecture Notes in Computer Science)*, Chris Hankin (Ed.), Vol. 1381. Springer, 122–138. https://doi.org/10.1007/BFb0053567

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

J. M. E. Hyland and C.-H. Luke Ong. 1995. Pi-Calculus, Dialogue Games and PCF. In *Proceedings of the seventh international conference on Functional programming languages and computer architecture, FPCA 1995, La Jolla, California, USA, June 25-28, 1995.* 96–107. https://doi.org/10.1145/224164.224189

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

Limin Jia, Hannah Gommerstadt, and Frank Pfenning. 2016. Monitors and Blame Assignment for Higher-order Session Types. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '16).* ACM, New York, NY, USA, 582–594. https://doi.org/10.1145/2837614.2837662

Wen Kokke, Fabrizio Montesi, and Marco Peressotti. 2019. Better Late Than Never: A Fully-abstract Semantics for Classical Processes. *Proc. ACM Program. Lang.* 3, POPL, Article 24 (Jan. 2019), 29 pages. https://doi.org/10.1145/3290337

Ugo Dal Lago, Ryo Tanaka, and Akira Yoshimizu. 2017. The Geometry of Concurrent Interaction: Handling Multiple Ports by Way of Multiple Tokens (Long Version). *CoRR* abs/1704.04620 (2017). arXiv:1704.04620 http://arxiv.org/abs/1704.04620

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

John Longley. 2009. Some Programming Languages Suggested by Game Models (Extended Abstract). *Electr. Notes Theor. Comput. Sci.* 249 (2009), 117–134. https://doi.org/10.1016/j.entcs.2009.07.087

P.A. Melliès. 2009. Categorical semantics of linear logic. In *Interactive models of computation and program behaviour.* Société mathématique de France.

Paul-André Melliès. 2019. Template games and differential linear logic. Accepted at LICS'19.

Paul-André Melliès. 2003. Asynchronous games 1: A group-theoretic formulation of uniformity. *Manuscript, Available online* (2003).

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

Robin Milner, Joachim Parrow, and David Walker. 1992. A Calculus of Mobile Processes, I. *Inf. Comput.* 100, 1 (1992), 1–40. https://doi.org/10.1016/0890-5401(92)90008-4

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 π-Calculus. In *Proceedings of the 20th International Conference on Foundations of Software Science and Computation Structures - Volume 10203.* Springer-Verlag New York, Inc., New York, NY, USA, 389–406. https://doi.org/10.1007/978-3-662-54458-7_23

Ken Sakayori and Takeshi Tsukada. 2019. A Categorical Model of an i/o-typed pi-calculus. In *Programming Languages and Systems - 28th European Symposium on Programming, ESOP 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings (Lecture Notes in Computer Science)*, Luís Caires (Ed.), Vol. 11423. Springer, 640–667. https://doi.org/10.1007/978-3-030-17184-1_23

Davide Sangiorgi. 1996. pi-Calculus, Internal Mobility, and Agent-Passing Calculi. *Theor. Comput. Sci.* 167, 1&2 (1996), 235–274. https://doi.org/10.1016/0304-3975(96)00075-8

Robert Seely. 1989. Linear logic, $\star$-autonomous categories and cofree coalgebras. *Contemporary mathematics* 92 (1989).

Gordon Stewart, Lennart Beringer, Santiago Cuellar, and Andrew W. Appel. 2015. Compositional CompCert. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*, Sriram K. Rajamani and David Walker (Eds.). ACM, 275–287. https://doi.org/10.1145/2676726.2676985

Bernardo Toninho and Nobuko Yoshida. 2018a. Interconnectability of Session-Based Logical Processes. *ACM Trans. Program. Lang. Syst.* 40, 4, Article 17 (Dec. 2018), 42 pages. https://doi.org/10.1145/3242173

Bernardo Toninho and Nobuko Yoshida. 2018b. On Polymorphic Sessions And Functions: A Tale of Two (Fully Abstract) Encodings. In *27th European Symposium on Programming (LNCS)*, Vol. 10801. Springer, 827–855.

Takeshi Tsukada and C.-H. Luke Ong. 2015. 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.* IEEE Computer Society, 220–231. https://doi.org/10.1109/LICS.2015.30

Philip Wadler. 2014. Propositions as sessions. *J. Funct. Program.* 24, 2-3 (2014), 384–418. https://doi.org/10.1017/S095679681400001X

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

Glynn Winskel. 2007. Event structures with symmetry. *Electronic Notes in Theoretical Computer Science* 172 (2007), 611–652.

Nobuko Yoshida, Martin Berger, and Kohei Honda. 2004. Strong Normalisation in the π-Calculus. *Information and Computation* 191(2004) (2004), 145–202.

A PROOFS OF § 2

In this section, we provide more details proofs of the results presented in § 2.

A.1 Categorical Structure

A.1.1 Lifting of Maps to Strategies. First, let us show some properties of the lifting of maps defined in § 2.3.1.

Given a strategy $\sigma : S \to A^\perp \parallel B$ and a map of games $f : B \to C$, we form the strategy $\sigma_f : S \downarrow V \to A^\perp \parallel \mathrm{dom}(f) \xrightarrow{A^\perp \parallel f} A^\perp \parallel C$ where V contains the set of $s \in S$ such that $\sigma s \in A \parallel \mathrm{dom}(f)$.

Lemma A.1. *σ_f is a strategy $A \mathrel{+\!\!\!\to} C$.*

Proof. Courtesy and determinism follow from f preserving the causal order and receptivity from f being bijective on its domain. □

It is easy to see that $\mathsf{lift}(f)$ is equal to $(\text{æ}_A)_f$. We have the following result:

Lemma A.2. *[Castellan et al. 2017] Given $\sigma : A \relbar\joinrel\mapsto B$ and $f : B \to C$ a map of games, we have:*

$$\sigma_f \cong \mathsf{lift}(f) \odot \sigma.$$

This result implise the functoriality of $\mathsf{lift}(-)$: $\mathsf{lift}(g \circ f) = \mathsf{lift}(g) \odot \mathsf{lift}(f)$.

A.1.2 Weak products. We now show that **LDStr** has weak products.

Lemma 2.7. *For every strategies $\sigma : A \relbar\joinrel\mapsto B$ and $\tau : A \relbar\joinrel\mapsto C$ there exists a strategy $\langle \sigma, \tau \rangle : A \relbar\joinrel\mapsto B \,\&\, C$ such that $\pi_1 \odot \langle \sigma, \tau \rangle \cong \sigma$ and $\pi_2 \odot \langle \sigma, \tau \rangle \cong \tau$. By duality, we obtain weak coproducts $A \oplus B = (A^\perp \,\&\, B^\perp)^\perp$.*

Proof. Consider $\sigma : S \to A^\perp \parallel B$ and $\tau : T \to A^\perp \parallel C$. By prefixing, we obtain maps of event structures (but not strategies):

$$\mathsf{L}^- \cdot \sigma : \mathsf{R}^- \cdot \sigma \to A^\perp \parallel B \,\&\, C \qquad \mathsf{R}^- \cdot \tau : \mathsf{L}^- \cdot \tau \to A^\perp \parallel B \,\&\, C$$

Then, it is easy to see that their sum forms a map of event structure $\sigma \sqcup \tau : (\mathsf{L}^- \cdot \sigma) + (\mathsf{R}^- \cdot \sigma) \to A^\perp \parallel (B \,\&\, C)$.

An easy verification shows that $\pi_1 \circ (\sigma \sqcup \tau) = \sigma$ and similarly for π_2. To get $\langle \sigma, \tau \rangle$, we precompose with copycat to get a strategy, and this operation preserves the equations with the projections. □

A.2 Reasoning on the equational theory

In this section, we provide proofs of properties on the equational theory.

Lemma 2.10 (Soundness). *If $P \equiv Q$, then $[\![P]\!] \cong [\![Q]\!]$.*

Proof. Rules in the compact-close fragment of the theory follow from the compact-closed structure of **LDStr**, in particular bifunctoriality of $\parallel$.

Rules for permutations follow from bifunctoriality of $\parallel$ and the fact that nested pairing commute (by receptivity).

In communication, the rules follow from the weak coproduct structure. In cut elimination, from an easy investigation on the composition. □

A.3 Proof of Correctness of the Encoding

We now show our encoding of the affine λ-calculus captures $\beta\eta$-equivalence.

A.3.1 Proof of Soundness. First, let us notice that to showing an equality $(\!| M |\!)^o \equiv (\!| N |\!)^o$ is equivalent to showing the equalities $(\!| M |\!)^o_i \equiv (\!| N |\!)^o_i$ for all $i \in \mathrm{ct}(\tau)$ where τ is the return type of M and N.

Lemma A.3. *Given $\Gamma, x : \tau_1 \vdash M : \tau_2$ and $\Delta \vdash N : \tau_2$, and $i \in ct(\tau_2)$, we have:*

$$(\!| M[x := N] |\!)^o_i \equiv (\!| M |\!)^o_i \;{}_x{\odot}_y\; (\!| N |\!)^y.$$

Proof. By induction. Let us detail a few cases:

- If $M = x$:

$$\begin{aligned} (\!| x |\!)^o_i \;{}_x{\odot}_y\; (\!| N |\!)^y &\equiv (x \oplus i.\, [x {\leftrightarrow} o]) \;{}_x{\odot}_y\; y \,\&\, \left\{ (\!| N |\!)^y_j \right\}_{j \in \mathrm{ct}(\tau_2)} \\ &\equiv [x {\leftrightarrow} o] \;{}_x{\odot}_y\; (\!| N |\!)^y_i \\ &\equiv (\!| N |\!)^o_i \end{aligned}$$

- If $M = w \neq x$:

$$\begin{aligned}(\!| w |\!)^o_i \;{}_x\!\odot_y\; (\!| N |\!)^y &\equiv (w \oplus i.\, [w{\leftrightarrow}o]) \;{}_x\!\odot_y\; (\!| N |\!)^y\\ &\equiv w \oplus i.\, ([w{\leftrightarrow}o] \parallel (\nu\, x\, y)\, ((\!| N |\!)^y))\\ &\equiv (\!| y |\!)^o\end{aligned}$$

  The second equality is due to the fact that x does not occur in the left-hand side of $\odot$ and the right-handside of $\odot$ starts with an input of y and the last equality due to the rule for deadlock ($(\!| N |\!)^y$ starts by an input on y).
- If $M = \lambda w.\, M'$:

$$\begin{aligned}(o\{w, o\}.\, (\!| M' |\!)^o_i) \;{}_x\!\odot_y\; (\!| N |\!)^y &\equiv o\{w, o\}.\, \left((\!| M' |\!)^o_i \;{}_x\!\odot_y\; (\!| N |\!)^y\right)\\ &\equiv (\!| \lambda w.\, M'[x := N] |\!)^o_i\end{aligned}$$

□

Lemma A.4. *Consider $\Gamma \vdash M =_{\beta\eta} N : \tau$. Then $(\!| t |\!)^a \equiv (\!| u |\!)^a$.*

Proof. By induction on the definiton of $=_{\beta\eta}$. We detail some interesting case:

- The case $(\lambda x.\, M)\, N =_{\beta\eta} M[x := N]$. Using Lemma A.3, we have for all $i \in \mathrm{ct}(\tau)$:

$$\begin{aligned}(\!| (\lambda x.\, M)\, N |\!)^o_i &\equiv ((o\{x, o\}.\, (\!| M |\!)^o_i)\{o{:=}x, o\}) \;{}_x\!\odot_y\; (\!| N |\!)^y\\ &\equiv (\!| M |\!)^o_i \;{}_x\!\odot_y\; (\!| N |\!)^y \equiv (\!| M[x := N] |\!)^o_i\end{aligned}$$

- The case $\lambda x.\, M\, x =_{\beta\eta} t$ when M has type $A \multimap B$.

$$\begin{aligned}(\!| \lambda x.\, M\, x |\!)^o_i &\equiv o\{x, o\}.\, (((\!| M |\!)^o_i)\{o{:=}u, o\} \;{}_u\!\odot_v\; (\!| x |\!)^v)\\ &\equiv o\{x, o\}.\, (((\!| M |\!)^o_i)\{o{:=}u, o\} \;{}_u\!\odot_v\; (\!| x |\!)^v)\end{aligned}$$

  Now, since naming actions float freely, we can assume that $(\!| M |\!)^o_i \equiv o\{u, o\}.\, P$, and we get:

$$\begin{aligned}(\!| \lambda x.\, M\, x |\!)^o_i &\equiv o\{x, o\}.\, (P \;{}_u\!\odot_v\; [x{\leftrightarrow}v])\\ &\equiv (\!| \lambda x.\, M\, x |\!)^o_i \equiv o\{x, o\}.\, P[u := x] \equiv (\!| M |\!)^o_i\end{aligned}$$

□

A.3.2 Proof of Completeness. We shall now prove completeness. First, let us define $\beta\eta$-long normal forms. We introduce two judgements $\Gamma \vdash_{nf} t : A$ (t is a normal form of type A) and $\Gamma \vdash_{ne} t : A$ (t is a neutral term of type A). The rules are as follows:

$$\frac{\Gamma, x : A \vdash_{nf} t : B}{\Gamma \vdash_{nf} \lambda x.\, t : A \multimap B} \qquad \frac{}{\Gamma \vdash_{nf} \mathsf{tt}, \mathsf{ff} : \mathbb{B}} \qquad \frac{\Gamma \vdash_{ne} n : \mathbb{B} \qquad \Gamma \vdash_{nf} t, u : \mathbb{B}}{\Gamma \vdash_{nf} \mathsf{if}\, n\, t\, u : \mathbb{B}} \qquad \frac{}{\Gamma, x : A \vdash_{ne} x : A}$$

$$\frac{\Gamma \vdash_{ne} n : A \multimap B \qquad \Gamma \vdash_{nf} t : A}{\Gamma \vdash_{ne} n\, t : B}$$

Lemma A.5. *Let $\Gamma \vdash t, u : A$ be two normal forms. Then $t =_{\beta\eta} u$ iff t and u are α-equivalent.*

Proof. Immediate since normal forms cannot be β-reduced or η-expanded. □

Lemma A.6. *For every term $\Gamma \vdash t : A$, there exists a normal form $\Gamma \vdash nf(t) : A$ such that $t =_{\beta\eta} nf(t)$.*

Proof. By β-normalisation and η-expansion. □

Lemma A.7. *Consider two normal forms $\Gamma \vdash_{nf} M, N : A$. Then $[\![M]\!] \cong [\![N]\!]$ implies that M and N are α-equivalent.*

Proof. Standard result on game semantics, done by induction on M and look at the first positive move of $[\![M]\!] \cong [\![N]\!]$. □

Theorem 2.11 (Soundness and Completeness). *We have $M =_{\beta\eta} N$ if and only if $(\!| M |\!)^a \equiv (\!| N |\!)^a$ if and only if $[\![M]\!] \cong [\![N]\!]$.*

Proof. We have already proved the direct implication. The converse implication is a consequence of Lemmata A.7 and A.6. □

B PROOFS OF § 3

B.1 Proofs of § 3.3

Lemma 3.6. *The functor $\uparrow^-(-)$ is the right adjoint to the inclusion $\mathbf{DStr}^- \to \mathbf{DStr}$. In particular, $\uparrow^-$ transports comonads on $\mathbf{DStr}^-$ to comonads on $\mathbf{DStr}$.*

Proof. The adjunction is fairly simple to describe:

- From $\sigma : \uparrow^+ A \twoheadrightarrow B$ in $\mathbf{DStr}^+$, by removing the initial negative event on $\uparrow^+ A$, we get a strategy $A \twoheadrightarrow B$ in $\mathbf{DStr}$ (not necessarily negative).
- From $\sigma : A \twoheadrightarrow B$ in $\mathbf{DStr}$, we can build a strategy $\mathbf{box}^- \cdot \sigma : \uparrow^+ A \twoheadrightarrow B$ which is well-defined because B is positive.

It is clear that these operations are inverse of each other; naturality is a simple calculation. □

B.2 Proofs of correctness of the call-by-name translation

B.2.1 Definition of $\to_{\mathsf{PCF}}$. First, we define exhaustively $\to_{\mathsf{PCF}}$:

$$\frac{}{(\lambda x.\, M)\, N \to_{\mathsf{PCF}} M[x := N]} \qquad \frac{}{\mathsf{if}\ \mathsf{tt}\ M\ N \to_{\mathsf{PCF}} M} \qquad \frac{}{\mathsf{if}\ \mathsf{ff}\ M\ N \to_{\mathsf{PCF}} N} \qquad \frac{}{\mathtt{iszero}(0) \to_{\mathsf{PCF}} \mathsf{tt}}$$

$$\frac{n > 0}{\mathtt{iszero}(n) \to_{\mathsf{PCF}} \mathsf{ff}} \qquad \frac{}{\mathtt{succ}(n) \to_{\mathsf{PCF}} n + 1} \qquad \frac{}{Y\, M \to_{\mathsf{PCF}} M\, (Y\, M)} \qquad \frac{M \to_{\mathsf{PCF}} M'}{E[M] \to_{\mathsf{PCF}} E[M']}$$

where $E[]$ is an evaluation context, defined by the following grammar:

$$E[] ::= [] \mid E\, N \mid \lambda x.E \mid \mathtt{iszero}(E) \mid \mathtt{succ}(E) \mid \mathsf{if}\ E\ M\ N.$$

First, it is straightforward to extend the substitution lemma in this setting:

Lemma B.1 (Substitution lemma). *Let $\Gamma, x : \sigma \vdash M : \tau$ and $\Gamma \vdash N : \sigma$. We have*

$$(\!| M[x := N] |\!)^o_i \equiv (\!| M |\!)^o_i \ {}_x\odot_y^{(\!|\Gamma|\!)}\ y!(r).\, (\!| N |\!)^r.$$

Proof. By induction on M. □

Lemma B.2 (Soundness). *For $\Gamma \vdash M, N : \tau$, if $M \to_{PCF} N$, then $(\!| M |\!)^o \equiv (\!| N |\!)^o$.*

Proof. The proof is essentially the same as for the affine case, by simple application of the equational rules.

Let us detail a few rules. First, it is enough to show that $(\!| M |\!)^o_i \equiv (\!| N |\!)^o_i$ for all $i \in \mathrm{ct}(\tau)$.

- β-reduction:

$$\begin{aligned}(\!| (\lambda x.\, M)\, N |\!)^o_i &\equiv \big(o\{x, o\}.\, (\!| M |\!)^o_i\big)\{o{:=}x, o\}\ {}_x\odot_y^{(\!|\Gamma|\!)}\ y!(r).\, (\!| N |\!)^r \\ &\equiv (\!| M[x := N] |\!)^o_i\end{aligned}$$

  by direct application of Lemma B.1.

- if:

$$\begin{aligned}(\!|\, \mathsf{if}\ \mathsf{tt}\, M\, N \,|\!)^o_i &\equiv u \oplus \mathsf{tt}\ \ {}_u\odot_v^{(\!|\Gamma|\!)}\ \ v \,\&\, \{\mathsf{tt} : (\!|\, M \,|\!)^o_i; \dots\} \\ &\equiv (\!|\, M \,|\!)^o_i\end{aligned}$$

- succ:

$$(\!|\, \mathsf{succ}(n) \,|\!)^o_{\mathsf{q}} \equiv x \oplus n\ \ {}_x\odot_y\ \ y \,\&\, \{o \oplus n+1\} \equiv o \oplus n+1$$

□

B.2.2 Finitary PCF and adequacy. Let us define finitary PCF here as the subset of PCF without fixpoint, with an added term $\bot : A$ denoting formal divergence. Closed terms of finitary PCF of type $\mathbb{N}$ converge either towards a natural number or towards $\bot$ in a finite number of step. Given a term of PCF M, we write its n-th approximation M_n defined as M where Y is substituted by $\lambda f.\, f^n \bot$ where f^n is the n-fold iteration of f.

Lemma B.3. *$(\!|\, M \,|\!)^o$ is the limit of the increasing chain $(\!|\, M_n \,|\!)^o$.*

Proof. Straightforward since Y is the limit of the $\lambda f.\, f^n \bot$. □

Lemma B.4. *For $\vdash M : \mathbb{N}$, if $[\![M]\!] = [\![n]\!]$, then $[\![M_k]\!] = [\![n]\!]$ for some $k \in \mathbb{N}$.*

Proof. Trivial, because we know that a positive event labelled n belongs to $[\![M]\!]$, and since $[\![M]\!]$ is the limit of the $[\![M_n]\!]$ by the previous lemma, and continuity of $[\![\cdot]\!]$, the result follows. □

Lemma B.5. *If M diverges for $\rightarrow_{PCF}$, then $(\!|\, M \,|\!)^o \equiv \mathbf{0}$.*

Proof. Since M diverge, we have $M_n \rightarrow^*_{\mathsf{PCF}} \bot$ for all $n \in \mathbb{N}$. Since $(\!|\, M_n \,|\!)^o \equiv \mathbf{0}$, we can conclude. □

Proposition 3.11 (Soundness and adequacy). *Given two terms $\Gamma \vdash M, N : \tau$, if $M \rightarrow_{PCF} N$ then $(\!|\, M \,|\!)^o \equiv (\!|\, N \,|\!)^o$. Moreover if $(\!|\, M \,|\!)^o \equiv (\!|\, n \,|\!)^o$ for $\vdash M : \mathbb{N}$, then $M \rightarrow_{PCF} n$.*

Proof. We already have proved soundness. For adequacy, it is a consequence of the previous lemma: M cannot diverge, so it must converge to a natural number, and clearly the model is injective on natural numbers: $(\!|\, n \,|\!) = (\!|\, m \,|\!)$ implies that $n = m$. □

B.3 Proof of correctness of the call-by-value translation

First, let us define the relation $\rightarrow_v$ as the subset of $\rightarrow_{\mathsf{PCF}}$ which contains only the instances of the β-reduction $(\lambda x.\, M) \rightarrow M[x := N]$ when N is a value, ie. a λ-abstraction, a natural number or a boolean.

Lemma 3.12. *If a term $\Gamma \vdash M : \tau$ is a value, then its translation is of the form: $(\!|\, M \,|\!)^o_{\vec{k}} \equiv o \oplus \ell.P_{\vec{k}}$ for some $\ell \in vt(\tau)$.*

Proof. Easy case distinction on N. □

Lemma 3.13. *The effect of substitution of values on the translation is the following:*

$$(\!|\, \Gamma \vdash M[x := N] \,|\!)^o_{\vec{k}} = P_{\vec{k}}\ \ {}_y\odot_x^{(\!|\Gamma|\!)/\vec{k}}\ \ (\!|\, \Gamma \vdash M \,|\!)^o_{\vec{k}, x:\ell}.$$

where N is assumed to be a value with $(\!|\, N \,|\!)^y_{\vec{k}} = y \oplus \ell.P_{\vec{k}}$.

Proof. By induction on M. □

Proposition 3.14 (Soundness and adequacy). *Given two terms $\Gamma \vdash M, N : \tau$, if $M \rightarrow_v N$, then $(\!|\, M \,|\!)^{c,o} \equiv (\!|\, N \,|\!)^{c,o}$. Moreover, if $[\![M]\!] \equiv [\![n]\!]$ for $\vdash M : \mathbb{N}$, then $M \rightarrow_v n$.*

Proof. Soundness is an easy verification, using Lemma 3.13.

For adequacy, assume that $[\![M]\!] \equiv [\![n]\!]$. Because there is no recursion, we can normalise M to a term containg $\bot$. Actually, it is easy to see that a normal form of type $\mathbb{N}$ is either a natural number, or a term of the form $E[\bot]$. Now, by induction, we can derive that $(\!| E[\bot] |\!)^{o}_{\vec{k}} \equiv \mathbf{0}$, which is a contradiction since $[\![E[\bot]]\!] = [\![M]\!]$ must not be empty. □

C PROOFS OF § 4

C.1 Proofs of § 4.2

Lemma 4.6. *For $P, Q \triangleright \Delta$, $P \longrightarrow Q$ implies $[\![P]\!] \longrightarrow [\![Q]\!]$.*

Proof. Note that if $\sigma \longrightarrow \sigma'$ then $\tau \odot \sigma \longrightarrow \tau \odot \sigma'$. This implies that the first, third and last rules are sound. The second rule is trivial.

The third rules is a consequence that we have the reduction:

$$(!A \parallel A \xrightarrow{c_A} !A \xrightarrow{d_A} A) \longrightarrow (!A \parallel A \twoheadrightarrow A)$$

where the right most strategy is simply the second projection.

□

C.2 Proofs of § 4.3

To prove Theorem 4.7, we show the following:

Theorem C.1. *For every closed program $\vdash M : \mathbb{N}$, we have $M \approx [\![M]\!]$, by which we mean:*

(1) If $M = \mathsf{new}\, \vec{r} := \vec{k}\ \mathsf{in}\, n$, then $[\![M]\!] = [\![n]\!]$
(2) If $M \to N$ then $[\![M]\!] \longrightarrow [\![N]\!]$
(3) If $[\![M]\!] = [\![n]\!]$ then $M \to^* \mathsf{new}\, \vec{r} := \vec{k}\ \mathsf{in}\, n$
(4) If $[\![M]\!] \longrightarrow \sigma$ then $M \to^* N$ and $[\![N]\!] \approx \sigma$.

Theorem 4.7 follows then by composing the bisimulations together.

C.2.1 Soundness. The first point is straightforward, so we focus on the second one.

Given a context Γ containing only locations , and a memory state $\rho : \mathrm{dom}(\Gamma) \to \mathbb{N}$, we define $\mathsf{mem}^{\rho}_{\Gamma} \stackrel{\text{def}}{=} \|_{r \in \Gamma}\, \mathsf{mem}^{\rho(r)}_{\bar{r}}$.

Given a program $\Gamma \vdash M : \mathbb{N}$ and $\rho : \mathrm{dom}(\Gamma) \to \mathbb{N}$ we write $(\!| M, \rho |\!)^{o}_{i}$ for $(\nu\, \Gamma\, \vec{\Gamma})\, \big((\!| M |\!)^{o}_{i} \parallel \mathsf{mem}^{\rho}_{\Gamma}\big)$.

Lemma C.2. *Consider that $(\!| M, \rho |\!)^{o}_{i} \longrightarrow (\!| M', \rho' |\!)^{o}_{i}$. Then $(\!| E[M], \rho |\!)^{o}_{i} \longrightarrow (\!| E[M'], \rho' |\!)^{o}_{i}$.*

Proof. By induction on E.

- If $E[M] = M; N$: then we have

$$\begin{aligned}(\!| M; N, \rho |\!)^{o}_{i} &\equiv (\nu\, \Gamma\, \bar{\Gamma}) \left((\!| M |\!)^{x}_{\mathsf{q}} \ {}_{x}\!\odot^{\Gamma}_{y}\ y().(\!| N |\!)^{o}_{i} \parallel \mathsf{mem}^{\rho}_{\Gamma} \right) \\ &\equiv (\nu\, x\, y; \Gamma\, \bar{\Gamma}) \left((\!| M |\!)^{x}_{\mathsf{q}} \parallel \mathsf{mem}^{\rho}_{\Gamma} \parallel_{\Gamma} y().(\!| N |\!)^{o}_{i} \right) \\ &\longrightarrow (\nu\, x\, y; \Gamma\, \bar{\Gamma}) \left((\!| M' |\!)^{x}_{\mathsf{q}} \parallel \mathsf{mem}^{\rho'}_{\Gamma} \parallel_{\Gamma} y().(\!| N |\!)^{o}_{i} \right) \\ &\equiv (\!| M'; N, \rho' |\!)^{o}_{i}\end{aligned}$$

- Other cases are similar.

□

Lemma C.3. *Consider programs* $\Gamma \vdash M, N : \mathbb{N}$. *Then, for any* $\rho : dom(\Gamma) \to \mathbb{N}$ *if* $\langle M, \rho \rangle \to^* \langle N, \rho \rangle$, *then* $(\!| M, \rho |\!)^o_i \longrightarrow (\!| N, \rho |\!)^o_i$.

Proof. First, without loss of generality by soundness of the PCF translation, we can assume that M and N are in normal form for $\to_{\mathsf{PCF}}$.

We detail a few rules:

- The rule for reading from a location: $M = {!r}$, $N = \rho(r) = k$. Then we have:

$$\begin{aligned}(\nu\, r\, \bar{r})\left((\!| {!r} |\!)^o_i \parallel \mathsf{mem}^k_{\bar{r}}\right) &\equiv (\nu\, r\, \bar{r})\left((r?[x].\, x \oplus \mathsf{rd}.\, [x \leftrightarrow o]) \parallel \mathsf{mem}^k_{\bar{r}}\right)\\ &\longrightarrow (\nu\, x\, y; r\, \bar{r})\left((x \oplus \mathsf{rd}.\, [r \leftrightarrow o]) \parallel y \,\&\, \left\{\mathsf{rd} : y \oplus k.\, \mathsf{mem}^k_{\bar{r}}; \dots\right\}\right)\\ &\equiv (\nu\, r\, \bar{r})\left(o \oplus k \parallel \mathsf{mem}^k_{\bar{r}}\right)\end{aligned}$$

  The rule for assignment has a similar proof.
- The rule $\langle E[M], \rho \rangle \to \langle E[M'], \rho' \rangle$ with $\langle M, \rho \rangle \to \langle M', \rho' \rangle$ is a consequence of Lemma C.2.

□

C.2.2 Adequacy. To prove adequacy,we first need to define some kind of Bohm trees of terms of IPA by reducing all possible redexes $\to_{\mathsf{PCF}}$. Given a term M we define its Bohm tree $\mathsf{BT}(M)$ as follows:

$$\mathsf{BT}(M) \stackrel{\text{def}}{=} \begin{cases} \bot & \text{if } M \text{ does not have a head normal form for } \to_{\mathsf{PCF}}\\ {!r} & \text{If } M \to^*_{\mathsf{PCF}} {!r}\\ r := \mathsf{BT}(N) & \text{If } M \to^*_{\mathsf{PCF}} r := N\\ \mathsf{succ}(N) & \text{If } M \to^*_{\mathsf{PCF}} \mathsf{succ}(N)\\ \mathsf{iszero}(N) & \text{If } M \to^*_{\mathsf{PCF}} \mathsf{iszero}(N)\\ r := \mathsf{BT}(N) & \text{If } M \to^*_{\mathsf{PCF}} r := N\\ \mathsf{BT}(N_1); \mathsf{BT}(N_2) & \text{If } M \to^*_{\mathsf{PCF}} N_1; N_2\\ \mathsf{BT}(N_1) \parallel \mathsf{BT}(N_2) & \text{If } M \to^*_{\mathsf{PCF}} N_1 \parallel N_2\\ \mathsf{new}\, r := k \,\mathsf{in}\, \mathsf{BT}(N) & \text{If } M \to^*_{\mathsf{PCF}} \mathsf{new}\, r := k \,\mathsf{in}\, \mathsf{BT}(N)\\ \mathsf{if}\, \mathsf{BT}(M)\, \mathsf{BT}(N_1)\, \mathsf{BT}(N_2) & \text{If } M \to^*_{\mathsf{PCF}} \mathsf{if}\, \mathsf{BT}(M)\, \mathsf{BT}(N_1)\, \mathsf{BT}(N_2) \end{cases}$$

Note that the head normal form cannot be a variable, a λ-abstraction or an application because M is a program. Bohm trees are in general infinite.

Lemma C.4. *For any program* M, *we have* $(\!| M |\!)^o \equiv (\!| BT(M) |\!)^o$.

A **normal form** a program M such that $\mathsf{BT}(M) = M$. The adequacy result follows from this result. Let us write $[\![\langle M, \rho \rangle]\!]$ for $[\![(\!| \langle M, \rho \rangle |\!)^o_{\mathsf{q}}]\!]$.

To prove the adequacy result, we will rely on this result:

Lemma C.5. *Consider a minimal event of* $[\![P \;{}_a\odot^{\Lambda}_b\; b \,\&\, \{Q_i\}_{i \in I}]\!]$ *where the type of* A *is* $\oplus_{i \in I} T_i$. *It is either:*

- *A minimal event of* $[\![P]\!]$ *not on* a
- *A minimal event of* $[\![Q_k]\!]$, *and* $[\![P]\!]$ *has an initial event output* k *to* a.

Proof. This is a simple reasoning on the composition. $[\![P]\!]$ and $[\![Q]\!]$ synchronise on $[\![\oplus_{i \in I} T_i$ which means that a most $[\![P]\!]$ sends a move to $[\![Q]\!]$. As a result, a minimal move of the process is either:

- A minimal move of $[\![P]\!]$ before any move on a
- A move of $[\![P]\!]$ straight after a move on a – impossible by courtesy. (a is positive)
- A move of $[\![b \,\&\, \{i.\, Q_i\}_{i\in I}]\!]$ after a reception on b, ie. a minimal move of $[\![Q_k]\!]$. Moreover, if that the case, then the causal history of this move in the interaction contains an output from P on a as desired. By courtesy this move must be minimal.

□

Lemma C.6. *Let* $\Gamma \vdash M : A$ *be a program of base type* A *and* $\rho : dom(\Gamma) \to \mathbb{N}$ *a memory state. Consider* e *a minimal event of* $[\![\langle M, \rho\rangle]\!]$.

- *If* e *is a neutral event, then there exists a reduction* $\langle M, \rho\rangle \to^* \langle M'\!,\rho'\rangle$ *with* $[\![\langle M', \rho'\rangle]\!] \approx [\![\langle M, \rho\rangle]\!]/e$
- *If* e *is a value* v *of* A, *then* $\langle M, \rho\rangle \to^* \langle \mathtt{new}\, \vec{r} := \vec{k}\ \mathtt{in}\, v, \rho'\rangle$.

Proof. Without loss of generality, we can assume that M is a normal form. Moreover, since we know that e is a minimal event of $[\![\langle M, \rho\rangle]\!]$ by continuity, there must exist a finite approximation M_n (ie. M where branches deeper than n are cut) of M such that $[\![\langle M_n, \rho\rangle]\!]$ contains e. We then proceed by induction on M_n.

We detail a few cases, but most are similar using Lemma C.5:

- If $M = !N$, then there are two cases:
  - Either $e = n$: then by induction we know that $\langle N, \rho\rangle$ must normalise to $\langle r, \rho'\rangle$. Then $\langle M, \rho\rangle \to^* \langle \rho'(r), \rho'\rangle$.
  - Or e is an internal event. Then e corresponds to an internal event of $[\![\langle N, \rho\rangle]\!]$ so by induction, $\langle N, \rho\rangle$ reduces to $\langle N', \rho'\rangle$, and the desired state is $\langle !N', \rho'\rangle$.
- $M = \mathtt{new}\, r := k\ \mathtt{in}\, N$. Then we directly apply the induction hypothesis to $\langle N, \rho[r := k]\rangle$.
- $N \parallel N'$. We us Lemma C.5. There are two main cases:
  - If the minimal event is in $[\![N]\!] \parallel [\![N']\!]$, then we apply the induction hypothesis
  - Otherwise, it must be that $[\![N]\!]$ and $[\![N']\!]$ both have an initial event outputting done, and by induction we know that N and N' reduce to `skip` and then we can conclude.

□

From this result, Theorem C.1 follows.