Games, Mobile Processes, and Functions

Guilhem Jaber
Université de Nantes, France

Davide Sangiorgi
Università di Bologna, Italy
Inria Sophia Antipolis, France

Abstract

We establish a tight connection between two models of the λ-calculus, namely Milner's encoding into the π-calculus (precisely, the Internal π-calculus), and operational game semantics (OGS). We first investigate the operational correspondence between the behaviours of the encoding provided by π and OGS.

We do so for various LTSs: the standard LTS for π and a new "concurrent" LTS for OGS; an "output-prioritised" LTS for π and the standard alternating LTS for OGS. We then show that the equivalences induced on λ-terms by all these LTSs (for π and OGS) coincide.

These connections allow us to transfer results and techniques between π and OGS. In particular we import up-to techniques from π onto OGS and we derive congruence and compositionality results for OGS from those of π. The study is illustrated for call-by-value; similar results hold for call-by-name.

2012 ACM Subject Classification Theory of computation → Program semantics

Keywords and phrases λ-calculus, π-calculus, game semantics

Digital Object Identifier 10.4230/LIPIcs.CSL.2022.25

Related Version *Full Version*: `https://hal.archives-ouvertes.fr/hal-03407123` [21]

Funding *Davide Sangiorgi*: supported by the MIUR-PRIN project "Analysis of Program Analyses" (ASPRA, ID 201784YSZ5_004).

1 Introduction

The topic of the paper is the comparison between *Operational Game semantics* (OGS) and the *π-calculus*, as generic models or frameworks for the semantics of higher-order languages.

Game semantics [4, 20] provides intentional models of higher-order languages, where the denotation of a program brings up its possible interactions with the surrounding context. Distinct points of game semantics are the rich categorical structure and the emphasis on compositionality. Game semantics provides a modular characterization of higher-order languages with computational effects like control operators [25], mutable store [3, 5] or concurrency [15, 27]. This gives rise to the "Semantic Cube" [2], a characterization of the absence of such computational effects in terms of appropriate restrictions on the interactions, with conditions like *alternation, well-bracketing, visibility* or *innocence*. For instance, well-bracketing corresponds to the absence of control operators like call/cc.

Game semantics has spurred Operational Game Semantics (OGS) [16, 23, 24, 28, 30], as a way to describe the interactions of a program with its environment by embedding programs into appropriate configurations and then defining rules that turn such configurations into an LTS. Besides minor differences on the representation of causality between actions, the main distinction with "standard" game semantics is in the way in which the denotation of programs is obtained: via an LTS, rather than, compositionally, by induction on the structure of the programs (or their types). It is nonetheless possible to establish a formal correspondence between these two representations [30].

OGS is particularly effective on higher-order programs. To avoid being too intensional, functional values exchanged between the program and its environment are represented as atoms, seen as free variables. Therefore OGS configurations include open terms. The basic actions in the LTS produced by OGS represent the calls and returns of functions between a program and its environment. The OGS semantics has been shown fully-abstract, that is, to characterize observational equivalence, for a wide class of programming languages, including effectful subsets of ML [22, 28], fragments of Java [24], aspect-oriented programs [23]. The conditions in the above-mentioned Semantic Cube (alternation, well-bracketing, etc.) equally apply to OGS.

In this paper, we consider forms of OGS for the pure untyped call-by-value λ-calculus, which enforce some of such conditions. Specifically we consider: an *Alternating* OGS, where only one term can be run at a time, and the control on the interactions alternates between the term and the environment; and a *Concurrent* OGS, where multiple terms can be run in parallel.

The π-calculus is the paradigmatical name-passing calculus, that is, a calculus where names (a synonym for "channels") may be passed around. In the literature about the π-calculus, and more generally in Programming Language theory, Milner's work on functions as processes [33], which shows how the evaluation strategies of *call-by-name λ-calculus* and *call-by-value λ-calculus* [1, 35] can be faithfully mimicked, is generally considered a landmark. The work promotes the π-calculus to be a model for higher-order programs, and provides the means to study λ-terms in contexts other than the purely sequential ones and with the instruments available to reason about processes. In the paper, π-calculus is actually meant to be the *Internal π-calculus* (Iπ), a subset of the original π-calculus in which only fresh names may be exchanged among processes [41]. The use of Iπ avoids a few shortcomings of Milner's encodings, notably for call-by-value; e.g., the failure of the β_v rule (i.e., the encodings of $(\lambda x. M)V$ and $M\{^V\!/\!_x\}$ may be behaviourally distinguishable in π).

Further investigations into Milner's encodings [11, 42] have revealed what is the equivalence induced on λ-terms by the encodings, whereby two λ-terms are equal if their encodings are behaviourally equivalent (i.e., bisimilar) Iπ terms. In call-by-value, this equivalence is *eager normal-form bisimilarity* [29], a tree structure proposed by Lassen (and indeed sometimes referred to as "Lassen's trees") as the call-by-value counterpart of Böhm Trees (or Lévy-Longo Trees).

In a nutshell, when used to give semantics to a language, major strengths of the π-calculus are its algebraic structure and the related algebraic properties and proof techniques; major strengths of OGS are its proximity to the source language – the configurations of OGS are built directly from the terms of the source language, as opposed to an encoding as in the π-calculus – and its flexibility – the semantics can be tuned to account for specific features of the source language like control operators or references.

The general goal of this paper is to show that there is a tight and precise correspondence between OGS and π-calculus as models of programming languages and that such a correspondence may be profitably used to take advantage of the strengths of the two models. We carry out the above program in the specific case of (untyped) call-by-value λ-calculus, Λ_{V}, which is richer and (as partly suggested above) with some more subtle aspects than call-by-name. However similar results also hold for call-by-name; see [21] for the technical details for more comments on it. Analogies and similarities between game semantics and π-calculus have been pointed out in various papers in the literature (e.g., [6, 19]; see Section 9), and used to, e.g., explain game semantics using π-like processes, and enhance type systems for π-terms. In this paper, in contrast, we carry out a direct comparison between the two models, on their interpretation of functions.

We take the (arguably) canonical representations of Λ_V into Iπ and OGS. The latter representation is Milner's encoding, rewritten in Iπ. We consider two variant behaviours for the Iπ terms, respectively produced by the ordinary LTS of Iπ, and by an "output-prioritised" LTS, opLTS, in which input actions may be observed only in the absence of outputs and internal actions. Intuitively, the opLTS is intended to respect sequentiality constraints in the Iπ terms: an output action stands for an ongoing computation (for instance, returning the result of a previous request) whereas an input action starts a new computation (for instance, a request of a certain service); therefore, in a sequential system, an output action should have priority over input actions. For OGS, the Λ_V representation is the straightforward adaptation of the OGS representations of typed λ-calculi in the literature, e.g., [28].

We then develop a thorough comparison between the behaviours of the OGS and Iπ representations. For this we define a mapping from OGS configurations to Iπ processes. We also exploit the fact that, syntactically, the actions in the OGS and Iπ LTSs are the same. We derive a tight correspondence between the two models, which allows us to transfer techniques and to switch freely between the two models in the analysis of the OGS and Iπ representations of Λ_V, so to establish new results or obtain new proofs. On these aspects, our main results are the following:

1. We show that the representation of Λ_V in the Alternating OGS is behaviourally the same as the representation in Iπ assuming the opLTS. Thus the semantics on λ-terms induced by the OGS and Iπ representations coincide. The same results are obtained between the Concurrent OGS and Iπ under its ordinary LTS.
2. We transfer "bisimulation up-to techniques" for Iπ, notably a form of "up-to context", onto (Concurrent) OGS. The result is a powerful technique, called "up-to composition" that allows us to split an OGS configuration into more elementary configurations during the bisimulation game.
3. We show that the semantics induced on Λ_V by the Alternating and by the Concurrent OGS are the same, both when the equality in OGS is based on traces and when it is based on bisimulation. In other words, all the OGS views of Λ_V (Alternating or Concurrent, traced-based or bisimulation-based) coincide. Moreover, we show that such induced semantics is the equality of Lassen's trees. We derive the result in two ways: one in which we directly import it from Iπ; the other in which we lift eager normal-form bisimulations into OGS bisimulations via the up-to-composition technique.
4. We derive congruence and compositionality properties for the OGS semantics, as well as a notion of tensor product over configurations that computes interleavings of traces.

The results about OGS in (2-4) are obtained exploitingthe mapping into Iπ and its algebraic properties and proof techniques, as well as the up-to-composition technique for OGS imported from Iπ.

Structure of the paper. Sections 2 to 5 contain background material: general notations, Iπ, Λ_V, the representations of Λ_V in the Alternating OGS (A-OGS) and in Iπ. The following sections contain the new material. In Section 6 we study the relationship between the two Λ_V representations, in Iπ using the output-prioritised LTS. In Section 7 we establish a similar relationship between a new Concurrent OGS (C-OGS) and Iπ using its ordinary LTS. We also transport up-to techniques onto OGS, and prove that all the semantics of Λ_V examined (OGS, Iπ, traces, bisimulations) coincide. We import compositionality results for OGS from Iπ in Section 8.

2 Notations

In the paper, we use various LTSs and behavioural relations for them, both for OGS and for the π-calculus. In this section, we introduce or summarise common notations.

We use a tilde, like in $\widetilde{a}$, for (possibly empty) tuples of objects (usually names). Let $K \xrightarrow{\mu}_{\mathrm{g}} K'$ be a generic LTS (for OGS or Iπ; the grammar for actions in the LTSs for OGS and Iπ will be the same). Actions, ranged over by μ, can be of the form $a(\widetilde{b})$, $\overline{a}(\widetilde{b})$, τ, and $(\widetilde{a})$, where τ, called *silent* or (*invisible*) action, represents an internal step in K, that is, an action that does not require interaction with the outside, and $(\widetilde{a})$ is a special action performed by abstractions in Iπ and initial configurations in OGS. If $\mu \neq \tau$ then μ is a *visible* action; we use ℓ to range over them. We sometimes abbreviate $\xrightarrow{\tau}_{\mathrm{g}}$ as $\longrightarrow_{\mathrm{g}}$. We write $\Longrightarrow_{\mathrm{g}}$ for the reflexive and transitive closure of $\xrightarrow{\tau}_{\mathrm{g}}$. We also write $K \xRightarrow{\mu}_{\mathrm{g}} K'$ if $K \Longrightarrow_{\mathrm{g}} \xrightarrow{\mu}_{\mathrm{g}} \Longrightarrow_{\mathrm{g}} K'$ (the composition of the three relations. Then $\xRightarrow{\widehat{\mu}}_{\mathrm{g}}$ is $\xRightarrow{\mu}_{\mathrm{g}}$ if $\mu \neq \tau$, and $\Longrightarrow_{\mathrm{g}}$ if $\mu = \tau$.

Traces, ranged over by s, are finite (and possibly empty) sequences of visible actions. If $s = \ell_1, \dots, \ell_n$ $(n \geq 0)$, then $K \xRightarrow{s}_{\mathrm{g}} K'$ holds if there are $K_0, \dots, K_n$ with $K_0 = K$, $K_n = K'$, and $K_i \xRightarrow{\ell_{i+1}}_{\mathrm{g}} K_{i+1}$ for $0 \leq i < n$; and $K \xRightarrow{s}_{\mathrm{g}}$ if there is K' with $K \xRightarrow{s}_{\mathrm{g}} K'$.

Two states K_1, K_2 of the LTS are *trace equivalent*, written $K_1 \simeq_{\mathrm{g}} K_2$, if $(K_1 \xRightarrow{s}_{\mathrm{g}}$ iff $K_2 \xRightarrow{s}_{\mathrm{g}})$, for all s.

Similarly, *bisimilarity*, written $\approx_{\mathrm{g}}$, is the largest symmetric relation on the state of the LTS such that whenever $K_1 \approx_{\mathrm{g}} K_2$ then $K_1 \xRightarrow{\mu}_{\mathrm{g}} K_1'$ implies there is K_2' with $K_2 \xRightarrow{\widehat{\mu}}_{\mathrm{g}} K_2'$ and $K_1' \approx_{\mathrm{g}} K_2'$. For instance, in the I$\pi$ LTS $\xrightarrow{\mu}_{\pi}$ of Section 3.1, $P \simeq_{\pi} Q$ means that the Iπ processes P and Q are trace equivalent, and $P \approx_{\pi} Q$ means that they are bisimilar.

▶ Remark 1 (bound names). In an action $a(\widetilde{b})$ or $\overline{a}(\widetilde{b})$ or $(\widetilde{b})$, name a is *free* whereas $\widetilde{b}$ are *bound*; the free and bound names of a trace are defined accordingly. Throughout the paper, in any statement (concerning OGS or Iπ), the bound names of an action or of a trace that appears in the statement are supposed to be all *fresh*; i.e., all distinct from each other and from the free names of the objects in the statement.

3 Background

3.1 The Internal π-calculus

The Internal π-calculus, Iπ, is, intuitively, a subset of the π-calculus in which all outputs are bound. This is syntactically enforced by having outputs written as $\overline{a}(\widetilde{b})$ (which in the π-calculus would be an abbreviation for $\boldsymbol{\nu}\widetilde{b}\,\overline{a}\langle\widetilde{b}\rangle$). All tuples of names in Iπ are made of pairwise distinct components. *Abstractions* are used to write name-parametrised processes, for instance, when writing recursive process definitions. The instantiation of the parameters of an abstraction B is done via the *application* construct $B\langle\widetilde{a}\rangle$. Processes and abstractions form the set of *agents*, ranged over by T. Lowercase letters $a, b, \dots, x, y, \dots$ range over the infinite set of names. The grammar of Iπ is thus:

$$\begin{array}{lclr} P & \triangleq & 0 \mid a(\widetilde{b}).P \mid \overline{a}(\widetilde{b}).P \mid \boldsymbol{\nu}a\,P \mid P_1 \mid P_2 \mid !a(\widetilde{b}).P \mid B\langle\widetilde{a}\rangle & \text{(processes)} \\ B & \triangleq & (\widetilde{a})\;P \mid \mathtt{K} & \text{(abstractions)} \end{array}$$

The operators have the usual meaning; we omit the standard definition of free names, bound names, and names of an agent, respectively indicated with $\mathtt{fn}(-)$, $\mathtt{bn}(-)$, and $\mathtt{n}(-)$.

In the grammar, $\mathtt{K}$ is a constant, used to write recursive definitions. Each constant $\mathtt{K}$ has a defining equation of the form $\mathtt{K} \triangleq (\tilde{x})\ P$, where $(\tilde{x})\ P$ is name-closed (that is, without free names); $\tilde{x}$ are the formal parameters of the constant. Replication could be avoided in the syntax since it can be encoded with recursion. However, its semantics is simple, and it is useful in encodings.

An application redex $((\widetilde{x})P)\langle\widetilde{a}\rangle$ can be normalised as $P\{\widetilde{a}/\widetilde{x}\}$. An agent is *normalised* if all such application redexes have been contracted. In the remainder of the paper *we identify an agent with its normalised expression.*

Since the calculus is polyadic, we assume a *sorting system* [32] to avoid disagreements in the arities of the tuples of names Being not essential, it will not be presented here.

Operational semantics and behavioural relations

In the LTS for Iπ, recalled in [21] transitions are of the form $T \xrightarrow{\mu}_\pi T'$, where the bound names of μ are fresh, i.e., they do not appear free in T.

Trace equivalence ($\simeq_\pi$) and bisimilarity ($\approx_\pi$) have been defined in Section 2. We refer to [21] for the standard definition of *expansion*, written $\lesssim_\pi$. (The expansion relation $\lesssim_\pi$ is an asymmetric variant of $\approx_\pi$ in which, intuitively, $P \lesssim_\pi Q$ holds if $P \approx_\pi Q$ but also Q has at least as many τ-moves as P.) All behavioural relations are extended to abstractions by requiring ground instantiation of the parameters; this is expressed by means of a transition; e.g., the action $(\tilde{x})\ P \xrightarrow{(\tilde{x})}_\pi P$.

The "up-to" techniques

The "up-to" techniques allow us to reduce the size of a relation $\mathcal{R}$ to exhibit for proving bisimilarities. Our main up-to technique will be *up-to context and expansion* [40], which admits the use of contexts and of behavioural equivalences such as expansion to achieve the closure of a relation in the bisimulation game. So the bisimulation clause becomes:

- if $P\ \mathcal{R}\ Q$ and $P \xrightarrow{\mu} P''$ then there are a static context $C_{\mathtt{ctx}}$ and processes P' and Q' s.t. $P''\ {}_\pi\!\gtrsim C_{\mathtt{ctx}}[P']$, $Q \overset{\widehat{\mu}}{\Longrightarrow}\ {}_\pi\!\gtrsim C_{\mathtt{ctx}}[Q']$ and $P'\ \mathcal{R}\ Q'$ $(*)$

where a *static context* is a context of the form $\boldsymbol{\nu}\tilde{c}\,(R \mid [\cdot])$.

We will also employ: *bisimulation up-to* $\approx_\pi$ [31], whereby bisimilarity itself is employed to achieve the closure of the candidate relation during the bisimulation game; a variant of bisimulation up-to context and expansion, called *bisimulation up-to context and up-to* $({}_\pi\!\gtrsim, \approx_\pi)$, in which, in $(*)$, when μ is a visible action, expansion is replaced by the coarser bisimilarity, at the price of imposing that the static context $C_{\mathtt{ctx}}$ cannot interact with the processes P or Q. (This technique, as far as we know, does not appear in the literature.) Details on these techniques may be found in [21].

3.2 The Call-By-Value λ-calculus

The grammar of the untyped call-by-value λ-calculus, Λ_{V}, has values V, terms M, evaluation contexts E, and general contexts C:

$$\begin{array}{llcl@{\qquad}llcl}
\text{Vals} & V & \triangleq & x \mid \lambda x.\, M & \text{ECtxs} & E & \triangleq & [\cdot] \mid VE \mid EM \\
\text{Terms} & M,N & \triangleq & V \mid MN & \text{Ctxs} & C & \triangleq & [\cdot] \mid \lambda x.\, C \mid MC \mid CM
\end{array}$$

where $[\cdot]$ stands for the hole of a context. The call-by-value reduction $\to_{\mathtt{v}}$ has two rules:

$$\frac{}{(\lambda x.\, M)V \to_{\mathtt{v}} M\{V/x\}} \qquad \frac{M \to_{\mathtt{v}} N}{E[M] \to_{\mathtt{v}} E[N]}$$

In the following, we write $M \Downarrow M'$ to indicate that $M \Rightarrow_{\mathrm{v}} M'$ with M' an eager normal form, that is, either a value or a stuck call $E[xV]$.

4 Operational Game Semantics

We introduce the representation of Λ_{V} in OGS. The LTS produced by the embedding of a term intends to capture the possible interactions between this term and its environment. Values exchanged between the term and the environment are represented by names, akin to free variables, called *variable names* and ranged over by x, y, z. Continuations (i.e., evaluation contexts) are also represented by names, called *continuation names* and ranged over by p, q, r.

Actions μ have been introduced in Section 2. In OGS, we have five kinds of (visible) actions:

- *Player Answers* (PA), $\bar{p}(x)$, and *Opponent Answers* (OA), $p(x)$, that exchange a variable x through a continuation name p;
- *Player Questions* (PQ), $\bar{x}(y, p)$, and *Opponent Questions* (OQ), $x(y, p)$, that exchange a variable y and a continuation name p through a variable x;
- *Initial Opponent Questions* (IOQ), (p), that introduce the initial continuation name p.

▶ Remark 2. The denotation of terms is usually represented in game semantics using the notion of *pointer structure* rather than traces. A pointer structure is defined as a sequence of *moves*, together with a pointer from each move (but the initial one) to a previous move that "justifies" it. Taking a trace s, one can reconstruct this pointer structure in the following way: an action μ is *justified* by an action μ' if the free name of μ is bound by μ' in s (here we are taking advantage of the "freshness" convention on the bound names of traces, Remark 1).

Environments, ranged over by γ, maintain the association from names to values and evaluations contexts, and are partial maps. A single mapping is either of the form $[x \mapsto V]$ (the variable x is mapped onto the value V), or $[p \mapsto (E, q)]$ (the continuation name p is mapped onto the pair of the evaluation context E and the continuation q).

There are two main kinds of configurations F: *active configurations* $\langle M, p, \gamma, \phi \rangle$ and *passive configurations* $\langle \gamma, \phi \rangle$, where M is a term, p a continuation name, γ an environment and ϕ a set of names called its *name-support*. Names in $\mathrm{dom}(\gamma)$ are called *P-names*, and those in $\phi \backslash \mathrm{dom}(\gamma)$ are called *O-names*. So we obtain a *polarity function* pol_F associated to F, defined as the partial maps from ϕ to $\{O, P\}$ mapping names to their polarity. In the following, we only consider *valid configurations*, for which:

- $\mathrm{dom}(\gamma) \subseteq \phi$
- $\mathtt{fv}(M), p$ are O-names;
- for all $a \in \mathrm{dom}(\gamma)$, the names appearing in $\gamma(a)$ are O-names.

The LTS is introduced in Figure 1. It is called *Alternating*, since, forgetting the Pτ transition, it is bipartite between active configurations that perform Player actions and passive configurations that perform Opponent actions. Accordingly, we call Alternating the resulting OGS, abbreviated A-OGS. In the OA rule, E is "garbage-collected" from γ, a behavior corresponding to *linear continuations*.

The term of an active configuration determines the next transition performed. First, the term needs to be reduced, using the rule (Pτ). When the term is a value V, a Player Answer (PA) is performed, providing a fresh variable x to Opponent, while V is stored in γ at position x. Freshness is enforced using the disjoint union $\uplus$. When the term is a callback $E[xV]$, with p the current continuation name, a Player Question (PQ) at x is performed, providing two fresh names y, q to Opponent, while storing V at y and (E, p) at q in γ.

$$\begin{array}{l|llll}
(P\tau) & \langle M, p, \gamma, \phi\rangle & \xrightarrow{\tau}_{\mathtt{a}} & \langle N, p, \gamma, \phi\rangle & \text{when } M \rightarrow_{\mathtt{v}} N \\
(PA) & \langle V, p, \gamma, \phi\rangle & \xrightarrow{\bar{p}(x)}_{\mathtt{a}} & \langle \gamma \cdot [x \mapsto V], \phi \uplus \{x\}\rangle & \\
(PQ) & \langle E[xV], p, \gamma, \phi\rangle & \xrightarrow{\bar{x}(y,q)}_{\mathtt{a}} & \langle \gamma \cdot [y \mapsto V] \cdot [q \mapsto (E,p)], \phi \uplus \{y, q\}\rangle & \\
(OA) & \langle \gamma \cdot [q \mapsto (E,p)], \phi\rangle & \xrightarrow{q(x)}_{\mathtt{a}} & \langle E[x], p, \gamma, \phi \uplus \{x\}\rangle & \\
(OQ) & \langle \gamma, \phi\rangle & \xrightarrow{x(y,p)}_{\mathtt{a}} & \langle Vy, p, \gamma, \phi \uplus \{y, p\}\rangle & \text{when } \gamma(x) = V \\
(IOQ) & \langle [? \mapsto M], \phi\rangle & \xrightarrow{(p)}_{\mathtt{a}} & \langle M, p, \varepsilon, \phi \uplus \{p\}\rangle &
\end{array}$$

Figure 1 The LTS for the Alternating OGS (A-OGS).

$$\mathcal{V}[\![V]\!] \triangleq (p)\ \overline{p}(y).\,\mathcal{V}^*[\![V]\!]\langle y\rangle \qquad \mathcal{V}^*[\![\lambda x.\,M]\!] \triangleq (y)\ !y(x,q).\,\mathcal{V}[\![M]\!]\langle q\rangle \qquad \mathcal{V}^*[\![x]\!] \triangleq (y)\ y \triangleright x$$
$$\mathcal{V}[\![MN]\!] \triangleq (p)\ \boldsymbol{\nu} q\ \big(\mathcal{V}[\![M]\!]\langle q\rangle \mid q(y).\,\boldsymbol{\nu} r\ \big(\mathcal{V}[\![N]\!]\langle r\rangle \mid r(w).\,\overline{y}(w',p').\,(w' \triangleright w \mid p' \triangleright p)\big)\big)$$

Figure 2 The encoding of call-by-value λ-calculus into Iπ.

On passive configurations, Opponent has the choice to perform different actions. It can perform an Opponent Answer (OA) by interrogating an evaluation context E stored in γ. For this, Opponent provides a fresh variable x that is plugged into the hole of E, while the continuation name q associated to E in γ is restored. Opponent may also perform an Opponent Question (OQ), by interrogating a value V stored in γ. For this, Opponent provides a fresh variable y as an argument to V.

To build the denotation of a term M, we introduce an *initial configuration* associated with it, written $\langle [? \mapsto M], \phi\rangle$, with ϕ the set of free variables we start with. When this set is taken to be the free variables of M, we simply write it as $\langle M\rangle$. In the initial configuration, the choice of the continuation name p is made by performing an Initial Opponent question (IOQ). (Formally, initial configurations should be considered as passive configurations.)

5 The encoding of call-by-value λ-calculus into the π-calculus

We recall here Milner's encoding of call-by-value λ-calculus, transplanted into Iπ. The core of any encoding of the λ-calculus into a process calculus is the translation of function application. This becomes a particular form of parallel combination of two processes, the function, and its argument; β-reduction is then modelled as a process interaction.

As in OGS, so in Iπ the encoding uses *continuation names* $p, q, r, \ldots$, and *variable names* $x, y, v, w \ldots$. Figure 2 presents the encoding. Process $a \triangleright b$ represents a *link* (sometimes called forwarder; for readability we have adopted the infix notation $a \triangleright b$ for the constant $\triangleright$). It transforms all outputs at a into outputs at b thus the body of $a \triangleright b$ is replicated, unless a and b are continuation names:

$$\triangleright \quad \triangleq \quad \begin{cases} (p,q).\,p(x).\,\overline{q}(y).\,y \triangleright x & \text{if } p, q \text{ are continuation names} \\ (x,y).\,!x(z,p).\,\overline{y}(w,q).\,(q \triangleright p \mid w \triangleright z) & \text{if } x, y \text{ are variable names} \end{cases}$$

The equivalence induced on call-by-value λ-terms by their encoding into Iπ coincides with Lassen's *eager normal-form (enf) bisimilarity* [29]. That is, $\mathcal{V}[\![M]\!] \approx_\pi \mathcal{V}[\![N]\!]$ iff M and N are enf-bisimilar [11]. In proofs about the behaviour of the Iπ representation of λ-terms we sometimes follow [11] and use an optimisation of Milner's encoding, reported in [21].

Encoding of environments:			Encoding of configurations:		
$\mathcal{V}[\![[y \mapsto V] \cdot \gamma']\!]$	$\triangleq$	$\mathcal{V}^*[\![V]\!]\langle y \rangle \mid \mathcal{V}[\![\gamma']\!]$	$\mathcal{V}[\![\langle M, p, \gamma, \phi \rangle]\!]$	$\triangleq$	$\mathcal{V}[\![M]\!]\langle p \rangle \mid \mathcal{V}[\![\gamma]\!]$
$\mathcal{V}[\![[q \mapsto (E, p)] \cdot \gamma']\!]$	$\triangleq$	$q(x).\, \mathcal{V}[\![E[x]]\!]\langle p \rangle \mid \mathcal{V}[\![\gamma']\!]$	$\mathcal{V}[\![\langle \gamma, \phi \rangle]\!]$	$\triangleq$	$\mathcal{V}[\![\gamma]\!]$
$\mathcal{V}[\![\varepsilon]\!]$	$\triangleq$	0	$\mathcal{V}[\![\langle [? \mapsto M], \phi \rangle]\!]$	$\triangleq$	$\mathcal{V}[\![M]\!]$

Figure 3 From OGS environments and configurations to Iπ.

6 Relationship between Iπ and A-OGS

To compare the A-OGS and Iπ representations of the (call-by-value) λ-calculus, we set a mapping from A-OGS configurations and environments to Iπ processes. The mapping is reported in Figure 3. It is an extension of Milner's encoding of the λ-calculus and is therefore indicated with the same symbol $\mathcal{V}$. The mapping uses a representation of environments γ as associative lists.

▶ Remark 3. The encoding of a configuration F with name-support ϕ does not depend on ϕ. This name-support ϕ is used in OGS both to enforce freshness of names, and to deduce the polarity of names, as represented by the function pol. And indeed, the process $\mathcal{V}[\![F]\!]$ has its set of free names included in ϕ, and uses P-names in outputs and O-names in inputs. The polarity property could be stated in π-calculus using i/o-sorting [34]. Indeed, a correspondence between arenas of game semantics (used to enforce polarities of moves) and sorting has been explored [18, 19].

6.1 Operational correspondence

The following theorems establish the operational correspondence between the A-OGS and Iπ representations. In Theorem 4, as well as in following theorems such as Theorems 5, 7, and 13, the appearance of the expansion relation ${}_{\pi}\gtrsim$ (in place of the coarser $\approx_\pi$), in the statement about silent actions, is essential, both to derive the statement in the theorems about visible actions and to use the theorems in up-to techniques for Iπ (more generally, in applications of the theorems in which one reasons about the number of steps performed).

▶ **Theorem 4.**
1. *If $F \Longrightarrow_{\mathsf{a}} F'$, then $\mathcal{V}[\![F]\!] \Longrightarrow_\pi {}_{\pi}\gtrsim \mathcal{V}[\![F']\!]$;*
2. *If $F \overset{\ell}{\Longrightarrow}_{\mathsf{a}} F'$, then $\mathcal{V}[\![F]\!] \overset{\ell}{\Longrightarrow}_\pi \approx_\pi \mathcal{V}[\![F']\!]$.*

▶ **Theorem 5.**
1. *If $\mathcal{V}[\![F]\!] \Longrightarrow_\pi P$ then there is F' such that $F \Longrightarrow_{\mathsf{a}} F'$ and $P \;{}_{\pi}\gtrsim \mathcal{V}[\![F']\!]$;*
2. *If $\mathcal{V}[\![F]\!] \overset{\ell}{\Longrightarrow}_\pi P$ and ℓ is an output, then there is F' such that $F \overset{\ell}{\Longrightarrow}_{\mathsf{a}} F'$ and $P \;{}_{\pi}\gtrsim \mathcal{V}[\![F']\!]$;*
3. *If F is passive and $\mathcal{V}[\![F]\!] \overset{\ell}{\Longrightarrow}_\pi P$, then there is F' such that $F \overset{\ell}{\Longrightarrow}_{\mathsf{a}} F'$ and $P \approx_\pi \mathcal{V}[\![F']\!]$.*

In Theorem 5, a clause is missing for input actions from $\mathcal{V}[\![F]\!]$ when F active. Indeed such actions are possible in Iπ, stemming from the (encoding of the) environment of F, whereas they are not possible in A-OGS. This is rectified in Section 6.2, introducing a constrained LTS for Iπ, and in Section 7, considering a concurrent OGS.

▶ **Corollary 6.** *If $F \overset{s}{\Longrightarrow}_{\mathsf{a}}$ then also $\mathcal{V}[\![F]\!] \overset{s}{\Longrightarrow}_\pi$.*

6.2 An output-prioritised Transition System

We define an LTS for Iπ in which input actions are visible only if no output can be consumed, either as a visible action or through an internal action (i.e., syntactically the process has no unguarded output). The new LTS, called *output-prioritised* and indicated as opLTS, is defined on the top of the ordinary one by means of the two rules below. A process P is *input reactive* if whenever $P \xrightarrow{\mu}_{\pi} P'$, for some μ, P' then μ is an input action.

$$\frac{P \xrightarrow{\mu}_{\pi} P' \qquad P \text{ input reactive}}{P \xrightarrow{\mu}_{\mathsf{o}\pi} P'} \qquad\qquad \frac{P \xrightarrow{\mu}_{\pi} P'}{P \xrightarrow{\mu}_{\mathsf{o}\pi} P'}\mu \text{ is an output or } \tau \text{ action}$$

The opLTS captures an aspect of *sequentiality* in π-calculi: a free input prefix is to be thought of as a service offered to the external environment; in a sequential system such a service is available only if there is no ongoing computations due to previous interrogations of the server. An ongoing computation is represented by a τ-action, indicating a step of computation internal to the server, or an output, indicating either an answer to a client or a request to an external server. The constraint imposed by the new LTS could also be formalised compositionally, see [21].

Under the opLTS, the analogous of Theorem 4 continue to hold: in A-OGS configurations, input transitions only occur in passive configurations, and the encodings of passive configurations are input-reactive processes. However, now we have the full converse of Theorem 5 and, as a consequence, we can also establish the converse direction of Corollary 6.

▶ **Theorem 7.**
1. *If* $\mathcal{V}[\![F]\!] \overset{\tau}{\Longrightarrow}_{\mathsf{o}\pi} P$ *then there is* F' *such that* $F \Longrightarrow_{\mathsf{a}} F'$ *and* $P \;{}_{\pi}\!\gtrsim \mathcal{V}[\![F']\!]$ *;*
2. *If* $\mathcal{V}[\![F]\!] \overset{\ell}{\Longrightarrow}_{\mathsf{o}\pi} P$ *then there is* F' *such that* $F \overset{\ell}{\Longrightarrow}_{\mathsf{a}} F'$ *and* $P \approx_{\pi} \mathcal{V}[\![F']\!]$.

▶ **Corollary 8.** *For any configuration* F *and trace* s*, we have* $F \overset{s}{\Longrightarrow}_{\mathsf{a}}$ *iff* $\mathcal{V}[\![F]\!] \overset{s}{\Longrightarrow}_{\mathsf{o}\pi}$.

▶ Remark 9. We recall that, following Remark 1 on the usage of bound names, in Corollary 8 the bound names in s are fresh; thus they do not appear in F. (Similarly, in Theorems 5 and 7 for the bound names in ℓ).

For both results we first establish a correspondence result on strong transitions. See [21] for details.

▶ Remark 10. Corollary 8 relies on Theorems 4 and 7. The corollary talks about the opLTS of Iπ; however the theorems make use of the ordinary expansion relation $\lesssim_{\pi}$, that is defined on the ordinary LTS. Such uses of expansion can be replaced by expansion on the opLTS (defined as ordinary expansion but on the opLTS). For more details on this, see [21].

As a consequence of Corollary 8, trace equivalence is the same, on A-OGS configurations and on the encoding Iπ terms. Moreover, from Theorem 7 the same result holds under a bisimulation semantics. Further, since the LTS produced by A-OGS is deterministic, its trace semantics coincides with its bisimulation semantics. We can thus conclude as in Corollary 11. We recall that $\simeq_{\mathsf{o}\pi}$ and $\approx_{\mathsf{o}\pi}$ are, respectively, trace equivalence and bisimilarity between Iπ processes in the opLTS; similarly for $\simeq_{\mathsf{a}}$ and $\approx_{\mathsf{a}}$ between A-OGS terms.

▶ **Corollary 11.** *For any* F, F' *we have:* $F \simeq_{\mathsf{a}} F'$ *iff* $\mathcal{V}[\![F]\!] \simeq_{\mathsf{o}\pi} \mathcal{V}[\![F']\!]$ *iff* $F \approx_{\mathsf{a}} F'$ *iff* $\mathcal{V}[\![F]\!] \approx_{\mathsf{o}\pi} \mathcal{V}[\![F']\!]$.

Corollary 11 holds in particular when F is the initial configuration for a λ-term. That is, the equality induced on call-by-value λ-terms by their representation in A-OGS and in Iπ (under the opLTS) is the same, both employing traces and employing bisimulation to handle the observables for the two models.

$$\begin{array}{l|llll}
(P\tau) & \langle A \cdot [p \mapsto M], \gamma, \phi \rangle & \xrightarrow{\tau}_{\mathsf{c}} & \langle A \cdot [p \mapsto N], \gamma, \phi \rangle & \text{when } M \to_{\mathsf{v}} N \\
(PA) & \langle A \cdot [p \mapsto V], \gamma, \phi \rangle & \xrightarrow{\bar{p}(x)}_{\mathsf{c}} & \langle A, \gamma \cdot [x \mapsto V], \phi \uplus \{x\} \rangle & \\
(PQ) & \langle A \cdot [p \mapsto E[xV]], \gamma, \phi \rangle & \xrightarrow{\bar{x}(y,q)}_{\mathsf{c}} & \langle A, \gamma \cdot [y \mapsto V] \cdot [q \mapsto (E,p)], \phi \uplus \{y,q\} \rangle & \\
(OA) & \langle A, \gamma \cdot [p \mapsto (E,q)], \phi \rangle & \xrightarrow{p(x)}_{\mathsf{c}} & \langle A \cdot [q \mapsto E[x]], \gamma, \phi \uplus \{x\} \rangle & \\
(OQ) & \langle A, \gamma, \phi \rangle & \xrightarrow{x(y,p)}_{\mathsf{c}} & \langle A \cdot [p \mapsto Vy], \gamma, \phi \uplus \{y,p\} \rangle & \text{when } \gamma(x) = V \\
(IOQ) & \langle [? \mapsto M], \phi \rangle & \xrightarrow{(p)}_{\mathsf{c}} & \langle [p \mapsto M], \varepsilon, \phi \uplus \{p\} \rangle &
\end{array}$$

Figure 4 The LTS for the Concurrent OGS.

▶ **Corollary 12.** *For any λ-terms M, N, we have:*
$\langle M \rangle \mathrel{\overset{\frown}{-}}_{\mathsf{a}} \langle N \rangle$ *iff* $\langle M \rangle \approx_{\mathsf{a}} \langle N \rangle$ *iff* $\mathcal{V}[\![M]\!] \mathrel{\overset{\frown}{-}}_{\mathrm{o}\pi} \mathcal{V}[\![N]\!]$ *iff* $\mathcal{V}[\![M]\!] \approx_{\mathrm{o}\pi} \mathcal{V}[\![N]\!]$.

From Theorem 5 and Corollary 8, it also follows that F and $\mathcal{V}[\![F]\!]$ are weakly bisimilar, on the union of the respective LTSs.

7 Concurrent Operational Game Semantics

In this section, we explore another way to derive an exact correspondence between OGS and Iπ, by relaxing the Alternating LTS for OGS so to allow multiple terms in configurations to run concurrently. We refer to the resulting OGS as the *Concurrent OGS*, briefly C-OGS (we recall that A-OGS refers to the Alternating OGS of Section 4).

We introduce *running terms*, ranged over by A, B, as finite mappings from continuation names to λ-terms. A *concurrent configuration* is a triple $\langle A, \gamma, \phi \rangle$ of a running term A, an environment γ, and a set of names ϕ. Moreover, the domains of A and γ must be disjoint. We extend the definition of the polarity function, considering names in the domain of both A and γ as Player names.

Passive and active configurations can be seen as special case of C-OGS configurations with zero and one running term, respectively. For this reason we still use F, G to range over C-OGS configurations. Moreover we freely take A-OGS configurations to be C-OGS configurations, and conversely for C-OGS configurations with zero and one running term, omitting the obvious syntactic coercions. Both the running term and the environment may be empty.

We present the rules of C-OGS in Figure 4. Since there is no more distinction between passive and active configurations, a given configuration can perform both Player and Opponent actions. Notice that only Opponent can add a new term to the running term A. A *singleton* is a configuration F whose P-support has only one element (that is, in C-OGS, F is either of the form $\langle [p \mapsto M], \varepsilon, \phi \rangle$, or $\langle \varepsilon, [x \mapsto V], \phi \rangle$, or $\langle \varepsilon, [p \mapsto (E,q)], \phi \rangle$).

In this and in the following section F, G ranges over C-OGS configurations, as reminded by the index "c" in the symbols for LTS and behavioural equivalence with which F, G appear (e.g., $\mathrel{\overset{\frown}{-}}_{\mathsf{c}}$).

7.1 Comparison between C-OGS and Iπ

The encoding of C-OGS into Iπ is a simple adaptation of that for A-OGS. We only have to consider the new or modified syntactic elements of C-OGS, namely running terms and configurations; the encoding remains otherwise the same. The encoding of running term is:

$$\mathcal{V}[\![[p \mapsto M] \cdot A]\!] \stackrel{\text{def}}{=} \mathcal{V}[\![M]\!]\langle p \rangle \mid \mathcal{V}[\![A]\!] \qquad\qquad \mathcal{V}[\![\varepsilon]\!] \stackrel{\text{def}}{=} 0$$

The encoding of configurations is then defined as: $\mathcal{V}[\![\langle A,\gamma,\phi\rangle]\!] \stackrel{\text{def}}{=} \mathcal{V}[\![A]\!] \mid \mathcal{V}[\![\gamma]\!]$.

The results about operational correspondence between C-OGS and Iπ are as those between A-OGS and Iπ under the opLTS.

▶ **Theorem 13.**
1. *If* $F \Longrightarrow_{\mathsf{c}} F'$ *then* $\mathcal{V}[\![F]\!] \Longrightarrow_{\pi\,\pi}\gtrsim \mathcal{V}[\![F']\!]$*;*
2. *if* $F \stackrel{\ell}{\Longrightarrow}_{\mathsf{c}} F'$ *then* $\mathcal{V}[\![F]\!] \stackrel{\ell}{\Longrightarrow}_{\pi}\approx_{\pi} \mathcal{V}[\![F']\!]$*;*
3. *the converse of (1), i.e. if* $\mathcal{V}[\![F]\!] \Longrightarrow_{\pi} P$ *then there is* F' *such that* $F \Longrightarrow_{\mathsf{c}} F'$ *and* $P \;{}_{\pi}\!\gtrsim \mathcal{V}[\![F']\!]$*.*
4. *the converse of (2), i.e. if* $\mathcal{V}[\![F]\!] \stackrel{\ell}{\Longrightarrow}_{\pi} P$ *then there is* F' *such that* $F \stackrel{\ell}{\Longrightarrow}_{\mathsf{c}} F'$ *and* $P \approx_{\pi} \mathcal{V}[\![F']\!]$*.*

▶ **Corollary 14.** *For any* C-OGS *configuration* F *and trace* s*, we have* $F \stackrel{s}{\Longrightarrow}_{\mathsf{c}}$ *iff* $\mathcal{V}[\![F]\!] \stackrel{s}{\Longrightarrow}_{\pi}$*.*

From Corollary 14 and Theorem 13, we derive:

▶ **Lemma 15.** *For any* F, F' *we have:*
1. $F_1 \simeq_{\mathsf{c}} F_2$ *iff* $\mathcal{V}[\![F_1]\!] \simeq_{\pi} \mathcal{V}[\![F_2]\!]$*;*
2. $F_1 \approx_{\mathsf{c}} F_2$ *iff* $\mathcal{V}[\![F_1]\!] \approx_{\pi} \mathcal{V}[\![F_2]\!]$*.*

To derive the full analogous of Corollary 12, we now show that, on the Iπ representation of λ-terms, trace equivalence is the same as bisimilarity. This result needs a little care: it is known that on deterministic LTSs bisimilarity coincides with trace equivalence. However, the behaviour of the Iπ representation of a C-OGS configuration need not be deterministic, because there could be multiple silent transitions as well as multiple output transitions (for instance, in C-OGS rule OQ may be applicable to different terms).

▶ **Lemma 16.** *For any* M, N *we have:* $\mathcal{V}[\![M]\!] \simeq_{\pi} \mathcal{V}[\![N]\!]$ *iff* $\mathcal{V}[\![M]\!] \approx_{\pi} \mathcal{V}[\![N]\!]$*.*

The proof uses the "bisimulation up-to context and up-to $({}_{\pi}\!\gtrsim, \approx_{\pi})$" technique. We can finally combine Lemmas 16 and 15 to derive that the C-OGS and Iπ semantics of λ-calculus coincide, both for traces and for bisimilarity.

▶ **Corollary 17.** *For all* M, N *we have:* $\langle M\rangle \simeq_{\mathsf{c}} \langle N\rangle$ *iff* $\langle M\rangle \approx_{\mathsf{c}} \langle N\rangle$ *iff* $\mathcal{V}[\![M]\!] \simeq_{\pi} \mathcal{V}[\![N]\!]$ *iff* $\mathcal{V}[\![M]\!] \approx_{\pi} \mathcal{V}[\![N]\!]$*.*

More details on proofs may be found in [21].

7.2 Tensor Product

We now introduce a way of combining configurations, which corresponds to the notion of tensor product of arenas and strategies in (denotational) game semantics.

▶ **Definition 18.** *Two concurrent configurations* F, G *are said to be* compatible *if their polarity functions* $\mathrm{pol}_F, \mathrm{pol}_G$ *are compatible – that is, for all* $a \in \mathrm{dom}(\mathrm{pol}_F) \cap \mathrm{dom}(\mathrm{pol}_G)$*, we have* $\mathrm{pol}_F(a) = \mathrm{pol}_G(a)$*.*

▶ **Definition 19.** *For compatible configurations* $F = \langle A,\gamma,\phi\rangle$ *and* $G = \langle B,\delta,\phi'\rangle$*, the* tensor product $F \otimes G$ *is defined as* $F \otimes G \triangleq \langle A\cdot B, \gamma\cdot\delta, \phi\cup\phi'\rangle$

The polarity function of $F\otimes G$ is then equal to $\mathrm{pol}_F \cup \mathrm{pol}_G$, and $\mathcal{V}[\![F_1 \otimes F_2]\!] \equiv \mathcal{V}[\![F_1]\!] \mid \mathcal{V}[\![F_2]\!]$, where $\equiv$ is the standard structural congruence of π-calculi. In the following, we write $\mathtt{inter}(s_1, s_2)$ for the set of traces obtained from an interleaving of the elements in the sequences s_1 and s_2.

▶ **Lemma 20.** *Suppose F_1, F_2 are compatible concurrent configurations. The set of traces generated by $F_1 \otimes F_2$ is the union of the sets of interleaving* $\texttt{inter}(s_1, s_2)$*, for $F_1 \overset{s_1}{\Longrightarrow}_{\mathsf{c}}$ and $F_2 \overset{s_2}{\Longrightarrow}_{\mathsf{c}}$.*

The tensor product of A-OGS configurations is defined similarly, with the additional hypothesis that at most one of the two configurations is active, in order for their tensor product to be a valid A-OGS configuration. The details can be found in [21].

7.3 Up-to techniques for games

We introduce up-to techniques for C-OGS, which allow, in bisimulation proofs, to split two C-OGS configurations into separate components and then to reason separately on these. These up-to techniques are directly imported from Iπ. Abstract settings for up-to techniques have been developed, see [36, 37]; but we cannot derive the OGS techniques from them because these settings are specific to first-order LTS (i.e., CCS-like, without binders within actions).

The new techniques are then used to prove that C-OGS and A-OGS yield the same semantics on λ-terms; a further application is in Section 7.5, discussing eager normal-form bisimilarity.

A relation $\mathcal{R}$ on configuration is *well-formed* if it relates configurations with the same polarity function. Below, all relations on configurations are meant to be well formed. Given a well-formed relation $\mathcal{R}$ we write:

- $\mathcal{R}^{|}$ for the relation $\{(F_1, F_2) \ : \ \exists\, G \text{ s.t. } F_i = F_i' \otimes G \ (i = 1, 2) \text{ and } F_1' \,\mathcal{R}\, F_2'\}$.
- $\mathcal{R}^{|\star}$ for the reflexive and transitive closure of $\mathcal{R}^{|}$. Thus from $F_1 \,\mathcal{R}\, G_1$ and $F_2 \,\mathcal{R}\, G_2$ we obtain $(F_1 \otimes F_2) \,\mathcal{R}^{|\star}\, (G_1 \otimes F_2) \,\mathcal{R}^{|\star}\, G_1 \otimes G_2$.
- $\Rightarrow_{\mathsf{c}} \mathcal{R}^{|\star} {}_{\mathsf{c}}\!\Leftarrow$ for the closure of $\mathcal{R}^{|\star}$ under reductions. That is, $F_1 \Rightarrow_{\mathsf{c}} \mathcal{R}^{|\star} {}_{\mathsf{c}}\!\Leftarrow F_2$ holds if there are F_i', $i = 1, 2$ with $F_i \Longrightarrow_{\mathsf{c}} F_i'$ and $F_1' \,\mathcal{R}^{|\star}\, F_2'$. (As $\Longrightarrow_{\mathsf{c}}$ is reflexive, we may have $F_i = F_i'$.)

▶ **Definition 21.** *A relation $\mathcal{R}$ on configurations is a* bisimulation up-to reduction and composition *if whenever $F_1 \,\mathcal{R}\, F_2$:*

1. *if $F_1 \xrightarrow{\mu}_{\mathsf{c}} F_1'$ then there is F_2' such that $F_2 \overset{\widehat{\mu}}{\Longrightarrow}_{\mathsf{c}} F_2'$ and $F_1' \Rightarrow_{\mathsf{c}} \mathcal{R}^{|\star} {}_{\mathsf{c}}\!\Leftarrow F_2'$;*
2. *the converse, on the transitions from F_2.*

A variant of the technique in Definition 21, where the bisimulation game is played only on visible actions at the price of being defined on singleton configurations is presented in [21] and used in Section 7.5 to lift any eager normal form bisimulation to an OGS "bisimulation up-to".

▶ **Theorem 22.** *If $\mathcal{R}$ is bisimulation up-to reduction and composition then $\mathcal{R} \subseteq \approx_{\mathsf{c}}$.*

The theorem is proved by showing that the Iπ image of $\mathcal{R}$ is a bisimulation up-to context and up-to $({}_{\pi}\!\gtrsim, \approx_\pi)$, and appealing to Lemma 15(2). See [21] for details.

▶ Remark 23. Results such as Corollary 17 and Theorem 22 might suggest that the equality between two configurations implies the equality of all their singleton components. That is, if $F \simeq_{\mathsf{c}} G$, with $[p \mapsto M]$ part of F and $[p \mapsto N]$ part of G, then also $\langle [p \mapsto M] \rangle \simeq_{\mathsf{c}} \langle [p \mapsto N] \rangle$. A counterexample is given by the configurations

$$\begin{aligned} F_1 &\overset{\text{def}}{=} \langle [p_1 \mapsto M] \cdot [p_2 \mapsto \Omega] \rangle \qquad \text{where } M \overset{\text{def}}{=} (\lambda z.\, \Omega)(x \lambda y.\, \Omega) \\ F_2 &\overset{\text{def}}{=} \langle [p_1 \mapsto \Omega] \cdot [p_2 \mapsto M] \rangle \end{aligned}$$

Intuitively the reason why $F_1 \doteq_{\mathsf{c}} F_2$ is that M can produce an output (along the variable x), but an observer will never obtain access to the name at which M is located (p_1 or p_2). That is, the term M can interrogate x, but it will never answer, neither at p_1 nor at p_2.

7.4 Relationship between Concurrent and Alternating OGS

In Section 6 we have proved that the trace-based and bisimulation-based semantics produced by A-OGS and by Iπ under the opLTS coincide. In Section 7.1 we have obtained the same result for C-OGS and Iπ under the ordinary LTS. In this section, we develop these results so to conclude that all such equivalences for λ-terms actually coincide. In other words, the equivalence induced on λ-terms by their representations in OGS and Iπ is the same, regardless of whether we adopt the alternating or concurrent flavour for OGS, the opLTS or the ordinary LTS in Iπ, a trace or a bisimulation semantics. For this, in one direction, we show that the trace semantics induced by C-OGS implies that induced by A-OGS. In the opposite direction, we lift a bisimulation over the alternating LTS on singleton configurations into a bisimulation up-to composition over the concurrent LTS.

▶ **Lemma 24.** *If F_1, F_2 are* A-OGS *singleton configurations and $F_1 \approx_{\mathsf{a}} F_2$, then also $F_1 \approx_{\mathsf{c}} F_2$.*

Details may be found in [21].

▶ **Corollary 25.** *For any λ-terms M, N, the following statements are the same: $\langle M\rangle \doteq_{\mathsf{a}} \langle N\rangle$; $\langle M\rangle \approx_{\mathsf{a}} \langle N\rangle$; $\langle M\rangle \doteq_{\mathsf{c}} \langle N\rangle$; $\langle M\rangle \approx_{\mathsf{c}} \langle N\rangle$; $\mathcal{V}[\![M]\!] \doteq_{\mathsf{o}\pi} \mathcal{V}[\![N]\!]$; $\mathcal{V}[\![M]\!] \approx_{\mathsf{o}\pi} \mathcal{V}[\![N]\!]$; $\mathcal{V}[\![M]\!] \doteq_{\pi} \mathcal{V}[\![N]\!]$; $\mathcal{V}[\![M]\!] \approx_{\pi} \mathcal{V}[\![N]\!]$.*

7.5 Eager Normal Form Bisimulations

We recall Lassen's *eager normal-form (enf) bisimilarity* [29].

▶ **Definition 26.** *An* enf-bisimulation *is a triple of relation on terms $\mathcal{R}_{\mathcal{M}}$, values $\mathcal{R}_{\mathcal{V}}$, and evaluation contexts $\mathcal{R}_{\mathcal{K}}$ that satisfies:*

- *$M_1 \; \mathcal{R}_{\mathcal{M}} \; M_2$ if either:*
  - *both M_1, M_2 diverge;*
  - *$M_1 \Downarrow E_1[xV_1]$ and $M_2 \Downarrow E_2[xV_2]$ for some x, values V_1, V_2, and evaluation contexts E_1, E_2 with $V_1 \; \mathcal{R}_{\mathcal{V}} \; V_2$ and $K_1 \; \mathcal{R}_{\mathcal{K}} \; K_2$;*
  - *$M_1 \Downarrow V_1$ and $M_2 \Downarrow V_2$ for some values V_1, V_2 with $V_1 \; \mathcal{R}_{\mathcal{V}} \; V_2$.*
- *$V_1 \; \mathcal{R}_{\mathcal{V}} \; V_2$ if $V_1 x \; \mathcal{R}_{\mathcal{M}} \; V_2 x$ for some fresh x;*
- *$K_1 \; \mathcal{R}_{\mathcal{V}} \; K_2$ if $K_1[x] \; \mathcal{R}_{\mathcal{M}} \; K_2[x]$ for some fresh x.*

The largest enf-bisimulation *is called* enf-bisimilarity*.*

From Corollary 25 and existing results in the π-calculus [11] we can immediately conclude that the semantics on λ-terms induced by OGS (Alternating or Concurrent) coincides with enf-bisimilarity (i.e., Lassen's trees).

In this section, we show a direct proof of the result, for C-OGS bisimilarity, as an example of application of the up-to composition technique for C-OGS.

Terms, Values, and Evaluations contexts can be directly lifted to singleton concurrent configurations, meaning that we can transform a relation on terms, values, and contexts $\mathcal{R}$ into a relation $\widehat{\mathcal{R}}$ on singleton concurrent configurations in the following way:

- If $(M_1, M_2) \in \mathcal{R}$ then $(\langle [p \mapsto M_1], \varepsilon, \phi\rangle, \langle [p \mapsto M_2], \varepsilon, \phi\rangle) \in \widehat{\mathcal{R}}$, with $\phi = \mathtt{fv}(M_1, M_2) \uplus \{p\}$
- If $(V_1, V_2) \in \mathcal{R}$ then $(\langle [x \mapsto V_1], \phi\rangle, \langle [x \mapsto V_2], \phi\rangle) \in \widehat{\mathcal{R}}$, with $\phi = \mathtt{fv}(V_1, V_2) \uplus \{x\}$;
- If $(E_1, E_2) \in \mathcal{R}$ then $(\langle [p \mapsto (E_1, q)], \phi\rangle, \langle [p \mapsto (E_2, q)], \phi\rangle) \in \widehat{\mathcal{R}}$, with $\phi = \mathtt{fv}(E_1, E_2) \uplus \{p, q\}$.

▶ **Theorem 27.** *Taking $(\mathcal{R}_\mathcal{M}, \mathcal{R}_\mathcal{V}, \mathcal{R}_\mathcal{K})$ an enf-bisimulation, then $(\widehat{\mathcal{R}_\mathcal{M}} \cup \widehat{\mathcal{R}_\mathcal{V}} \cup \widehat{\mathcal{R}_\mathcal{K}})$ is a singleton bisimulation up-to composition in* C-OGS *(as defined in [21]).*

Proof. Taking F_1, F_2 two singleton configurations s.t. $(F_1, F_2) \in \widehat{\mathcal{R}_\mathcal{M}}$, then we can write F_i as $\langle [p \mapsto M_i], \phi\rangle$ for $i = 1, 2$, such that $(M_1, M_2) \in \mathcal{R}$. suppose that $F_1 \Longrightarrow_{\mathsf{c}} \xrightarrow{\ell}_{\mathsf{c}} F'_1$, with ℓ a Player action. We only present the Player Question case, which is where "up-to composition" is useful. So writing ℓ as $\bar{x}(y, q)$, F'_1 can be written as $\langle [y \mapsto V_1] \cdot [q \mapsto (E_1, p)]\rangle$, so that $M_1 \rightarrow^*_{\mathrm{v}} E_1[xV_1]$.

As $(M_1, M_2) \in \mathcal{R}$, there are E_2, V_2 s.t. $M_2 \rightarrow^*_{\mathrm{v}} E_2[xV_2]$, $(V_1, V_2) \in \mathcal{R}$ and $(E_1, E_2) \in \mathcal{R}$. Hence $F_2 \Longrightarrow_{\mathsf{c}} \xrightarrow{\ell}_{\mathsf{c}} F'_2$ with $F'_2 = \langle [y \mapsto V_2] \cdot [q \mapsto (E_2, p)]\rangle$.

Finally, $(\langle [y \mapsto V_1]\rangle, \langle [y \mapsto V_2]\rangle) \in \widehat{\mathcal{R}}$ and $(\langle [q \mapsto (E_1, p)]\rangle, \langle [q \mapsto (E_2, p)]\rangle) \in \widehat{\mathcal{R}}$, so that $(F'_1, F'_2) \in \widehat{\mathcal{R}}^{|\star}$.

The case of passive singleton configurations is proved in a similar way. ◀

8 Compositionality of OGS via Iπ

We now present some compositionality results about C-OGS and A-OGS, that can be proved via the correspondence between OGS and Iπ.

Compositionality of OGS amounts to compute the set of traces generated by $\langle M\{V/x\}\rangle$ from the set of traces generated by $\langle M\rangle$ and $\langle [x \mapsto V]]\rangle$. This is the cornerstone of (denotational) game semantics, where the combination of $\langle M\rangle$ and $\langle [x \mapsto V]\rangle$ is represented via the so-called "parallel composition plus hiding". This notion of parallel composition of two processes P, Q plus hiding over a name x is directly expressible in Iπ as the process $\nu x(P \mid Q)$. Precisely, suppose F, G are two configurations that agree on their polarity functions, but on a name x; then we write

$$\nu x(F \mid G) \tag{*}$$

for the Iπ process $\nu x(\mathcal{V}[\![F]\!] \mid \mathcal{V}[\![G]\!])$, the *parallel composition plus hiding over x*. To define this operation directly at the level of OGS, we would have to generalize its LTS, allowing internal interactions over a name x used both in input and in output.

As the translation $\mathcal{V}$ from OGS configurations into Iπ validates the β_v rule, we can prove that the behaviour of $\langle M\{V/x\}\rangle$ (e.g., its set of traces) is the same as that of the parallel composition plus hiding over x of $\langle M\rangle$ and $\langle [x \mapsto V]\rangle)$. We use the notation $(*)$ above to express the following two results.

▶ **Theorem 28.** *For $\bowtie \in \{\approx_{\mathrm{o}\pi}, \approx_\pi, \simeq_{\mathrm{o}\pi}, \simeq_\pi\}$, we have*
$\mathcal{V}[\![\langle [p \mapsto M\{V/x\}], \gamma, \phi \cup \phi'\rangle]\!] \bowtie \nu x(\langle [p \mapsto M], \varepsilon, \phi \uplus \{x\}\rangle \mid \langle \gamma \cdot [x \mapsto V], \phi' \uplus \{x\}\rangle)$

▶ **Corollary 29.** *For any trace s:*
1. $\langle M\{V/x\}, p, \gamma, \phi \cup \phi'\rangle \overset{s}{\Longrightarrow}_{\mathsf{a}}$ *iff* $\nu x(\langle M, p, \varepsilon, \phi \uplus \{x\}\rangle \mid \langle \gamma \cdot [x \mapsto V], \phi' \uplus \{x\}\rangle) \overset{s}{\Longrightarrow}_{\mathrm{o}\pi}$
2. $\langle [p \mapsto M\{V/x\}], \gamma, \phi \cup \phi'\rangle \overset{s}{\Longrightarrow}_{\mathsf{c}}$ *iff* $\nu x(\langle [p \mapsto M], \varepsilon, \phi \uplus \{x\}\rangle \mid \langle \gamma \cdot [x \mapsto V], \phi' \uplus \{x\}\rangle) \overset{s}{\Longrightarrow}_\pi$

Other important properties that we can import in OGS from Iπ are the congruence properties for the A-OGS and C-OGS semantics. We report the result for $\simeq_{\mathsf{a}}$; the same result holds for $\approx_{\mathsf{a}}, \simeq_{\mathsf{c}}, \approx_{\mathsf{c}}$.

▶ **Theorem 30.** *If $\langle M\rangle \simeq_{\mathsf{a}} \langle N\rangle$ then for any Λ_{V} context C, $\langle C[M]\rangle \simeq_{\mathsf{a}} \langle C[N]\rangle$.*

The result is obtained from the congruence properties of Iπ, Corollary 25, and the compositionality of the encoding $\mathcal{V}$.

9 Related and Future Work

Analogies between game semantics and π-calculus, as semantic frameworks in which *names* are central, have been pointed out from the very beginning of game semantics. In the pioneering work [19], the authors obtain a translation of PCF terms into the π-calculus from a game model of PCF by representing *strategies* (the denotation of PCF terms in the game model) as processes of the π-calculus. The encoding bears similarities with Milner's, though they are syntactically rather different ("it is clear that the two are conceptually quite unrelated", [19]). The connection has been developed in various papers, e.g., [8, 14, 17, 18, 46]. Milner's encodings into the π-calculus have sometimes been a source of inspiration in the definition of the game semantics models (e.g., transporting the work [19], in call-by-name, onto call-by-value [18]). In [46], a typed variant of π-calculus, influenced by differential linear logic [13], is introduced as a metalanguage to represent game models. In [9], games are defined using algebraic operations on sets of traces, and used to prove type soundness of a simply-typed call-by-value λ-calculus with effects. Although the calculus of traces employed is not a π-calculus (e.g., being defined from operators and relations over trace sets rather than from syntactic process constructs), there are similarities, which would be interesting to investigate.

Usually in the above papers the source language is a form of λ-calculus, that is interpreted into game semantics, and the π-calculus (or dialects of it) is used to represent the resulting strategies and games. Another goal has been to shed light into typing disciplines for π-calculus processes, by transplanting conditions on strategies such as well-bracketing and innocence into appropriate typings for the π-calculus (see, e.g., [6, 45]).

The results in this paper (e.g., operational correspondence and transfer of techniques) are not derivable from the above works, where analogies between game semantics and π-calculus are rather used to better understand one of the two models (i.e., explaining game semantics in terms of process interactions, or enhancing type systems for processes following structures in game semantics). Indeed, in the present paper we have carried out a *direct comparison* between the two models (precisely OGS and Iπ). For this we have started from the (arguably natural) representations of the λ-calculus into OGS and Iπ (the latter being Milner's encodings). Our goal was understanding the relation between the behaviours of the terms in the two models, and transferring techniques and results between them.

Technically, our work builds on [10, 11], where a detailed analysis of the behaviour of Milner's call-by-value encoding is carried out using proof techniques for π-calculus based on unique-solution of equations. Various results in [10, 11] are essential to our own (the observation that Milner's encodings should be interpreted in Iπ rather than the full π-calculus is also from [10, 11]).

Bisimulations over OGS terms, and tensor products of configurations, were introduced in [30], in order to provide a framework to study compositionality properties of OGS. In our case, the compositionality result of OGS is derived from the correspondence with the π-calculus. In [39], a correspondence between an i/o typed asynchronous π-calculus and a computational λ-calculus with channel communication is established, using a common categorical model (a compact closed Freyd category). It would be interesting to see if our concurrent operational game model could be equipped with this categorical semantics.

Normal form (or *open*) bisimulations [29, 43], as game semantics, manipulate open terms, and sometimes make use of environments or stacks of evaluation contexts (see, e.g., the recent work [7], where a fully abstract normal-form bisimulation for a λ-calculus with store is obtained).

There are also works that build game models directly for the π-calculus, i.e., [12, 26, 27, 38] A correspondence between a synchronous π-calculus with session types and concurrent game semantics [44] is given in [8], relating games (represented as arenas) to session types, and strategies (defined as coincident event structures) to processes. We have exploited the full abstraction results between OGS and Iπ to transport a few up-to techniques for bisimulation from Iπ onto OGS. However, in Iπ, there are various other such techniques, even a theory of bisimulation enhancements. We would like to see which other techniques could be useful in OGS, possibly transporting the theory of enhancements itself.

References

1 Samson Abramsky. The lazy lambda calculus. In D. Turner, editor, *Research Topics in Functional Programming*, pages 65–116. Addison-Wesley, 1989.

2 Samson Abramsky. Game semantics for programming languages (abstract). In *Proceedings of the 22nd International Symposium on Mathematical Foundations of Computer Science*, MFCS '97, pages 3–4, Berlin, Heidelberg, 1997. Springer-Verlag.

3 Samson Abramsky, Kohei Honda, and Guy McCusker. A fully abstract game semantics for general references. In *Proceedings of the 13th Annual IEEE Symposium on Logic in Computer Science*, LICS '98, page 334, USA, 1998. IEEE Computer Society.

4 Samson Abramsky, Radha Jagadeesan, and Pasquale Malacaria. Full abstraction for pcf. *Inf. Comput.*, 163(2):409–470, December 2000. doi:10.1006/inco.2000.2930.

5 Samson Abramsky and Guy McCusker. Linearity, sharing and state: a fully abstract game semantics for Idealized Algol with active expressions. In *Algol-like languages*, pages 297–329. Springer, 1997.

6 Martin Berger, Kohei Honda, and Nobuko Yoshida. Sequentiality and the π-calculus. In *International Conference on Typed Lambda Calculi and Applications*, pages 29–45. Springer, 2001.

7 Dariusz Biernacki, Sergueï Lenglet, and Piotr Polesiuk. A complete normal-form bisimilarity for state. In *International Conference on Foundations of Software Science and Computation Structures*, pages 98–114. Springer, 2019.

8 Simon Castellan and Nobuko Yoshida. Two sides of the same coin: Session types and game semantics: A synchronous side and an asynchronous side. *Proc. ACM Program. Lang.*, January 2019. doi:10.1145/3290340.

9 Tim Disney and Cormac Flanagan. Game semantics for type soundness. In *Proceedings of the 2015 30th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, LICS '15, pages 104–114, USA, 2015. IEEE Computer Society. doi:10.1109/LICS.2015.20.

10 Adrien Durier. *Unique solution techniques for processes and functions.* PhD thesis, University of Lyon, France, 2020.

11 Adrien Durier, Daniel Hirschkoff, and Davide Sangiorgi. Eager functions as processes. In *33nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018.* IEEE Computer Society, 2018.

12 Clovis Eberhart, Tom Hirschowitz, and Thomas Seiller. An intensionally fully-abstract sheaf model for pi. In *6th Conference on Algebra and Coalgebra in Computer Science (CALCO 2015).* Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

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

14 Marcelo Fiore and Kohei Honda. Recursive types in games: Axiomatics and process representation. In *Proceedings of the 13th Annual IEEE Symposium on Logic in Computer Science*, LICS '98, page 345, USA, 1998. IEEE Computer Society.

15 Dan R. Ghica and Andrzej S. Murawski. Angelic semantics of fine-grained concurrency. In *International Conference on Foundations of Software Science and Computation Structures*, pages 211–225. Springer, 2004.

16 Dan R. Ghica and Nikos Tzevelekos. A system-level game semantics. *Electron. Notes Theor. Comput. Sci.*, 286:191–211, September 2012. `doi:10.1016/j.entcs.2012.08.013`.

17 Kohei Honda. Processes and games. *Electron. Notes Theor. Comput. Sci.*, 71:40–69, 2002. `doi:10.1016/S1571-0661(05)82528-9`.

18 Kohei Honda and Nobuko Yoshida. Game-theoretic analysis of call-by-value computation. *Theor. Comput. Sci.*, 221(1–2):393–456, June 1999. `doi:10.1016/S0304-3975(99)00039-0`.

19 J. M. E. Hyland and C.-H. L. Ong. Pi-calculus, dialogue games and full abstraction PCF. In *Proceedings of the Seventh International Conference on Functional Programming Languages and Computer Architecture*, FPCA '95, pages 96–107, New York, NY, USA, 1995. Association for Computing Machinery. `doi:10.1145/224164.224189`.

20 J.M.E. Hyland and C.-H. L. Ong. On full abstraction for PCF. *Inf. Comput.*, 163(2):285–408, December 2000. `doi:10.1006/inco.2000.2917`.

21 Guilhem Jaber and Davide Sangiorgi. Games, mobile processes, and functions (long version), 2021. URL: `https://hal.archives-ouvertes.fr/hal-03407123`.

22 Guilhem Jaber and Nikos Tzevelekos. Trace semantics for polymorphic references. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '16, pages 585–594, New York, NY, USA, 2016. Association for Computing Machinery. `doi:10.1145/2933575.2934509`.

23 Radha Jagadeesan, Corin Pitcher, and James Riely. Open bisimulation for aspects. In *Transactions on Aspect-Oriented Software Development V*, pages 72–132. Springer, 2009.

24 Alan Jeffrey and Julian Rathke. Java jr: Fully abstract trace semantics for a core java language. In *European symposium on programming*, pages 423–438. Springer, 2005.

25 James Laird. Full abstraction for functional languages with control. In *Proceedings of the 12th Annual IEEE Symposium on Logic in Computer Science*, LICS '97, page 58, USA, 1997. IEEE Computer Society.

26 James Laird. A game semantics of the asynchronous π-calculus. In *International Conference on Concurrency Theory*, pages 51–65. Springer, 2005.

27 James Laird. Game semantics for higher-order concurrency. In *Proceedings of the 26th International Conference on Foundations of Software Technology and Theoretical Computer Science*, FSTTCS'06, pages 417–428, Berlin, Heidelberg, 2006. Springer-Verlag. `doi:10.1007/11944836.38`.

28 James Laird. A fully abstract trace semantics for general references. In *Proceedings of the 34th International Conference on Automata, Languages and Programming*, ICALP'07, pages 667–679, Berlin, Heidelberg, 2007. Springer-Verlag.

29 Søren B. Lassen. Eager normal form bisimulation. In *20th IEEE Symposium on Logic in Computer Science (LICS 2005), 26-29 June 2005, Chicago, IL, USA, Proceedings*, pages 345–354, 2005. `doi:10.1109/LICS.2005.15`.

30 Paul Blain Levy and Sam Staton. Transition systems over games. In *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, CSL-LICS '14, New York, NY, USA, 2014. Association for Computing Machinery. `doi:10.1145/2603088.2603150`.

31 R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.

32 R. Milner. The polyadic π-calculus: a tutorial. Technical Report ECS–LFCS–91–180, LFCS, 1991. Also in *Logic and Algebra of Specification*, ed. F.L. Bauer, W. Brauer and H. Schwichtenberg, Springer Verlag, 1993.

33 R. Milner. Functions as processes. *MSCS*, 2(2):119–141, 1992.

34 B. Pierce and D. Sangiorgi. Typing and subtyping for mobile processes. *MSCS*, 6(5):409–454, 1996.

35 G.D. Plotkin. Call by name, call by value and the λ-calculus. *TCS*, 1:125–159, 1975.

36 Damien Pous and Davide Sangiorgi. Enhancements of the bisimulation proof method. In Davide Sangiorgi and Jan Rutten, editors, *Advanced Topics in Bisimulation and Coinduction*. Cambridge University Press, 2012.

37 Damien Pous and Davide Sangiorgi. Bisimulation and coinduction enhancements: A historical perspective. *Formal Asp. Comput.*, 31(6):733–749, 2019. `doi:10.1007/s00165-019-00497-w`.

38 Ken Sakayori and Takeshi Tsukada. A truly concurrent game model of the asynchronous π-calculus. In *International Conference on Foundations of Software Science and Computation Structures*, pages 389–406. Springer, 2017.

39 Ken Sakayori and Takeshi Tsukada. A categorical model of an **i**/**o**-typed π-calculus. In *European Symposium on Programming*, pages 640–667. Springer, 2019.

40 D. Sangiorgi. Locality and non-interleaving semantics in calculi for mobile processes. *TCS*, 155:39–83, 1996.

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

42 D. Sangiorgi. Lazy functions and mobile processes. In G. Plotkin, C. Stirling, and M. Tofte, editors, *Proof, Language and Interaction: Essays in Honour of Robin Milner*. MIT Press, 2000.

43 Kristian Støvring and Soren B. Lassen. A complete, co-inductive syntactic theory of sequential control and state. In *Proceedings of the 34th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '07, pages 161–172, New York, NY, USA, 2007. Association for Computing Machinery. `doi:10.1145/1190216.1190244`.

44 Glynn Winskel, Silvain Rideau, Pierre Clairambault, and Simon Castellan. Games and strategies as event structures. *Logical Methods in Computer Science*, 13, 2017.

45 Nobuko Yoshida, Martin Berger, and Kohei Honda. Strong normalisation in the π-Calculus. In *16th Annual IEEE Symposium on Logic in Computer Science (LICS'01)*, pages 311–322. IEEE Computer Society, 2001.

46 Nobuko Yoshida, Simon Castellan, and Léo Stefanesco. Game semantics: Easy as pi. *arXiv preprint*, 2020. `arXiv:2011.05248`.