Deciding Contextual Equivalence of ν-Calculus with Effectful Contexts

Daniel Hirschkoff[1], Guilhem Jaber[2(✉)], and Enguerrand Prebet[1]

[1] Université de Lyon, ENS de Lyon, UCB Lyon 1, CNRS, INRIA, LIP, France
[2] Nantes Université, LS2N, Inria, France
guilhem.jaber@univ-nantes.fr

Abstract. We prove decidability for contextual equivalence of the $\lambda\mu\nu$-calculus, that is the simply-typed call-by-value $\lambda\mu$-calculus equipped with booleans and fresh name creation, with contexts taken in $\lambda\mu_{\mathtt{ref}}$, that is $\lambda\mu\nu$-calculus extended with higher-order references.
The proof exploits a labelled transition system capturing the interactions between $\lambda\mu\nu$ programs and $\lambda\mu_{\mathtt{ref}}$ contexts. The induced bisimulation equivalence is characterized as equality of certain trees, inspired by the work of Lassen. Since these trees are computable and finite, decidability follows. Bisimulation coincides also with trace equivalence, which in turn coincides with contextual equivalence .

1 Introduction

Dynamic allocation is central to many programming constructions. Many languages provide built-in support for dynamically-allocated resources, for example, objects in Java or references in ML. The creation of these resources is *local*, meaning that resources can be accessed only within their scope. They can also be passed around via function applications, in which case their scope is not static but evolves dynamically. When building semantics for such languages, one represents dynamic allocation as the creation of fresh locations, that can be seen as atoms or names.

In this paper, we study a paradigmatic language with dynamic allocation, namely the ν-calculus, a simply-typed call-by-value λ-calculus with fresh atom creation and equality test of atoms, as introduced by Pitts and Stark in [24]. For instance, the ν-calculus program $\mathtt{new}\ n\ \mathtt{in}\ \lambda x.(x = n)$ allocates a new atom n, receives an atom x and returns the result of the comparison between x and n.

A central question while studying this language is to determine when two programs can be considered to be equivalent. The most studied approach to express behavioral equivalence between programs is contextual equivalence. Intuitively, two programs are deemed equivalent if and only if whenever they are run as part of an enclosing program called the *context*, it is not possible to distinguish one from the other. For instance, because the context has no way to guess the atom n, we expect the program above to be equivalent to $\lambda x.\mathtt{false}$.

Reasoning on contextual equivalence for the ν-calculus has shown to be challenging, due to the interplay between the higher-order control flow and the scope

O. Kupferman and P. Sobocinski (Eds.): FoSSaCS 2023, LNCS 13992, pp. 24–45, 2023.
https://doi.org/10.1007/978-3-031-30829-1_2

extrusion of atoms. A variety of frameworks has been introduced to do so, based on logical relations [24], environmental bisimulations [5], and game semantics [1].

However, the question of whether this equivalence is *decidable* remains open since the introduction of this language 30 years ago.

In this paper, we address this question by working in an asymmetric setting, giving contexts more discriminating power than just the mere creation of atoms. Indeed, contextual equivalence depends on two languages: the language for programs, and the language for contexts interacting with these programs. We take contexts in the $\lambda\mu_{\mathtt{ref}}$-calculus, an extension of the ν-calculus with both higher-order references and continuations. In this setting, atoms are simply references where only the unit value can be stored. Contextual equivalence is then coarser than for the symmetric setting when the contexts are also taken in the ν-calculus. For example, one of the standard examples of equivalence of the literature

$$\mathtt{new}\ n\ \mathtt{in}\ \mathtt{new}\ n'\ \mathtt{in}\ \lambda f.(f\ n = f\ n') \ \simeq_{ctx}\ \lambda f.\mathtt{true}$$

is not an equivalence anymore, since a $\lambda\mu_{\mathtt{ref}}$ context can provide a function that stores its argument in a reference and use it to discriminate these programs.

The main result we establish in this paper is the decidability of contextual equivalence for terms of ν-calculus with contexts in the $\lambda\mu_{\mathtt{ref}}$-calculus. More generally, we establish this result for terms of the $\lambda\mu\nu$-calculus, which corresponds to terms of the $\lambda\mu_{\mathtt{ref}}$-calculus that only use references storing the unit value.

To establish this result, we provide a Böhm-like tree representation [6,3] for the terms of the $\lambda\mu\nu$-calculus. Being in call-by-value, equality of such trees coincides with Lassen's eager normal form bisimulations [16]. Moreover, since programs in the $\lambda\mu\nu$-calculus are terminating, these trees, which we call *Lassen trees*, are finite. It is thus straightforward to check their equality. Then, we prove that Lassen trees equality is fully-abstract, that is it coincides with contextual equivalence with contexts in the $\lambda\mu_{\mathtt{ref}}$-calculus.

Proving this full-abstraction result is done through the introduction of an *operational game semantics* (OGS) for $\lambda\mu_{\mathtt{ref}}$ by defining a Labelled Transition System (LTS) that distinguishes between internal operations, Proponent moves (originating in the program) and Opponent moves (originating in the context). Trace equivalence based on these labelled transitions is shown to coincide with the contextual equivalence of $\lambda\mu_{\mathtt{ref}}$.

The OGS also gives rise to a notion of *bipartite bisimulation*, describing a game between Proponent (the program in $\lambda\mu_{\mathtt{ref}}$) and Opponent (a context in $\lambda\mu_{\mathtt{ref}}$). Proponent reduces the program until it reaches a normal form, that triggers an interaction with the context. Along the game, knowledge is accumulated in configurations. When it is Opponent's turn to play, it chooses between answering a previous function call from Proponent, or generating a new function call, to which Proponent shall answer. Among this knowledge, we accumulate the atoms that have been disclosed by the two players, so that Opponent cannot use an atom private to Proponent.

The OGS LTS generates infinite trees since Opponent can interrogate an arbitrary number of times each value provided by Proponent. The Lassen trees used to decide contextual equivalence are generated using a *linearized* variant of the OGS LTS, called the *Prime Operational Game Semantics* (POGS) LTS. The POGS LTS enforces that Opponent interrogates only once each value provided by Proponent. For this linearization to be sound, one has to guess the disclosed status of atoms as soon as they are created. This can be illustrated by considering the following example of inequivalence

$$\texttt{new}\ n\ \texttt{in}\ \lambda x.n \not\approx_{ctx} \lambda x.\texttt{new}\ n\ \texttt{in}\ n.$$

Opponent must be able to interrogate at least twice each of these two programs to discriminate them. The first program would then return the same atom at each call, while the second program would return two different atoms. The Lassen tree of the first program would declare n to be disclosed when giving back the control to Opponent by providing the λ-abstraction, but this could not be matched by the second program, since n would not exist yet at that point of the interaction.

The main technical challenge at this point is to prove that this forecasting of the disclosure process is sound and complete. This is done by proving that the bipartite bisimilarities defined over the OGS LTS and the POGS LTS coincide. One direction is proven by lifting POGS bisimulations into OGS bisimulations via an *up-to* technique. The other direction is done by introducing a new *limit* construction of the disclosed set of atoms appearing in the OGS bisimulations, to transform it into a POGS bisimulation.

Paper outline. After introducing the $\lambda\mu_{\texttt{ref}}$-calculus and the $\lambda\mu\nu$-calculus in Section 2, we define the LTS for the OGS in Section 3. The induced trace equivalence coincides with contextual equivalence. We then move to Lassen trees in Section 4, and show that they yield an equivalence that coincides with bipartite bisimilarity in the OGS in Section 5. We discuss related work in Section 6, and present concluding remarks in Section 7. For lack of space, several technical developments are given in [9].

2 The $\lambda\mu_{\texttt{ref}}$-calculus and the $\lambda\mu\nu$-calculus

The syntax of the $\lambda\mu_{\texttt{ref}}$-calculus is given by the following grammar:

$$\begin{array}{lll}
\text{Values} & \texttt{V},\texttt{W} & \triangleq x \mid () \mid \lambda x.\texttt{M} \mid \texttt{true} \mid \texttt{false} \mid \ell \\
\text{Terms} & \texttt{M},\texttt{N} & \triangleq \texttt{V} \mid \texttt{let}\ x = \texttt{M}\ \texttt{in}\ \texttt{N} \mid \texttt{V}\texttt{W} \mid \texttt{if}\ \texttt{V}\ \texttt{then}\ \texttt{N}_1\ \texttt{else}\ \texttt{N}_2 \\
 & & \mid \texttt{V} = \texttt{W} \mid \texttt{new}\ x = \texttt{V}\ \texttt{in}\ \texttt{M} \mid \texttt{V} := \texttt{W} \mid !\texttt{V} \mid \mu\texttt{c}.\texttt{M} \mid [\texttt{c}]\texttt{M} \\
\text{Contexts} & \texttt{C},\texttt{C}' & \triangleq \bullet \mid [\texttt{c}]\texttt{C} \mid \texttt{let}\ x = \texttt{C}\ \texttt{in}\ \texttt{M} \mid \texttt{let}\ x = \texttt{M}\ \texttt{in}\ \texttt{C} \mid \lambda x.\texttt{C} \mid \mu\texttt{c}.\texttt{C} \\
 & & \mid \texttt{if}\ \texttt{V}\ \texttt{then}\ \texttt{C}\ \texttt{else}\ \texttt{M} \mid \texttt{if}\ \texttt{V}\ \texttt{then}\ \texttt{M}\ \texttt{else}\ \texttt{C} \mid \texttt{new}\ x = \texttt{V}\ \texttt{in}\ \texttt{C} \\
\text{Evaluation Contexts} & \texttt{E},\texttt{E}' & \triangleq [\texttt{c}]\bullet \mid \texttt{E}[\texttt{let}\ x = \bullet\ \texttt{in}\ \texttt{M}] \\
\text{Types} & \sigma,\tau & \triangleq \texttt{Unit} \mid \texttt{Bool} \mid \sigma \to \tau \mid \mathbf{ref}_\sigma \mid \bot
\end{array}$$

with $x \in$ Vars (variables), $\texttt{c} \in$ Covars (continution variables), $\ell \in$ Locs (locations). We write $\mathsf{supp}(\texttt{M})$ for the set of locations appearing in $\texttt{M}$, and $\mathbf{FV}(\texttt{M})$ for

$$\frac{\Gamma(x) = \sigma}{\Sigma;\Gamma \vdash x : \sigma} \qquad \frac{\Gamma(\mathtt{c}) = \neg\sigma}{\Sigma;\Gamma \vdash \mathtt{c} : \neg\sigma} \qquad \frac{\Sigma(\ell) = \mathbf{ref}_\sigma}{\Sigma;\Gamma \vdash \ell : \mathbf{ref}_\sigma} \qquad \frac{}{\Sigma;\Gamma \vdash () : \mathtt{Unit}}$$

$$\frac{\mathtt{b} \in \{\mathtt{true}, \mathtt{false}\}}{\Sigma;\Gamma \vdash \mathtt{b} : \mathtt{Bool}} \qquad \frac{\Sigma;\Gamma, x : \sigma \vdash \mathtt{M} : \tau}{\Sigma;\Gamma \vdash \lambda x.\mathtt{M} : \sigma \to \tau} \qquad \frac{\Sigma;\Gamma \vdash \mathtt{V} : \sigma \to \tau \quad \Sigma;\Gamma \vdash \mathtt{W} : \sigma}{\Sigma;\Gamma \vdash \mathtt{VW} : \tau}$$

$$\frac{\Sigma;\Gamma \vdash \mathtt{N} : \sigma \qquad \Sigma;\Gamma, x : \sigma \vdash \mathtt{M} : \tau}{\Sigma;\Gamma \vdash \mathtt{let}\ x = \mathtt{N}\ \mathtt{in}\ \mathtt{M} : \tau} \qquad \frac{\Sigma;\Gamma \vdash \mathtt{V} : \mathtt{Bool} \qquad \Sigma;\Gamma \vdash \mathtt{M}_1 : \sigma \qquad \Sigma;\Gamma \vdash \mathtt{M}_2 : \sigma}{\Sigma;\Gamma \vdash \mathtt{if}\ \mathtt{V}\ \mathtt{then}\ \mathtt{M}_1\ \mathtt{else}\ \mathtt{M}_2 : \sigma}$$

$$\frac{\Sigma;\Gamma \vdash \mathtt{V} : \tau \qquad \Sigma;\Gamma, x : \mathbf{ref}_\tau \vdash \mathtt{M} : \sigma}{\Sigma;\Gamma \vdash \mathtt{new}\ x = \mathtt{V}\ \mathtt{in}\ \mathtt{M} : \sigma} \qquad \frac{\Sigma;\Gamma \vdash \mathtt{V} : \mathbf{ref}_\sigma \qquad \Sigma;\Gamma \vdash \mathtt{W} : \sigma}{\Sigma;\Gamma \vdash \mathtt{V} := \mathtt{W} : \mathtt{Unit}}$$

$$\frac{\Sigma;\Gamma \vdash \mathtt{V} : \mathbf{ref}_\sigma}{\Sigma;\Gamma \vdash !\mathtt{V} : \sigma} \qquad \frac{\Sigma;\Gamma \vdash \mathtt{V} : \mathbf{ref}_\sigma \qquad \Sigma;\Gamma \vdash \mathtt{W} : \mathbf{ref}_\sigma}{\Sigma;\Gamma \vdash \mathtt{V} = \mathtt{W} : \mathtt{Bool}} \qquad \frac{\Sigma;\Gamma, \mathtt{c} : \neg\sigma \vdash \mathtt{M} : \bot}{\Sigma;\Gamma \vdash \mu\mathtt{c}.\mathtt{M} : \sigma}$$

$$\frac{\Sigma;\Gamma \vdash \mathtt{M} : \sigma \qquad \Gamma(\mathtt{c}) = \neg\sigma}{\Sigma;\Gamma \vdash [\mathtt{c}]\mathtt{M} : \bot} \qquad \frac{\Gamma(\mathtt{c}) = \neg\sigma}{\Sigma;\Gamma \vdash [\mathtt{c}]\bullet : \neg\sigma} \qquad \frac{\Sigma;\Gamma, x : \sigma \vdash \mathtt{M} : \tau \qquad \Sigma;\Gamma \vdash \mathtt{E} : \neg\tau}{\Sigma;\Gamma \vdash \mathtt{E}[\mathtt{let}\ x = \bullet\ \mathtt{in}\ \mathtt{M}] : \neg\sigma}$$

Fig. 1. $\lambda\mu_{\mathtt{ref}}$: typing rules for terms and evaluation contexts

the *free variables* of M. This language has two binders, the standard λ-abstraction, and the μ binder for *continuation variables* $\mathtt{c}, d$ [22].

A store, ranged over by $\mathtt{S}, \mathtt{T}$, is a finite mapping from locations to values. $\mathtt{S}(\ell)$ stands for the value associated to ℓ in S. We use notation $\mathtt{S} \cdot [\ell \mapsto \mathtt{V}]$ for the extension of S with a mapping for ℓ, which is only defined if ℓ is not defined in S. $\mathtt{S}[\ell \mapsto \mathtt{V}]$ denotes the store S in which the value associated to ℓ is updated.

The operational semantics $\mapsto_{\mathsf{op}}$ of the $\lambda\mu_{\mathtt{ref}}$-calculus is defined over *configurations*, which are pairs $(\mathtt{M}, \mathtt{S})$ formed by a term and a store. It is given by the following rules:

$$\begin{array}{ll}
(\mathtt{E}[(\lambda x.\mathtt{M})\mathtt{V}], \mathtt{S}) & \mapsto_{\mathsf{op}} (\mathtt{E}[\mathtt{M}\{x := \mathtt{V}\}], \mathtt{S}) \\
(\mathtt{E}[\mathtt{let}\ x = \mathtt{V}\ \mathtt{in}\ \mathtt{M}], \mathtt{S}) & \mapsto_{\mathsf{op}} (\mathtt{E}[\mathtt{M}\{x := \mathtt{V}\}], \mathtt{S}) \\
(\mathtt{E}[\mathtt{if}\ \mathtt{true}\ \mathtt{then}\ \mathtt{N}_1\ \mathtt{else}\ \mathtt{N}_2], \mathtt{S}) & \mapsto_{\mathsf{op}} (\mathtt{E}[\mathtt{N}_1], \mathtt{S}) \\
(\mathtt{E}[\mathtt{if}\ \mathtt{false}\ \mathtt{then}\ \mathtt{N}_1\ \mathtt{else}\ \mathtt{N}_2], \mathtt{S}) & \mapsto_{\mathsf{op}} (\mathtt{E}[\mathtt{N}_2], \mathtt{S}) \\
(\mathtt{E}[\mathtt{new}\ x = \mathtt{V}\ \mathtt{in}\ \mathtt{M}], \mathtt{S}) & \mapsto_{\mathsf{op}} (\mathtt{E}[\mathtt{M}\{x := \ell\}], \mathtt{S} \cdot [\ell \mapsto \mathtt{V}]) \\
(\mathtt{E}[\ell := \mathtt{V}], \mathtt{S}) & \mapsto_{\mathsf{op}} (\mathtt{E}[()], \mathtt{S}[\ell \mapsto \mathtt{V}]) \\
(\mathtt{E}[!\ell], \mathtt{S}) & \mapsto_{\mathsf{op}} (\mathtt{E}[\mathtt{S}(\ell)], \mathtt{S}) \\
(\mathtt{E}[\ell = \ell], \mathtt{S}) & \mapsto_{\mathsf{op}} (\mathtt{E}[\mathtt{true}], \mathtt{S}) \\
(\mathtt{E}[\ell = \ell'], \mathtt{S}) & \mapsto_{\mathsf{op}} (\mathtt{E}[\mathtt{false}], \mathtt{S}) \\
(\mathtt{E}[\mu\mathtt{c}.\mathtt{M}], \mathtt{S}) & \mapsto_{\mathsf{op}} (\mathtt{M}\{\mathtt{c} := \mathtt{E}\})
\end{array}$$

The typing system for terms is given by the rules in Figure 1. We chose here a typing judgement with a single typing context Γ, so that continuation variables are given types of the shape $\neg\sigma$. Such negated types are also used to

type evaluation contexts, as specified by the two last rules in Figure 1. While we cannot store a continuation variable $\mathtt{c}$ in a reference, we can always store its associated function $\lambda x.[\mathtt{c}]x$. Typing rules force terms of type $\bot$ to be of the shape $[d]\mathtt{M}$, following Parigot's original presentation of the $\lambda\mu$-calculus [22].

We also consider a typing judgement of the shape $\Sigma \vdash \mathtt{C} : (\Gamma;\sigma) \rightsquigarrow (\Delta;\tau)$, for contexts $\mathtt{C}$ that take terms $\mathtt{M}$ of type $\Sigma;\Gamma \vdash \mathtt{M} : \sigma$ and produce terms of type $\Sigma;\Delta \vdash \mathtt{C}[\mathtt{M}] : \tau$. The typing rules defining this judgement are standard and not recalled here.

In the following, we consider the $\lambda\mu\nu$-calculus, the fragment of the $\lambda\mu_{\mathtt{ref}}$-calculus that only handles references of type $\mathbf{ref}_{\mathtt{Unit}}$. That is, for any reference type $\mathbf{ref}_\sigma$ appearing in the typing derivation, we have $\sigma = \mathtt{Unit}$.

We use $\mathtt{a}, \mathtt{b}, \ldots$ to range over locations of type $\mathbf{ref}_{\mathtt{Unit}}$, also called atoms, and introduce the slightly shorter notation $\mathtt{new}\ n\ \mathtt{in}\ \mathtt{M}$ to stand for $\mathtt{new}\ n = ()\ \mathtt{in}\ \mathtt{M}$ in $\lambda\mu\nu$. The syntax for values and terms of the $\lambda\mu\nu$-calculus is thus:

$$\begin{array}{ll} \text{Values} & \mathtt{V},\mathtt{W} \triangleq x \mid () \mid \lambda x.\mathtt{N} \mid \mathtt{true} \mid \mathtt{false} \mid \mathtt{a} \\ \text{Terms} & \mathtt{M},\mathtt{N} \triangleq \mathtt{V} \mid \mathtt{let}\ x = \mathtt{M}\ \mathtt{in}\ \mathtt{N} \mid \mathtt{V}\mathtt{W} \mid \mathtt{if}\ \mathtt{V}\ \mathtt{then}\ \mathtt{N}_1\ \mathtt{else}\ \mathtt{N}_2 \mid \mathtt{V} = \mathtt{W} \mid \mathtt{new}\ n\ \mathtt{in}\ \mathtt{M} \\ & \quad \mid \mu\mathtt{c}.\mathtt{M} \end{array}$$

In this setting, we see stores $\mathtt{S}$ directly as sets of atoms, all mapping to the unit value $()$. For $\mathtt{L}$ a set of atoms. we write $\widehat{\mathtt{L}}$ for the store that maps atoms in $\mathtt{L}$ to the unit value $()$.

We consider the following extension of the typing judgement respectively to stores $\mathtt{S}$ and value-mapping substitutions γ:

$$\frac{\forall \ell \in \mathrm{dom}(\mathtt{S}), \Sigma;\varnothing \vdash \mathtt{S}(\ell) : \Sigma(\ell) \qquad \mathrm{dom}(\mathtt{S}) = \mathrm{dom}(\Sigma)}{\vdash \mathtt{S} : \Sigma}$$

$$\frac{\forall x \in \mathrm{dom}(\Gamma), \Sigma;\Delta \vdash \gamma(x) : \Gamma(x) \qquad \mathrm{dom}(\gamma) = \mathrm{dom}(\Gamma)}{\Sigma;\Delta \vdash \gamma : \Gamma}$$

Definition 1. *A* normal form $(\mathtt{M},\mathtt{S})$ *is a configuration that is irreducible for the reduction relation* $\mapsto_{\mathsf{op}}$. *We write* $(\mathtt{M},\mathtt{S}) \Downarrow \mathtt{N}$ *when there exists a store* $\mathtt{T}$ *such that* $(\mathtt{M};\mathtt{S}) \mapsto_{\mathsf{op}}^{*} (\mathtt{N};\mathtt{T})$ *and that* $(\mathtt{N};\mathtt{T})$ *is a normal form.*

We call the types $\mathtt{Bool}, \mathtt{Unit}$ and $\mathbf{ref}_\sigma$ *positive* types, while $\sigma \to \tau$ and $\neg\sigma$ are called *negative* types. By only allowing free variables of negative types, we can provide a sharp characterization of normal forms.

Theorem 2. *Taking a term* $\mathtt{M}$ *such that* $\Sigma;\Gamma \vdash \mathtt{M} : \bot$ *with* Γ *a typing context mapping variables to negative types, if* $(\mathtt{M},\mathtt{S})$ *is in normal form with respect to* $\mapsto_{\mathsf{op}}$, *then* $\mathtt{M}$ *is either a named value* $[\mathtt{c}]\mathtt{V}$ *or a neutral term* $\mathtt{E}[x\mathtt{V}]$.

Moreover, for any configuration $(\mathtt{M},\mathtt{S})$ *such that* $\mathtt{M}$ *is in* $\lambda\mu\nu$, $\Sigma;\Gamma \vdash \mathtt{M} : \bot$ *and* $\vdash \mathtt{S} : \Sigma$, *there exists* $\mathtt{N}$ *such that* $(\mathtt{M},\mathtt{S}) \Downarrow \mathtt{N}$.

Definition 3. *Taking two terms* $\mathtt{M},\mathtt{N}$ *such that* $\Sigma;\Gamma \vdash \mathtt{M} : \sigma$ *and* $\Sigma;\Gamma \vdash \mathtt{N} : \sigma$, *we say that they are* contextually equivalent, *written* $\Sigma;\Gamma \vdash \mathtt{M} \simeq_{ctx} \mathtt{N} : \sigma$, *when for all*

continuation variable $\mathtt{c}$ *and context* $\mathtt{C}$ *such that* $\Sigma \vdash \mathtt{C} : (\Gamma; \sigma) \rightsquigarrow (\mathtt{c} : \neg\mathtt{Unit}; \bot)$, *and for all store* $\mathtt{S}$ *such that* $\vdash \mathtt{S} : \Sigma'$, *we have* $(\mathtt{C}[\mathtt{M}], \mathtt{S}) \Downarrow [\mathtt{c}]()$ *if and only if* $(\mathtt{C}[\mathtt{N}], \mathtt{S}) \Downarrow [\mathtt{c}]()$.

In the definition above, we use $\lambda\mu_{\mathtt{ref}}$ contexts to observe $\lambda\mu\nu$ terms. Such contexts can use higher-order references, and lead to divergent computations. For this reason, testing for convergence to () is enough when defining $\simeq_{ctx}$.

3 Operational Game Semantics

We now introduce a fully-abstract trace semantics for $\lambda\mu_{\mathtt{ref}}$ programs. We follow a modular presentation, inspired by the one provided by Laird in [15], where the semantics is built from a synchronization product of three LTS:

- the Interactive LTS $\mathcal{L}_{\mathsf{I}}$, that represents the raw interactions of programs with their environment.
- the Typing LTS $\mathcal{L}_{\mathsf{Ty}}$, that keeps track of the polarization and types of names exchanged, to preserve well-typedness.
- the Disclosing LTS $\mathcal{L}_{\mathsf{Di}}$, that prevents the environment from using private resources that have not been disclosed by Proponent.

3.1 Abstract values

To represent the interaction between the program and its environment, we distinguish between values that we can observe and values that we can interact with. The two players only exchange observable values, called *abstract values* in this paper. They are defined by the following grammar:

$$\mathtt{A}, \mathtt{B} \triangleq \mathtt{f} \mid \mathtt{a} \mid \mathtt{true} \mid \mathtt{false} \mid ()$$

with $\mathtt{f}$ a *function name*, that is a variable used to represent functions exchanged between the two players. These correspond to the positive part of values, and are also called *ultimate patterns* in [17]. Like for terms, $\mathsf{supp}(\mathtt{A})$ stands for the set of atoms occurring in $\mathtt{A}$. We consider the typing judgement $\Delta \Vdash \mathtt{A} : \sigma$ for abstract values, with σ a positive type, that is defined similarly as done for terms.

Then we introduce the abstraction relation $\nearrow$ that transforms a value $\mathtt{V}$ into a pair $(\mathtt{A}, \gamma)$ formed by an abstract value and a substitution, such that $\mathtt{A}\{\gamma\} = \mathtt{V}$:

$$\frac{\mathtt{f}, \mathtt{g} \text{ function names}}{\mathtt{f} \nearrow (\mathtt{g}, [\mathtt{g} \mapsto \mathtt{f}])} \qquad \frac{}{() \nearrow ((), \varepsilon)} \qquad \frac{\mathtt{b} \in \{\mathtt{true}, \mathtt{false}\}}{\mathtt{b} \nearrow (\mathtt{b}, \varepsilon)} \qquad \frac{\mathtt{a} \text{ an atom}}{\mathtt{a} \nearrow (\mathtt{a}, \varepsilon)}$$

$$\frac{}{\lambda x.\mathtt{M} \nearrow (\mathtt{f}, [\mathtt{f} \mapsto \lambda x.\mathtt{M}])}$$

3.2 Labelled Transition Systems

The two players, Opponent and Proponent, exchange *moves*, which are in one of six forms:

P-question	P-answer	O-question	O-answer	P-init question	O-init question
$\overline{\mathsf{f}}(\mathtt{A},\mathtt{c})$	$\overline{\mathtt{c}}(\mathtt{A})$	$\mathsf{f}(\mathtt{A},\mathtt{c})$	$\mathtt{c}(\mathtt{A})$	$\overline{?}(\overrightarrow{\mathtt{A}_i})$	$?(\overrightarrow{\mathtt{A}_i})$

We use $\mathbf{m}$ to range over moves, and $\mathbf{p}$ (resp. $\mathbf{o}$) to range over Proponent (resp. Opponent) moves. Initial questions are the introductory moves. In contrast with other moves, they can introduce multiple abstract values in a row, which is useful to instantiate all the variables of a typing context Γ. They use a distinguished function name ?.

Traces $\mathbf{t}$ are sequences of moves. We write $\overline{\mathbf{m}}$ for the corresponding move with reversed polarity (input switched to output, and vice-versa). We extend this definition to switch traces, written $\overline{\mathbf{t}}$.

The three labelled transition systems we define are instances of the following definition:

Definition 4. *A labelled transition system (LTS)* $\mathcal{L}$ *is a triple of the form* $(\text{Confs}, \text{Actions}, \rightarrow)$. Confs *is a set of configurations* $\mathbb{C}, \mathbb{D}$. Actions *is a set of actions* $\mathbf{a}$*, formed by the* moves $\mathbf{m}$*, together with a silent action* $\mathtt{op}$*, corresponding to internal computations. Relation* $\rightarrow \subseteq \text{Confs} \times \text{Actions} \times \text{Confs}$ *is the labelled transition relation. We write* $\mathbb{C} \xrightarrow{\mathbf{a}} \mathbb{D}$ *for* $(\mathbb{C}, \mathbf{a}, \mathbb{D}) \in \rightarrow$.

Taking $\mathbb{C}$ a configuration of an LTS $\mathcal{L}$, we write $\mathbf{Tr}_{\mathcal{L}}(\mathbb{C})$ for the set of traces, as sequences of moves generated by this LTS over $\mathbb{C}$ (so with $\mathtt{op}$ actions removed). We write $\mathbb{C} \simeq_{\mathsf{tr}} \mathbb{D}$ for the trace equivalence relation, which equates configurations $\mathbb{C}, \mathbb{D}$ when both have the same set of traces.

3.3 Interactive LTS

We consider *interactive configurations* $\mathbb{I}; \mathbb{J} \in \text{IConfs}$ which are either passive of the shape $\langle \mathtt{S}; \gamma \rangle$, or active of the shape $\langle \mathtt{M}; \mathtt{S}; \gamma \rangle$ with $\mathtt{M}$ a term, $\mathtt{S}$ a store, and γ a substitution. The Interactive LTS $\mathcal{L}_{\mathsf{I}}$ is then defined as the triple $(\text{IConfs}, \text{Actions}, \rightarrow_{\mathsf{I}})$ with relation $\rightarrow_{\mathsf{I}}$ defined in Figure 2.

The two rules for Proponent moves describe transitions performed by normal forms and make use of the abstraction relation. In the two rules for Opponent, the notation $\mathtt{S} \odot [\widehat{\mathsf{supp}(\mathtt{A})}]$ stands for $\mathtt{S}$ extended with a binding $\mathtt{a} \mapsto ()$ in the case when $\mathtt{A} = \mathtt{a}$ and $\mathtt{a}$ is fresh for Proponent, and simply $\mathtt{S}$ otherwise: Proponent extends its store when a new atom is received.

3.4 Typing LTS

We consider *type-context configurations* $\mathbb{S}, \mathbb{T} \in \text{Confs}_{\mathsf{Ty}}$ which are either active of the shape $\langle \Delta_{\mathsf{O}} \mid \bot; \Delta_{\mathsf{P}} \rangle$ or passive of the shape $\langle \Delta_{\mathsf{O}} \mid \Delta_{\mathsf{P}} \rangle$, with $\Delta_{\mathsf{O}}, \Delta_{\mathsf{P}}$ two disjoint typing contexts that map variables to *negative* types.

$$\mathsf{op}\ \frac{(\mathtt{M};\mathtt{S}) \mapsto_{\mathsf{op}} (\mathtt{N};\mathtt{T})}{\langle \mathtt{M};\mathtt{S};\gamma\rangle \xrightarrow{\mathsf{op}}_{\mathsf{I}} \langle \mathtt{N};\mathtt{T};\gamma\rangle}$$

$$\mathsf{PQ}\ \frac{\mathtt{V} \not\Rightarrow (\mathtt{A};\gamma')}{\langle \mathtt{E}[\mathtt{fV}];\mathtt{S};\gamma\rangle \xrightarrow{\overline{\mathtt{f}}(\mathtt{A},\mathtt{c})}_{\mathsf{I}} \langle \mathtt{S};\gamma\cdot\gamma'\cdot[\mathtt{c}\mapsto\mathtt{E}]\rangle} \qquad \frac{\mathtt{V} \not\Rightarrow (\mathtt{A};\gamma')}{\langle [\mathtt{c}]\mathtt{V};\mathtt{S};\gamma\rangle \xrightarrow{\overline{\mathtt{c}}(\mathtt{A})}_{\mathsf{I}} \langle \mathtt{S};\gamma\cdot\gamma'\rangle}\ \mathsf{PA}$$

$$\mathsf{OQ}\ \frac{}{\langle \mathtt{S};\gamma\rangle \xrightarrow{\mathtt{f}(\mathtt{A},\mathtt{c})}_{\mathsf{I}} \langle [\mathtt{c}]\gamma(\mathtt{f})\mathtt{A};\mathtt{S}\odot[\widehat{\mathsf{supp}(\mathtt{A})}];\gamma\rangle} \qquad \frac{}{\langle \mathtt{S};\gamma\rangle \xrightarrow{\mathtt{c}(\mathtt{A})}_{\mathsf{I}} \langle \gamma(\mathtt{c})[\mathtt{A}];\mathtt{S}\odot[\widehat{\mathsf{supp}(\mathtt{A})}];\gamma\rangle}\ \mathsf{OA}$$

Fig. 2. Definition of $\mathcal{L}_{\mathsf{I}}$, the Interactive LTS: transitions of interactive configurations

$$\mathsf{PQ}\ \frac{\Delta_{\mathsf{O}}(\mathtt{f}) = \sigma \to \tau \qquad \Delta \Vdash \mathtt{A} : \sigma}{\langle \Delta_{\mathsf{O}} \mid \bot ; \Delta_{\mathsf{P}}\rangle \xrightarrow{\overline{\mathtt{f}}(\mathtt{A},\mathtt{c})}_{\mathsf{Ty}} \langle \Delta_{\mathsf{O}} \mid \Delta_{\mathsf{P}}, \Delta, \mathtt{c} : \neg\tau\rangle} \qquad \frac{\Delta_{\mathsf{O}}(\mathtt{c}) = \neg\sigma \qquad \Delta \Vdash \mathtt{A} : \sigma}{\langle \Delta_{\mathsf{O}} \mid \bot ; \Delta_{\mathsf{P}}\rangle \xrightarrow{\overline{\mathtt{c}}(\mathtt{A})}_{\mathsf{Ty}} \langle \Delta_{\mathsf{O}} \mid \Delta_{\mathsf{P}}, \Delta\rangle}\ \mathsf{PA}$$

$$\mathsf{OQ}\ \frac{\Delta_{\mathsf{P}}(\mathtt{f}) = \sigma \to \tau \qquad \Delta \Vdash \mathtt{A} : \sigma}{\langle \Delta_{\mathsf{O}} \mid \Delta_{\mathsf{P}}\rangle \xrightarrow{f(\mathtt{A},\mathtt{c})}_{\mathsf{Ty}} \langle \Delta_{\mathsf{O}}, \Delta, \mathtt{c} : \neg\tau \mid \bot ; \Delta_{\mathsf{P}}\rangle} \qquad \frac{\Delta_{\mathsf{P}}(\mathtt{c}) = \neg\sigma \qquad \Delta \Vdash \mathtt{A} : \sigma}{\langle \Delta_{\mathsf{O}} \mid \Delta_{\mathsf{P}}\rangle \xrightarrow{\mathtt{c}(\mathtt{A})}_{\mathsf{Ty}} \langle \Delta_{\mathsf{O}}, \Delta \mid \bot ; \Delta_{\mathsf{P}}\rangle}\ \mathsf{OA}$$

Fig. 3. Definition of $\mathcal{L}_{\mathsf{Ty}}$, the typing LTS: transitions of type-context configurations

The Interactive LTS $\mathcal{L}_{\mathsf{I}}$ is then defined as the triple $(\mathsf{Confs}_{\mathsf{Ty}}, \mathsf{Actions}, \rightarrow_{\mathsf{Ty}})$ with relation $\rightarrow_{\mathsf{Ty}}$ defined in Figure 3. Notice that the type of the active term is $\bot$ since the reduction relation $\mapsto_{\mathsf{op}}$ is well-defined only on terms of this type.

Typing configurations can be used to specify interactive configurations, via the following validity judgement.

Definition 5. *An interactive configuration $\mathbb{I}$ is said to be validated by a typing configuration $\mathbb{S}$, written $\mathbb{I} \triangleright \mathbb{S}$, when:*

- *either $\mathbb{I} = \langle \mathtt{S};\gamma\rangle$, $\mathbb{S} = \langle \Delta_{\mathsf{O}} \mid \Delta_{\mathsf{P}}\rangle$, and there exists a store typing context Σ such that $\Sigma; \Delta_{\mathsf{O}} \vdash \gamma : \Delta_{\mathsf{P}}$ and $\vdash \mathtt{S} : \Sigma$,*
- *or $\mathbb{I} = \langle \mathtt{M};\mathtt{S};\gamma\rangle$, $\mathbb{S} = \langle \Delta_{\mathsf{O}} \mid \bot ; \Delta_{\mathsf{P}}\rangle$, and there exists a store typing context Σ such that $\Sigma; \Delta_{\mathsf{O}} \vdash \mathtt{M} : \bot$, $\Sigma; \Delta_{\mathsf{O}} \vdash \gamma : \Delta_{\mathsf{P}}$ and $\vdash \mathtt{S} : \Sigma$.*

3.5 Disclosing LTS

In order to enforce a *non-omniscient* condition on Opponent transitions, we introduce a Disclosing LTS $\mathcal{L}_{\mathsf{Di}} \triangleq (\mathsf{DConfs}, \mathsf{Actions}, \rightarrow_{\mathsf{Di}})$ whose configurations DConfs are pairs of sets of locations $\langle \mathtt{L};\mathtt{D}\rangle$ with $\mathtt{D}$ a set of atoms contained in $\mathtt{L}$. The transition relation $\rightarrow_{\mathsf{Di}}$ is defined in Figure 4. The condition $\mathtt{L} \cap \mathsf{supp}(\mathbf{o}) \subseteq \mathtt{D}$ corresponds to the fact that Opponent cannot play Proponent atoms that have not been disclosed yet, i.e. not in $\mathtt{D}$.

$$\mathsf{op}\ \frac{}{\langle \mathsf{L};\mathsf{D}\rangle \xrightarrow{\mathsf{op}}_{\mathsf{Di}} \langle \mathsf{L}\cup\mathsf{L}';\mathsf{D}\rangle}$$

$$\mathsf{PQ/PA}\ \frac{}{\langle \mathsf{L};\mathsf{D}\rangle \xrightarrow{\mathbf{p}}_{\mathsf{Di}} \langle \mathsf{L};\mathsf{D}\cup \mathsf{supp}(\mathbf{p})\rangle} \qquad \frac{\mathsf{L}\cap \mathsf{supp}(\mathbf{o})\subseteq \mathsf{D}}{\langle \mathsf{L};\mathsf{D}\rangle \xrightarrow{\mathbf{o}}_{\mathsf{Di}} \langle \mathsf{L}\cup \mathsf{supp}(\mathbf{o});\mathsf{D}\cup \mathsf{supp}(\mathbf{o})\rangle}\ \mathsf{OQ/OA}$$

Fig. 4. Definition of $\mathcal{L}_{\mathsf{Di}}$, the Disclosing LTS

Definition 6. *An interactive configuration* $\mathbb{I}$ *is said to be validated by a disclosing configuration* $\mathbb{D} = \langle \mathsf{L};\mathsf{D}\rangle$*, written* $\mathbb{I} \triangleright \mathbb{D}$*, if when writing* S *for the store component of* $\mathbb{I}$*, we have* $\mathrm{dom}(\mathsf{S}) = \mathsf{L}$.

3.6 Operational Game Semantics: LTS and Trace Equivalence

The *Operational Game Semantics* (OGS) LTS $\mathcal{L}_{\mathsf{OGS}} \triangleq (\mathrm{Confs}_{\mathsf{ogs}}, \mathrm{Actions}, \xrightarrow{\mathbf{a}}_{\mathsf{ogs}})$ is defined over configurations $\mathbb{G},\mathbb{H} \in \mathrm{Confs}_{\mathsf{ogs}}$ of the shape $(\mathbb{I},\mathbb{S},\mathbb{D})$, with $\mathbb{I}\triangleright\mathbb{S}$ and $\mathbb{I}\triangleright\mathbb{D}$, or over initial configurations $\langle \Sigma;\Gamma \vdash \mathtt{M} : \sigma\rangle$ for Proponent and $\langle \mathtt{c} : \neg\mathtt{Unit} \vdash (\mathtt{S};\delta) : (\Sigma;\Gamma)\rangle$ for Opponent. Its transition relation is defined by the following rules:

$$\frac{\mathbb{I} \xrightarrow{\mathbf{a}}_{\mathsf{I}} \mathbb{J} \qquad \mathbb{S} \xrightarrow{\mathbf{a}}_{\mathsf{Ty}} \mathbb{T} \qquad \mathbb{D} \xrightarrow{\mathbf{a}}_{\mathsf{Di}} \mathbb{E} \qquad \mathbb{J}\triangleright\mathbb{T} \qquad \mathbb{J}\triangleright\mathbb{E}}{(\mathbb{I},\mathbb{S},\mathbb{D}) \xrightarrow{\mathbf{a}}_{\mathsf{ogs}} (\mathbb{J},\mathbb{T},\mathbb{E})}$$

$$\frac{\Gamma = \overrightarrow{(x_i : \sigma_i)} \qquad \overrightarrow{\Delta_i \Vdash \mathtt{A}_i : \sigma_i} \qquad \mathtt{L} = (\cup_i \mathsf{supp}(\mathtt{A}_i)) \cup \mathrm{dom}(\Sigma)}{\langle \Sigma;\Gamma \vdash \mathtt{M} : \bot\rangle \xrightarrow{?(\overrightarrow{\mathtt{A}_i})}_{\mathsf{ogs}} \left(\langle \mathtt{M}\{\overrightarrow{x_i := \mathtt{A}_i}\}; \widehat{\mathtt{L}}; \varepsilon\rangle, \langle \overrightarrow{\Delta_i} | \bot; \varnothing\rangle, \langle \mathtt{L};\mathtt{L}\rangle\right)}$$

$$\frac{\Gamma = \overrightarrow{(x_i : \sigma_i)} \qquad \overrightarrow{\delta(x_i) \mathrel{/\!\!/} (\mathtt{A}_i;\gamma_i)} \qquad \overrightarrow{\Delta_i \Vdash \mathtt{A} : \sigma_i} \qquad \mathtt{L} = \Sigma^{-1}(\mathbf{ref}_{\mathtt{Unit}})}{\langle \mathtt{c} : \neg\mathtt{Unit} \vdash (\mathtt{S};\delta) : (\Sigma;\Gamma)\rangle \xrightarrow{\overline{?}(\overrightarrow{\mathtt{A}_i})}_{\mathsf{ogs}} \left(\langle \mathtt{S};\overrightarrow{\gamma_i}\rangle, \langle \mathtt{c} : \neg\mathtt{Unit} | \overrightarrow{\Delta_i}\rangle, \langle \mathtt{L};\mathtt{L}\rangle\right)}$$

The initial question generated by $\langle \Sigma;\Gamma \vdash \mathtt{M} : \sigma\rangle$ provides a way for Opponent to instantiate variables of Γ with abstract values. In this setting Σ only contains atoms since $\mathtt{M}$ is a term of $\lambda\mu\nu$. The transition for $\langle \mathtt{c} : \neg\mathtt{Unit} \vdash (\mathtt{S};\delta) : (\Sigma;\Gamma)\rangle$ represents this behavior from the point of view of Opponent. Since contexts belong to $\lambda\mu_{\mathtt{ref}}$, these initial configurations come equipped with an initial store $\mathtt{S}$ of type Σ, but only the locations of type $\mathbf{ref}_{\mathtt{Unit}}$ are considered to be disclosed, since the other ones cannot be used by Proponent. The continuation name $\mathtt{c}$ is used for Opponent to provide its final answer, which is of type $\mathtt{Unit}$, following the notion of observation used to define contextual equivalence.

We use notation $\overset{\mathbf{p}}{\Rightarrow}_{\mathsf{ogs}}$ to denote a $\mathbf{p}$ transition preceded by a possibly empty sequence of op transitions. Trace equivalence according to $\mathcal{L}_{\mathsf{OGS}}$ and contextual equivalence coincide.

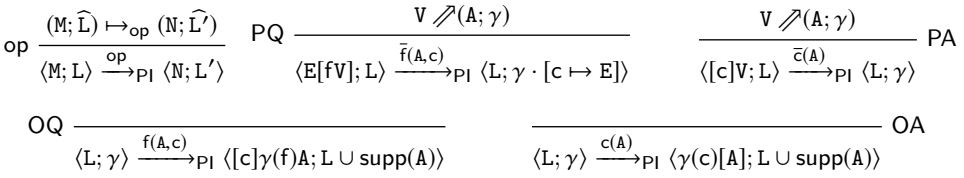


Fig. 5. Definition of $\mathcal{L}_{\mathsf{PI}}$: transitions of prime interactive configurations

Theorem 7. *Consider two terms* $\mathtt{M},\mathtt{N}$ *such that* $\Sigma;\Gamma \vdash \mathtt{M},\mathtt{N} : \sigma$.
We have $\langle \Sigma;\Gamma \vdash \mathtt{M} : \sigma\rangle \simeq_{\mathsf{tr}} \langle \Sigma;\Gamma \vdash \mathtt{N} : \sigma\rangle$ *if and only if* $\Sigma;\Gamma \vdash \mathtt{M} \simeq_{ctx} \mathtt{N} : \sigma$.

Such a full-abstraction theorem was proven in [13] for *RefML*, that is the intuitionistic fragment of $\lambda\mu_{\mathtt{ref}}$-calculus, without control operators. It was also proven in [10] for *HOSC*, a variant of the $\lambda\mu_{\mathtt{ref}}$-calculus, with the call/cc operator, but without atom disclosure. Such a full-abstraction result being rather standard, we have chosen to present its proof in [9].

In the remainder of the paper, we focus on the $\lambda\mu\nu$-calculus. In particular, we only consider OGS configurations corresponding to $\lambda\mu\nu$ from now on.

4 Lassen Trees for the $\lambda\mu\nu$-calculus

4.1 POGS and POGS bipartite bisimulation

We introduce Lassen trees for terms of the $\lambda\mu\nu$-calculus, as a form of *linearized* version of $\mathcal{L}_{\mathsf{OGS}}$, where Opponent can interrogate a name provided by Proponent only once, immediately after it has been introduced. So we consider *prime interactive configurations* which are either passive of the shape $\langle \mathtt{L};\gamma\rangle$, or active of the shape $\langle \mathtt{M};\mathtt{L}\rangle$ with $\mathtt{M}$ a term, $\mathtt{L}$ a set of atoms, and γ a substitution. Compared to interactive configurations, the active configurations do not carry an environment γ. Furthermore, we have a set of atoms rather than a full store, since this LTS is defined only for the $\lambda\mu\nu$-calculus and not for the whole $\lambda\mu_{\mathtt{ref}}$-calculus.

The Prime Interactive LTS, $\mathcal{L}_{\mathsf{PI}}$, is then defined as $(\mathrm{Confs}_{\mathsf{PI}}, \mathrm{Actions}, \rightarrow_{\mathsf{PI}})$, with $\rightarrow_{\mathsf{PI}}$ defined in Figure 5.

The corresponding Typing LTS is defined using the transitions given in Figure 6, which are very close in spirit to the transitions in Figure 3.

The transitions for the Disclosing LTS for POGS are presented on Figure 7. We compare these with the Disclosing LTS for OGS (Figure 4) below.

The Prime Operational Game Semantics LTS is introduced as a synchronization product, together with initial transitions, like for OGS. More precisely, the synchronization between the interactive and typing LTSs requires that active configurations $\langle \mathtt{M};\mathtt{L}\rangle$ correspond to type-contexts of the shape $\langle \Delta_{\mathsf{O}} \mid \bot\rangle$, with $\Sigma;\Delta_O \vdash \mathtt{M} : \bot$ and $\vdash \widehat{\mathtt{L}} : \Sigma$, for some store typing context Σ. Accordingly, for passive configurations $\langle \mathtt{L};\gamma\rangle$, we synchronize with $\langle \Delta_{\mathsf{O}} \mid \Delta_{\mathsf{P}}\rangle$, and check that $\Sigma;\Delta_O \vdash \gamma : \Delta_P$ and $\vdash \widehat{\mathtt{L}} : \Sigma$, for some store typing context Σ.

$$\mathsf{PQ}\ \frac{\Delta_{\mathsf{O}}(\mathsf{f}) = \sigma \to \tau \qquad \Delta \Vdash \mathtt{A} : \sigma}{\langle \Delta_{\mathsf{O}} \mid \bot \rangle \xrightarrow{\overline{\mathsf{f}}(\mathtt{A},\mathsf{c})}_{\mathsf{PTy}} \langle \Delta_{\mathsf{O}} \mid \Delta, \mathsf{c} : \neg\tau \rangle} \qquad \frac{\Delta_{\mathsf{O}}(\mathsf{c}) = \neg\sigma \qquad \Delta \Vdash \mathtt{A} : \sigma}{\langle \Delta_{\mathsf{O}} \mid \bot \rangle \xrightarrow{\overline{\mathsf{c}}(\mathtt{A})}_{\mathsf{PTy}} \langle \Delta_{\mathsf{O}} \mid \Delta \rangle}\ \mathsf{PA}$$

$$\mathsf{OQ}\ \frac{\Delta_{\mathsf{P}}(\mathsf{f}) = \sigma \to \tau \qquad \Delta \Vdash \mathtt{A} : \sigma}{\langle \Delta_{\mathsf{O}} \mid \Delta_{\mathsf{P}} \rangle \xrightarrow{f(\mathtt{A},\mathsf{c})}_{\mathsf{PTy}} \langle \Delta_{\mathsf{O}}, \Delta, \mathsf{c} : \neg\tau \mid \bot \rangle} \qquad \frac{\Delta_{\mathsf{P}}(\mathsf{c}) = \neg\sigma \qquad \Delta \Vdash \mathtt{A} : \sigma}{\langle \Delta_{\mathsf{O}} \mid \Delta_{\mathsf{P}} \rangle \xrightarrow{\mathsf{c}(\mathtt{A})}_{\mathsf{PTy}} \langle \Delta_{\mathsf{O}}, \Delta \mid \bot \rangle}\ \mathsf{OA}$$

Fig. 6. Definition of $\mathcal{L}_{\mathsf{PTy}}$: transitions of prime type-context configurations

$$\mathsf{op}\ \frac{\mathsf{D}' \subseteq \mathsf{L}'}{\langle \mathsf{L}; \mathsf{D} \rangle \xrightarrow{\mathsf{op}}_{\mathsf{pd}} \langle \mathsf{L} \uplus \mathsf{L}'; \mathsf{D} \uplus \mathsf{D}' \rangle}$$

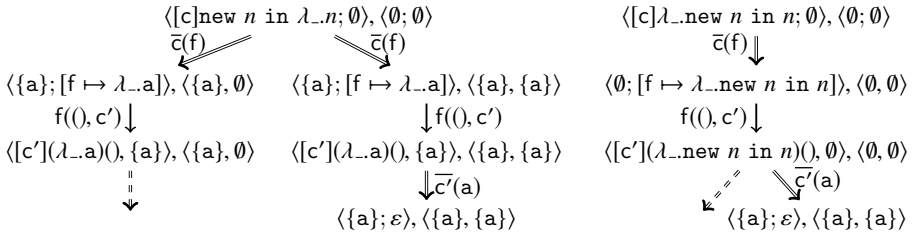
$$\mathsf{PQ/PA}\ \frac{\mathsf{supp}(\mathbf{p}) \subseteq \mathsf{D}}{\langle \mathsf{L}; \mathsf{D} \rangle \xrightarrow{\mathbf{p}}_{\mathsf{pd}} \langle \mathsf{L}; \mathsf{D} \rangle} \qquad \frac{\mathsf{L} \cap \mathsf{supp}(\mathbf{o}) \subseteq \mathsf{D}}{\langle \mathsf{L}; \mathsf{D} \rangle \xrightarrow{\mathbf{o}}_{\mathsf{pd}} \langle \mathsf{L} \cup \mathsf{supp}(\mathbf{o}); \mathsf{D} \cup \mathsf{supp}(\mathbf{o}) \rangle}\ \mathsf{OQ/OA}$$

Fig. 7. Definition of $\mathcal{L}_{\mathsf{PDi}}$: Disclosing LTS for POGS

To synchronize with the Disclosing LTS, whose states are of the form $\langle \mathsf{L}; \mathsf{D} \rangle$, we simply impose that the L component is the same in the state of $\mathcal{L}_{\mathsf{PI}}$, both for active and passive configurations.

We call $\mathcal{L}_{\mathsf{POGS}}$ the LTS obtained by synchronizing $\mathcal{L}_{\mathsf{PI}}$, $\mathcal{L}_{\mathsf{PTy}}$ and $\mathcal{L}_{\mathsf{PDi}}$. We write $\mathbb{P}, \mathbb{Q} \in \mathsf{Confs}_{\mathsf{POGS}}$ the configurations of $\mathcal{L}_{\mathsf{POGS}}$. The *Lassen tree* of a term is then defined as the unfolding of the $\mathcal{L}_{\mathsf{POGS}}$ on the initial active configuration associated with this term.

Example 8. The Lassen trees (omitting the typing configurations) for $[\mathsf{c}]\mathtt{new}\ n\ \mathtt{in}\ \lambda\_.n$ and $[\mathsf{c}]\lambda\_.\mathtt{new}\ n\ \mathtt{in}\ n$ are given by:



Due to the condition $\mathsf{supp}(\mathbf{p}) \subseteq \mathsf{D}$ in $\xrightarrow{\mathbf{p}}_{\mathsf{pd}}$, some configurations with terms in normal form do not have a corresponding Proponent transition. The dashed arrows correspond to op transitions that lead to such stuck configurations.

4.2 Bipartite Bisimulations for OGS and POGS

We consider typed relations on passive and active configurations, that is, we require related configurations to have the same type. This means in particular that the environment components γ of the two configurations have the same domain. In addition to the typing, we also enforce that both sets of disclosed atoms are identical.

Definition 9. *A* bipartite bisimulation *is a pair of relations* $(\mathcal{R}_{Act}, \mathcal{R}_{Pas})$ *respectively on active and passive configurations, such that:*

- *If* $(\mathbb{G}_1, \mathbb{G}_2) \in \mathcal{R}_{Pas}$ *then for all Opponent moves* $\mathbf{o}$ *and* $\mathbb{H}_1, \mathbb{H}_2$ *such that* $\mathbb{G}_1 \xrightarrow{\mathbf{o}} \mathbb{H}_1$ *and* $\mathbb{G}_2 \xrightarrow{\mathbf{o}} \mathbb{H}_2$*, we have* $(\mathbb{H}_1, \mathbb{H}_2) \in \mathcal{R}_{Act}$*.*
- *If* $(\mathbb{G}_1, \mathbb{G}_2) \in \mathcal{R}_{Act}$ *then there exists a Proponent move* $\mathbf{p}$ *and* $(\mathbb{H}_1, \mathbb{H}_2) \in \mathcal{R}_{Pas}$ *such that* $\mathbb{G}_1 \stackrel{\mathbf{p}}{\Rightarrow} \mathbb{H}_1$ *and* $\mathbb{G}_2 \stackrel{\mathbf{p}}{\Rightarrow} \mathbb{H}_2$*.*

An OGS-bipartite bisimulation *is a bipartite bisimulation defined over* $\mathcal{L}_{\mathsf{OGS}}$*, and a* POGS-bipartite bisimulation *is a bipartite bisimulation defined over* $\mathcal{L}_{\mathsf{POGS}}$*. We write* $\simeq_{\mathsf{ogs}}$ *and* $\simeq_{\mathsf{pogs}}$ *respectively for the greatest bipartite bisimulation respectively over* $\mathcal{L}_{\mathsf{OGS}}$ *and* $\mathcal{L}_{\mathsf{POGS}}$*.*

The following property follows from the fact that the transition relation is deterministic (up to the choice of fresh names).

Lemma 10. $\simeq_{\mathsf{ogs}}$ *coincides with trace equivalence on* OGS *configurations.*

For op transitions, the difference between OGS and POGS shows up in the disclosing LTS: in $\xrightarrow{\mathsf{op}}_{\mathsf{pd}}$, a D' component can be chosen non-deterministically. This observation is related to the existential quantification in the second clause of Definition 13. Both in $\mathcal{L}_{\mathsf{OGS}}$ and $\mathcal{L}_{\mathsf{POGS}}$, there is only one possible next visible (Proponent) move. However, in $\simeq_{\mathsf{pogs}}$, the game involves choosing an appropriate set of atoms to be disclosed along $\xrightarrow{\mathsf{op}}_{\mathsf{pd}}$ transitions. For instance, when constructing a POGS bipartite bisimulation between terms $\texttt{new}\ n\ \texttt{in}\ \lambda\_.n$ and $\lambda\_.\texttt{new}\ n\ \texttt{in}\ n$ from Example 8, we have two choices for the second step:

$$\big((\langle\{\mathsf{a}\}; [\mathsf{f} \mapsto \lambda\_.\mathsf{a}]\rangle, \langle\{\mathsf{a}\}, \emptyset\rangle), \quad (\langle\emptyset; [\mathsf{f} \mapsto \lambda\_.\texttt{new}\ n\ \texttt{in}\ n]\rangle, \langle\emptyset, \emptyset\rangle)\big)$$
$$\big((\langle\{\mathsf{a}\}; [\mathsf{f} \mapsto \lambda\_.\mathsf{a}]\rangle, \langle\{\mathsf{a}\}, \{\mathsf{a}\}\rangle), \quad (\langle\emptyset; [\mathsf{f} \mapsto \lambda\_.\texttt{new}\ n\ \texttt{in}\ n]\rangle, \langle\emptyset, \emptyset\rangle)\big)$$

The latter does not satisfy the constraint on the disclosed set, since the sets are not the same in the two configurations. The former leads to a stuck configuration: $(\langle[\mathsf{c}'](\lambda\_.\mathsf{a})(), \{\mathsf{a}\}\rangle, \langle\{\mathsf{a}\}, \emptyset\rangle)$ cannot perform any Proponent move. Thus the two programs are not equivalent.

4.3 Deciding $\simeq_{\mathsf{pogs}}$

We now study how to decide when two POGS configurations are bisimilar. First, trees generated by $\mathcal{L}_{\mathsf{POGS}}$ are of finite depth.

Lemma 11. *Taking a* POGS *configuration* $\mathbb{G}$*, any trace in* $\mathbf{Tr}_{\mathsf{POGS}}(\mathbb{G})$ *is finite.*

This lemma is proven using a biorthogonal logical predicate, following the use of biorthogonality to prove strong normalization of $\lambda\mu$-calculus [23], the computational metalanguage [18], and cut elimination for linear logic [8]. The proof can be found in [9].

Due to the non-determinism of atom generation in $\mapsto_{\mathsf{op}}$, of function name generation in $\nrightarrow$, and of name picking in Opponent transitions, the trees generated by $\mathcal{L}_{\mathsf{POGS}}$ are infinitely branching. To tame this infinite branching, we see the set of moves Moves and the set of configurations $\mathsf{Confs}_{\mathsf{POGS}}$ of $\mathcal{L}_{\mathsf{POGS}}$ as *nominal sets* [7] over atoms, function and continuation variables. So taking π a finite permutation over these sets, we write $\pi * X$ for the action of permutation π over elements of nominal set X. The transition relation $\rightarrow_{\mathsf{pogs}}$ of $\mathcal{L}_{\mathsf{POGS}}$ preserves this action of permutation, i.e., it is *equivariant*: if $\mathbb{P} \xrightarrow{\mathbf{m}}_{\mathsf{pogs}} \mathbb{Q}$ then for all finite permutation π, we have $\pi * \mathbb{P} \xrightarrow{\pi * \mathbf{m}}_{\mathsf{pogs}} \pi * \mathbb{Q}$.

One can then consider a variant $\mathcal{L}_{\mathsf{DPOGS}}$ of the POGS LTS which uses the same set of configurations as $\mathcal{L}_{\mathsf{POGS}}$, but whose transition relation $\rightarrow_{\mathsf{dpogs}}$ chooses fresh atoms and names deterministically. So $\rightarrow_{\mathsf{dpogs}}$ is then deterministic on op and Proponent actions, and finitely branching on Opponent actions.

We remark at this point that the notion of bipartite bisimulation $\simeq_{\mathsf{pogs}}$ introduced in Definition 13 is not suited for $\mathcal{L}_{\mathsf{DPOGS}}$. Indeed, it requires equality of actions in the bisimulation game, and also that configurations related by bisimulation have the same type. So we relax the definition of $\simeq_{\mathsf{pogs}}$ and work with ternary relations, adding a finite permutation of names and atoms in order to match the actions, rather than enforcing syntactic equality.

Definition 12. *A relation* $\mathcal{R} \subseteq \mathsf{Confs}_{\mathsf{POGS}} \times \mathsf{Confs}_{\mathsf{POGS}} \times \mathrm{Perm}$ *is said to be* valid *when, for all* $((\mathbb{I}, \mathbb{S}, \langle \_, \mathsf{D} \rangle), (\mathbb{J}, \mathbb{T}, \langle \_, \mathsf{D}' \rangle), \pi) \in \mathcal{R}$*, we have* $\mathbb{T} = \pi * \mathbb{S}$ *and* $\mathsf{D}' = \pi * \mathsf{D}$*.*

Definition 13. *A* relaxed bipartite bisimulation *is a pair of valid relations* $(\mathcal{R}_{Act}, \mathcal{R}_{Pas})$ *respectively on active and passive configurations such that:*

- *If* $(\mathbb{P}_1, \mathbb{P}_2, \pi) \in \mathcal{R}_{Pas}$ *then for all Opponent moves* $\mathbf{o}_1, \mathbf{o}_2$*, permutation* π' *extending* π*, and active* POGS *configurations* $\mathbb{Q}_1, \mathbb{Q}_2$ *satisfying* $\mathbf{o}_2 = \pi' * \mathbf{o}_1$*,* $\mathbb{P}_1 \xrightarrow{\mathbf{o}_1} \mathbb{Q}_1$ *and* $\mathbb{P}_2 \xrightarrow{\mathbf{o}_2} \mathbb{Q}_2$*, we have* $(\mathbb{Q}_1, \mathbb{Q}_2, \pi') \in \mathcal{R}_{Act}$*.*
- *If* $(\mathbb{P}_1, \mathbb{P}_2, \pi) \in \mathcal{R}_{Act}$ *then there exists a permutation* π' *extending* π*, two Proponent moves* $\mathbf{p}_1, \mathbf{p}_2$ *s.t.* $\mathbf{p}_2 = \pi' * \mathbf{p}_1$*, and two passive* POGS *configurations* $\mathbb{Q}_1, \mathbb{Q}_2$ *such that* $(\mathbb{Q}_1, \mathbb{Q}_2, \pi') \in \mathcal{R}_{Pas}$*,* $\mathbb{P}_1 \overset{\mathbf{p}_1}{\Longrightarrow} \mathbb{Q}_1$ *and* $\mathbb{P}_2 \overset{\mathbf{p}_2}{\Longrightarrow} \mathbb{Q}_2$*.*

We write $\simeq^r_{\mathsf{pogs}}$ for the greatest relaxed bipartite bisimulation over $\mathcal{L}_{\mathsf{POGS}}$. From the fact that $\rightarrow_{\mathsf{pogs}}$ is equivariant, we deduce that $\simeq^r_{\mathsf{pogs}}$ and $\simeq_{\mathsf{ogs}}$ coincide. Since $\mathcal{L}_{\mathsf{DPOGS}}$ generates finite Lassen trees, we deduce that the bisimulation game can be decided.

Theorem 14. *Taking two* POGS *configurations* $\mathbb{P}, \mathbb{Q}$*, we can decide if* $\mathbb{P} \simeq_{\mathsf{pogs}} \mathbb{Q}$*.*

4.4 Relating the Transitions in OGS and POGS

To relate the transitions in the OGS and in the POGS, we need to introduce some relations and operations on OGS configurations.

Definition 15. *Let* $\mathbb{G} = (\mathbb{I}, \mathbb{S}, \langle \mathsf{L}; \mathsf{D} \rangle)$ *and* $\mathbb{H} = (\mathbb{I}, \mathbb{S}, \langle \mathsf{L}; \mathsf{D}' \rangle)$ *be two* OGS *configurations. We write* $\mathbb{G} \subseteq_{\mathsf{Di}} \mathbb{H}$ *when* $\mathsf{D} \subseteq \mathsf{D}'$.

When $\mathbb{G} \subseteq_{\mathsf{Di}} \mathbb{H}$, the configurations only differ by their set of disclosed atoms.

Lemma 16. *If* $\mathbb{G} \subseteq_{\mathsf{Di}} \mathbb{H}$ *and* $\mathbb{G} \xrightarrow{\mathbf{a}}_{\mathsf{ogs}} \mathbb{G}'$ *then* $\mathbb{H} \xrightarrow{\mathbf{a}}_{\mathsf{ogs}} \mathbb{H}'$ *and* $\mathbb{G}' \subseteq_{\mathsf{Di}} \mathbb{H}'$.

Lemma 17. *Let* $\mathbb{P}$ *be an active prime configuration. We have the following:*

- *if* $\mathbb{P} \xrightarrow{\mathsf{op}}_{\mathsf{ogs}} \mathbb{P}'$, *then* $\mathbb{P} \xrightarrow{\mathsf{op}}_{\mathsf{pogs}} \mathbb{P}'$,
- *if* $\mathbb{P} \xrightarrow{\mathsf{op}}_{\mathsf{pogs}} \mathbb{P}'$, *then* $\mathbb{P} \xrightarrow{\mathsf{op}}_{\mathsf{ogs} \subseteq_{\mathsf{Di}}} \mathbb{P}'$.

In POGS, the disclosed set increases in op transitions as seen above, but not in **p** transitions. In a sense, disclosing in OGS is done only when needed, whereas in POGS, disclosing must be declared as soon as the atom is created. This is ensured by the additional condition $\mathsf{supp}(\mathbf{p}) \subseteq \mathsf{D}$ in the rule for $\xrightarrow{\mathbf{p}}_{\mathsf{pd}}$.

Lemma 18. *When* $\mathbb{P} \xrightarrow{\mathbf{p}}_{\mathsf{pogs}} \mathbb{P}'$ *with* $\mathbb{P}$ *active, we also have* $\mathbb{P} \xrightarrow{\mathbf{p}}_{\mathsf{ogs}} \mathbb{P}'$.

However, the converse does not always hold, specifically if an atom has been declared non-disclosed but still appears in the action **p**. Indeed, the transition $(\langle [\mathsf{c}]\mathsf{a}; \widehat{\mathsf{L}}; \emptyset \rangle, \mathbb{S}, \langle \mathsf{L}; \emptyset \rangle) \xrightarrow{\overline{\mathsf{c}}(\mathsf{a})}_{\mathsf{ogs}} (\langle \widehat{\mathsf{L}}; \emptyset \rangle, \mathbb{S}, \langle \mathsf{L}; \{\mathsf{a}\} \rangle)$ is valid for OGS, but has no counterpart in POGS, since $\langle \mathsf{L}; \emptyset \rangle$ cannot make the transition $\xrightarrow{\overline{\mathsf{c}}(\mathsf{a})}_{\mathsf{pd}}$.

Using the following notion of limit (on OGS configurations), we can intuitively replace D by its minimal extension, preventing this phenomenon from happening.

Definition 19. *Given a configuration* $\mathbb{G} = (\mathbb{I}, \mathbb{S}, \langle \mathsf{L}; \mathsf{D} \rangle)$, *we define its* limit *as:*

$$\mathbf{lim}(\mathbb{G}) \triangleq (\mathbb{I}, \mathbb{S}, \langle \mathsf{L}; \bigcup_{\mathsf{t} \in \mathtt{Traces}} (\mathsf{L} \cap \mathsf{D}') \rangle) \text{ with } \mathbb{G} \xrightarrow{\mathbf{t}}_{\mathsf{ogs}} (\_, \_, \langle \_, \mathsf{D}' \rangle).$$

We have that $\mathbb{G} \subseteq_{\mathsf{Di}} \mathbf{lim}(\mathbb{G})$ and **lim** is idempotent. We call *limit configurations* those configurations that are a limit (or alternatively, that are their own limit). Being a limit configuration is preserved by moves but not necessarily by op.

Lemma 20. *Let* $\mathbb{P}$ *be a limit configuration. If* $\mathbb{P} \xrightarrow{\mathbf{p}}_{\mathsf{ogs}} \mathbb{P}'$, *then* $\mathbb{P} \xrightarrow{\mathbf{p}}_{\mathsf{pogs}} \mathbb{P}'$.

For Opponent transitions, the situation is less simple since not all active OGS configurations are active POGS configurations. To circumvent that issue,

we reuse the tensor product from [12]. For two OGS configurations where at least one is passive, we define the tensor product, written $\otimes$, as follows:

$$(\mathbb{I}, \mathbb{S}, \mathbb{D}) \otimes (\mathbb{J}, \mathbb{T}, \mathbb{E}) = (\mathbb{I} \otimes \mathbb{J}, \mathbb{S} \otimes \mathbb{T}, \mathbb{D} \otimes \mathbb{E})$$

$$\langle \mathtt{S}; \gamma \rangle \otimes \langle \mathtt{S}'; \gamma' \rangle = \langle \mathtt{S} \cup \mathtt{S}'; \gamma \cdot \gamma' \rangle \qquad \langle \mathtt{M}; \mathtt{S}; \gamma \rangle \otimes \langle \mathtt{S}'; \gamma' \rangle = \langle \mathtt{M}; \mathtt{S} \cup \mathtt{S}'; \gamma \cdot \gamma' \rangle$$

$$\langle \mathtt{L}; \mathtt{D} \rangle \otimes \langle \mathtt{L}'; \mathtt{D}' \rangle = \langle \mathtt{L} \cup \mathtt{L}'; \mathtt{D} \cup \mathtt{D}' \rangle \text{ when } \begin{array}{l} \mathtt{D}' \cap \mathtt{L} \subseteq \mathtt{D} \\ \mathtt{D} \cap \mathtt{L}' \subseteq \mathtt{D}' \end{array}$$

The side conditions for the L and D components ensure that no shared atom is disclosed on one configuration but not the other.

We can then describe an active OGS configuration as the tensor of two POGS configurations (where $\mathtt{S} = \widehat{\mathtt{L}}$):

$$(\langle \mathtt{M}; \mathtt{S}; \gamma \rangle, \langle \Delta_O \vdash \bot; \Delta_P \rangle, \langle \mathtt{L}, \mathtt{D} \rangle) = (\langle \mathtt{M}; \mathtt{L} \rangle, \langle \Delta_O \vdash \bot \rangle, \langle \mathtt{L}, \mathtt{D} \rangle) \otimes (\langle \mathtt{L}; \gamma \rangle, \langle \Delta_O \vdash \Delta_P \rangle, \langle \mathtt{L}, \mathtt{D} \rangle)$$

Finally, we have the following property for opponent transitions:

Lemma 21. *When* $\mathbb{P} \xrightarrow{\mathbf{o}}_{\mathsf{pogs}} \mathbb{Q}$*, we have* $\mathbb{P} \xrightarrow{\mathbf{o}}_{\mathsf{ogs}} \mathbb{Q} \otimes \mathbb{P}$*.*
When $\mathbb{P} \xrightarrow{\mathbf{o}}_{\mathsf{ogs}} \mathbb{G}$*, we have* $\mathbb{P} \xrightarrow{\mathbf{o}}_{\mathsf{pogs}} \mathbb{Q}$ *with* $\mathbb{G} = \mathbb{Q} \otimes \mathbb{P}$*.*

5 Relating Bisimilarities in OGS and POGS

In this section, we show that $\simeq_{\mathsf{pogs}}$ can be used to characterize $\simeq_{\mathsf{ogs}}$ for the limit configurations introduced above. We rely for that on up-to techniques for bipartite bisimulation in OGS, which we introduce first.

5.1 Up-to techniques for $\simeq_{\mathsf{ogs}}$

The proofs in this section use the theory of compatible functions [27,25]. More details can be found in [9].

Definition 22 (Bipartite bisimulation up-to). *Given a function f, a bipartite bisimulation up to f is a pair $(\mathcal{R}_{Act}, \mathcal{R}_{Pas})$ such that:*

- *If $(\mathbb{G}_1, \mathbb{G}_2) \in \mathcal{R}_{Pas}$ then for all Opponent moves $\mathbf{o}$ and $\mathbb{H}_1, \mathbb{H}_2$ such that $\mathbb{G}_1 \xrightarrow{\mathbf{o}}_{\mathsf{ogs}} \mathbb{H}_1$ and $\mathbb{G}_2 \xrightarrow{\mathbf{o}}_{\mathsf{ogs}} \mathbb{H}_2$, we have $(\mathbb{H}_1, \mathbb{H}_2) \in f(\mathcal{R}_{Act})$.*
- *If $(\mathbb{G}_1, \mathbb{G}_2) \in \mathcal{R}_{Act}$ then there exists a Proponent move $\mathbf{p}$ and $(\mathbb{H}_1, \mathbb{H}_2) \in f(\mathcal{R}_{Pas})$ such that $\mathbb{G}_1 \overset{\mathbf{p}}{\Rightarrow}_{\mathsf{ogs}} \mathbb{H}_1$ and $\mathbb{G}_2 \overset{\mathbf{p}}{\Rightarrow}_{\mathsf{ogs}} \mathbb{H}_2$.*

We then define $\mathtt{hide}(\mathcal{R}_{Act}, \mathcal{R}_{Pas}) \triangleq (\subseteq_{\mathsf{Di}} \mathcal{R}_{Act} \supseteq_{\mathsf{Di}}, \subseteq_{\mathsf{Di}} \mathcal{R}_{Pas} \supseteq_{\mathsf{Di}})$. Recall that we still require that $\mathtt{hide}(\mathcal{R}_{Act}, \mathcal{R}_{Pas})$ only contains pairs of configurations with the same disclosed set. The soundness of hide can be proved using Lemma 16.

Lemma 23. hide *is a sound up-to technique, i.e. if $(\mathcal{R}_{Act}, \mathcal{R}_{Pas})$ is a bisimulation up to* hide*, then $(\mathcal{R}_{Act}, \mathcal{R}_{Pas}) \subseteq \simeq_{\mathsf{ogs}}$.*

Given a pair of relations $(\mathcal{R}_{Act}, \mathcal{R}_{Pas})$ on active and passive OGS configurations respectively, we define the following functions:

$$\texttt{tensor}(\mathcal{R}_{Act}, \mathcal{R}_{Pas}) \triangleq \left(\begin{array}{l} \{(\mathbb{G}_1 \otimes \mathbb{G}_2, \mathbb{H}_1 \otimes \mathbb{H}_2) \text{ s.t. } (\mathbb{G}_1, \mathbb{H}_1) \in \mathcal{R}_{Act}, (\mathbb{G}_2, \mathbb{H}_2) \in \mathcal{R}_{Pas}\}, \\ \{(\mathbb{G}_1 \otimes \mathbb{G}_2, \mathbb{H}_1 \otimes \mathbb{H}_2) \text{ s.t. } (\mathbb{G}_1, \mathbb{H}_1), (\mathbb{G}_2, \mathbb{H}_2) \in \mathcal{R}_{Pas}\} \end{array} \right)$$

$$\texttt{split}(\mathcal{R}_{Act}, \mathcal{R}_{Pas}) \triangleq \left(\begin{array}{ll} \{(\mathbb{G}_1, \mathbb{H}_1) & \text{s.t. } (\mathbb{G}_1 \otimes \mathbb{G}_2, \mathbb{H}_1 \otimes \mathbb{H}_2) \in \mathcal{R}_{Act}\}, \\ \{(\mathbb{G}_1, \mathbb{H}_1) & \text{s.t. } (\mathbb{G}_1 \otimes \mathbb{G}_2, \mathbb{H}_1 \otimes \mathbb{H}_2) \in \mathcal{R}_{Pas}\} \end{array} \right)$$

Lemma 24. $\texttt{split}(\simeq_{\mathsf{ogs}}) \subseteq \simeq_{\mathsf{ogs}}$.

`tensor` is not a sound up-to technique. It is nevertheless useful to reason about POGS bipartite bisimilar configurations; see Theorem 30 below.

5.2 Properties of the Limit (in OGS)

Lemma 25 (Monotonicity). *If* $\mathbb{G}$ *is passive and* $\mathbb{G} \xrightarrow{\mathbf{t}}_{\mathsf{ogs}} \mathbb{H}$*, then there exists* $\mathbb{G}'$ *such that* $\mathbb{G} \otimes \mathbb{G}' \subseteq_{\mathsf{Di}} \mathbb{H}$.

Lemma 25 shows that transitions can only increase the substitution and the store (corresponding to the $\mathbb{G}'$ component), and the set of disclosed atoms (represented by the use of $\subseteq_{\mathsf{Di}}$). More precisely, $\subseteq_{\mathsf{Di}}$ is required if some atoms from $\mathbb{G}$ are disclosed along the trace $\mathbf{t}$, in which case new ones can appear in $\mathbb{G}'$.

Lemma 25 is language specific. It does not hold when the language allows the content of the store to be modified (like, e.g. in $\lambda\mu_{\mathtt{ref}}$). Additionally, LTSs enforcing some local restriction on the usage of function or continuation names usually have extra components that are modified along the transitions; we return to this point in Section 7.

In a limit configuration (Definition 19), all atoms that may be disclosed at some point are disclosed. By Lemma 25, these atoms can be disclosed using a single trace.

Lemma 26. *Given a passive configuration* $\mathbb{G}$*, there exists a trace* $\mathbf{t}$ *and a configuration* $\mathbb{H}$ *such that* $\mathbb{G} \xrightarrow{\mathbf{t}}_{\mathsf{ogs}} \mathbf{lim}(\mathbb{G}) \otimes \mathbb{H}$.

The limit is also useful to relate transitions in OGS and in POGS as follows.

Lemma 27. *Take a* POGS *configuration* $\mathbb{P}$.
If $\mathbb{P}$ *is active and* $\mathbb{P} \xrightarrow{\mathbf{a}}_{\mathsf{ogs}} \mathbb{Q}$*, then* $\mathbf{lim}(\mathbb{P}) \xrightarrow{\mathbf{a}}_{\mathsf{pogs}} \mathbf{lim}(\mathbb{Q})$.
If $\mathbb{P}$ *is passive and* $\mathbb{P} \xrightarrow{\mathbf{o}}_{\mathsf{ogs}} \mathbb{Q} \otimes \mathbb{P}$*, then* $\mathbf{lim}(\mathbb{P}) \xrightarrow{\mathbf{o}}_{\mathsf{pogs}} \mathbf{lim}(\mathbb{Q})$.

All in all, we obtain that $\simeq_{\mathsf{ogs}}$ is a congruence for $\mathbf{lim}$. For $\mathcal{R}$ a relation over configurations, we write $\mathbf{lim}(\mathcal{R})$ for the set $\{(\mathbf{lim}(\mathbb{G}), \mathbf{lim}(\mathbb{H})) \mid (\mathbb{G}, \mathbb{H}) \in \mathcal{R}\}$.

Lemma 28. $\simeq_{\mathsf{ogs}}$ *is closed by computing the limit:* $\mathbf{lim}(\simeq_{\mathsf{ogs}}) \subseteq \simeq_{\mathsf{ogs}}$.

The case for passive configurations follows immediately from Lemmas 26 and 24.

The property of the limit might make us think that the disclosure process of an atom could be decided statically, by annotating `new` syntactically. The following example shows that it is not the case:

$$\lambda b.\texttt{new}\ n, m\ \texttt{in}\ \lambda\_.\texttt{if}\ b\ \texttt{then}\ n\ \texttt{else}\ m$$

Either n or m will be disclosed depending on the boolean b given by Opponent, but never both. So this term is indeed contextually equivalent to $\lambda b.\texttt{new}\ n\ \texttt{in}\ \lambda\_.n$.

5.3 Correspondence Between $\simeq_{\mathsf{ogs}}$ and $\simeq_{\mathsf{pogs}}$

Theorem 29 (From $\simeq_{\mathsf{ogs}}$ to $\simeq_{\mathsf{pogs}}$). *Consider two* POGS *configurations* $\mathbb{P}$ *and* $\mathbb{Q}$. *If* $\mathbb{P} \simeq_{\mathsf{ogs}} \mathbb{Q}$ *are both limit configurations, then* $\mathbb{P} \simeq_{\mathsf{pogs}} \mathbb{Q}$.

To reason about bisimilar POGS configurations, we use the closure of `tensor`, written $\widetilde{\texttt{tensor}}$. Intuitively, $\widetilde{\texttt{tensor}}(\mathcal{R}_{Act})$ contains the pairs $(\mathbb{G}_1 \otimes \mathbb{G}_2, \mathbb{H}_1 \otimes \mathbb{H}_2)$ with $(\mathbb{G}_1, \mathbb{H}_1) \in \mathcal{R}_{Act}$, $(\mathbb{G}_2, \mathbb{H}_2) \in \widetilde{\texttt{tensor}}(\mathcal{R}_{Pas})$, and $\widetilde{\texttt{tensor}}(\mathcal{R}_{Pas})$ contains the pairs $(\mathbb{G}_1 \otimes \mathbb{G}_2, \mathbb{H}_1 \otimes \mathbb{H}_2)$ with $(\mathbb{G}_1, \mathbb{H}_1) \in \mathcal{R}_{Pas}$, $(\mathbb{G}_2, \mathbb{H}_2) \in \widetilde{\texttt{tensor}}(\mathcal{R}_{Pas})$.

Theorem 30 (From $\simeq_{\mathsf{pogs}}$ to $\simeq_{\mathsf{ogs}}$). *Suppose* $\mathcal{R}$ *is a* POGS *bipartite bisimulation. Then* $\widetilde{\texttt{tensor}}(\mathcal{R})$ *is a* OGS *bipartite bisimulation up-to hiding.*

By Lemma 23, Theorem 30 means that if $\mathbb{P} \simeq_{\mathsf{pogs}} \mathbb{Q}$, then $\mathbb{P} \simeq_{\mathsf{ogs}} \mathbb{Q}$.

The correspondence between $\simeq_{\mathsf{ogs}}$ and $\simeq_{\mathsf{pogs}}$ is restricted to prime configurations as $\simeq_{\mathsf{pogs}}$ can only relate those. Having the additional conditions of configurations being limits is enough for our decidability result.

6 Related Work

The ν-calculus was introduced in [24], together with logical relations to reason over contextual equivalence for this language. These logical relations use a Kripke-style definition, worlds being defined as spans of atoms to keep track of the disclosed atoms, similar to the permutation we use in our relaxed bipartite bisimulations. They capture contextual equivalence for programs of *first order* type, but are an incomplete technique for higher-order programs. This entails a decidability result for the first-order fragment of the ν-calculus, since logical relations only quantify over finite objects at first-order types.

Categorical models of the ν-calculus were provided in [29,30], using a representation of name creation via a strong monad. Two examples of such models were given: (i) the functor category Set^I with I the category of finite sets and injection; (ii) the category $\mathbf{B}G$ of continuous G-sets, with G the topological group of automorphisms over $\mathbb{N}$. None of these models are fully-abstract, since they distinguish $\texttt{new}\ n\ \texttt{in}\ \lambda x.x = n$ from $\lambda x.\texttt{false}$.

These models were later refined using nominal sets [7], so that types are interpreted via *Fraenkel-Mostowski* sets [28] or domains [14]. Both of these works are continuation models; they might be used to provide a semantics for the $\lambda\mu\nu$-calculus studied in this paper, a direction we wish to explore in future work. Such

use of continuations was justified in [28] to provide a model for an extension of the ν-calculus with recursion. More recently, *proof-relevant* logical relations were introduced to deal with recursion in the presence of name generation [4].

In [26], a model of the ν-calculus is given in quasi-Borel spaces, showing a correspondence between random sampling and fresh name generation. This model is shown to be fully-abstract for terms of first-order types.

In [5], environmental bisimulations for the ν-calculus are defined and shown to be fully abstract. Nevertheless, it does not seem possible to extract a decision procedure from that result, since environmental bisimulations are played over a higher-order LTS, that is, an LTS whose actions contain λ-terms. So this LTS is infinitely branching at higher-order types.

Eager normal-form bisimulations have been introduced by Lassen for the call-by-value λ-calculus [16] and $\lambda\mu$-calculus. In [31], a notion of bisimulation similar to $\simeq_{\mathsf{ogs}}$ is introduced and shown to be fully abstract for an untyped version of $\lambda\mu_{\mathtt{ref}}$. Compared to the standard notion of eager normal form bisimulations, the configurations in the bisimulations in [31] contain an environment similar to the environment component γ of the OGS LTS in Section 3.

In [1], a fully-abstract game model is provided for the ν-calculus. However, this model requires an extensional collapse, that is not directly computable at higher-order type. So that model could only be used to prove the decidability of contextual equivalence for terms of first-order types. Enforcing a well-bracketed and visible behavior for Opponent in the OGS model, we believe that our trace model would coincide with the intentional game model of [1]. Nominal game semantics was developed for languages with nominal references and exceptions in [32]. In that setting, algorithmic presentations of game semantics make it possible to provide a classification of decidability of call-by-value languages with (bounded) integer references [19], and ground references [21]. In this setting, the undecidability of contextual equivalence originates from the use of integer references by Proponent. A detailed survey on the literature on contextual equivalence for the ν-calculus is available in [33].

7 Conclusion

To decide the contextual equivalence between two $\lambda\mu\nu$ typed terms `M` and `N` with contexts in the $\lambda\mu_{\mathtt{ref}}$-calculus, we first construct the corresponding initial configurations, and we can decide by Thm. 14 if they are POGS-bisimilar. This decidability result comes from the fact that the POGS LTS generates finite trees.

Then, we prove in Thm. 29 and Thm. 30 that two initial active configurations are POGS-bisimilar iff they are OGS-bisimilar. This is possible because initial configurations are prime (they are active and γ is empty) and are also limit configurations (their disclosed sets contain all the atoms of the store). In Thm. 7 and Lemma 10, we prove that `M` and `N` are contextually equivalent iff the corresponding initial configurations are OGS-bisimilar, which yields decidability.

We now examine the obstacles that remain to prove the decidability of contextual equivalence with contexts in the ν-calculus.

First of all, in that setting, trace equivalence would not be fully-abstract anymore (Thm. 7). Indeed, without integer references, one cannot observe the sequentiality of calls and returns. So an extensional collapse would be necessary.

Another obstacle is that in the absence of higher-order references, Opponent must satisfy a condition of *O-visibility* [2], that corresponds to a local well-scoping discipline, for the function names it is allowed to call. Working in an intuitionistic type system, corresponding to the standard λ-calculus without control operators, the call-and-return discipline of the interaction between Proponent and Opponent has to be *well-bracketed*. These two conditions, namely O-visibility and well-bracketing, can be enforced operationally [13] in the LTS, by keeping track of part of the history of the interaction. However the reduction of $\simeq_{\mathsf{ogs}}$ to $\simeq_{\mathsf{pogs}}$ is not possible anymore in that setting. Indeed, the limit over-approximates the set of atoms that can be tested. This can be seen when comparing the programs

$$\texttt{new}\ n\ \texttt{in}\ \mathsf{let}\ \_ = y(\lambda z.z = a)\ \mathsf{in}\ n \quad \text{and} \quad \texttt{new}\ n\ \texttt{in}\ \mathsf{let}\ \_ = y(\lambda z.\texttt{false})\ \mathsf{in}\ n$$

Assuming n is immediately disclosed makes it possible to distinguish the two programs. Because the local conditions of well-bracketing or visibility would prevent Opponent from playing some actions, Opponent could perform irreversible changes that would invalidate Lemma 25. This would make $\simeq_{\mathsf{pogs}}$ incomplete.

To handle this difficulty, we could try and use Kripke eager normal-form bisimulation [11], using a structure for worlds richer than just a set of atoms.

Finally, in absence of *full ground references*, that can store locations, atoms played by Opponent would also follow a local well-scoping discipline, but the discriminatory power over Player atoms would also be restricted [20]. In such a setting, the same difficulties as with well-bracketing and O-visibility would arise, and a more complex extensional collapse would be needed.

References

1. Samson Abramsky, Dan R. Ghica, Andrzej S. Murawski, C.-H. Luke Ong, and Ian David Bede Stark. Nominal games and full abstraction for the nu-calculus. In *19th IEEE Symposium on Logic in Computer Science (LICS 2004), 14-17 July 2004, Turku, Finland, Proceedings*, pages 150–159. IEEE Computer Society, 2004.
2. Samson Abramsky, Kohei Honda, and Guy McCusker. A fully abstract game semantics for general references. In *Thirteenth Annual IEEE Symposium on Logic in Computer Science, Indianapolis, Indiana, USA, June 21-24, 1998*, pages 334–344. IEEE Computer Society, 1998.
3. Hendrik Pieter Barendregt. *The lambda calculus - its syntax and semantics*, volume 103 of *Studies in logic and the foundations of mathematics*. North-Holland, 1985.
4. Nick Benton, Martin Hofmann, and Vivek Nigam. Proof-relevant logical relations for name generation. *Log. Methods Comput. Sci.*, 14(1), 2018.
5. Nick Benton and Vasileios Koutavas. A mechanized bisimulation for the nu-calculus. *Higher Order and Symbolic Computation - Special Issue in Honor of Mitchell Wand*, sep 2012. In Press.

6. Corrado Böhm. Alcune proprieta delle forme β-η-normali nel λ-k-calcolo. *Pubblicazioni dell'Istituto per le Applicazioni del Calcolo*, 696:19, 1968.
7. Murdoch Gabbay and Andrew M. Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects Comput.*, 13(3-5):341–363, 2002.
8. Jean-Yves Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987.
9. Daniel Hirschkoff, Guilhem Jaber, and Enguerrand Prebet. Deciding contextual equivalence of ν-calculus with effectful contexts (full version). https://hal.science/hal-03955303.
10. Guilhem Jaber and Andrzej S. Murawski. Complete trace models of state and control. In Nobuko Yoshida, editor, *Programming Languages and Systems - 30th European Symposium on Programming, ESOP 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings*, volume 12648 of *Lecture Notes in Computer Science*, pages 348–374. Springer, 2021.
11. Guilhem Jaber and Andrzej S. Murawski. Compositional relational reasoning via operational game semantics. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–13. IEEE, 2021.
12. Guilhem Jaber and Davide Sangiorgi. Games, mobile processes, and functions. In Florin Manea and Alex Simpson, editors, *30th EACSL Annual Conference on Computer Science Logic, CSL 2022, February 14-19, 2022, Göttingen, Germany (Virtual Conference)*, volume 216 of *LIPIcs*, pages 25:1–25:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.
13. James Laird. A fully abstract trace semantics for general references. In Lars Arge, Christian Cachin, Tomasz Jurdzinski, and Andrzej Tarlecki, editors, *Automata, Languages and Programming, 34th International Colloquium, ICALP 2007, Wroclaw, Poland, July 9-13, 2007, Proceedings*, volume 4596 of *Lecture Notes in Computer Science*, pages 667–679. Springer, 2007.
14. James Laird. Sequentiality and the CPS semantics of fresh names. In Marcelo Fiore, editor, *Proceedings of the 23rd Conference on the Mathematical Foundations of Programming Semantics, MFPS 2007, New Orleans, LA, USA, April 11-14, 2007*, volume 173 of *Electronic Notes in Theoretical Computer Science*, pages 203–219. Elsevier, 2007.
15. James Laird. A Curry-style semantics of interaction: From untyped to second-order lazy λ μ-calculus. In Jean Goubault-Larrecq and Barbara König, editors, *Foundations of Software Science and Computation Structures - 23rd International Conference, FOSSACS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings*, volume 12077 of *Lecture Notes in Computer Science*, pages 422–441. Springer, 2020.
16. 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. IEEE Computer Society, 2005.
17. Søren B. Lassen and Paul Blain Levy. Typed normal form bisimulation. In Jacques Duparc and Thomas A. Henzinger, editors, *Computer Science Logic, 21st International Workshop, CSL 2007, 16th Annual Conference of the EACSL, Lausanne, Switzerland, September 11-15, 2007, Proceedings*, volume 4646 of *Lecture Notes in Computer Science*, pages 283–297. Springer, 2007.
18. Sam Lindley and Ian Stark. Reducibility and tt-lifting for computation types. In Pawel Urzyczyn, editor, *Typed Lambda Calculi and Applications, 7th International*

Conference, TLCA 2005, Nara, Japan, April 21-23, 2005, Proceedings, volume 3461 of *Lecture Notes in Computer Science*, pages 262–277. Springer, 2005.

19. Andrzej S. Murawski and Nikos Tzevelekos. Algorithmic nominal game semantics. In Gilles Barthe, editor, *Programming Languages and Systems - 20th European Symposium on Programming, ESOP 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*, volume 6602 of *Lecture Notes in Computer Science*, pages 419–438. Springer, 2011.
20. Andrzej S. Murawski and Nikos Tzevelekos. Full abstraction for reduced ML. *Ann. Pure Appl. Log.*, 164(11):1118–1143, 2013.
21. Andrzej S. Murawski and Nikos Tzevelekos. Algorithmic games for full ground references. *Formal Methods Syst. Des.*, 52(3):277–314, 2018.
22. Michel Parigot. Lambda-mu-calculus: An algorithmic interpretation of classical natural deduction. In Andrei Voronkov, editor, *Logic Programming and Automated Reasoning,International Conference LPAR'92, St. Petersburg, Russia, July 15-20, 1992, Proceedings*, volume 624 of *Lecture Notes in Computer Science*, pages 190–201. Springer, 1992.
23. Michel Parigot. Strong normalization for second order classical natural deduction. In *Proceedings of the Eighth Annual Symposium on Logic in Computer Science (LICS '93), Montreal, Canada, June 19-23, 1993*, pages 39–46. IEEE Computer Society, 1993.
24. Andrew M. Pitts and Ian David Bede Stark. Observable properties of higher order functions that dynamically create local names, or what's new? In Andrzej M. Borzyszkowski and Stefan Sokolowski, editors, *Mathematical Foundations of Computer Science 1993, 18th International Symposium, MFCS'93, Gdansk, Poland, August 30 - September 3, 1993, Proceedings*, volume 711 of *Lecture Notes in Computer Science*, pages 122–141. Springer, 1993.
25. D. Pous and D. Sangiorgi. *Advanced Topics in Bisimulation and Coinduction (D. Sangiorgi and J. Rutten editors)*, chapter Enhancements of the coinductive proof method. Cambridge University Press, 2011.
26. Marcin Sabok, Sam Staton, Dario Stein, and Michael Wolman. Probabilistic programming semantics for name generation. *Proc. ACM Program. Lang.*, 5(POPL):1–29, 2021.
27. Davide Sangiorgi. On the bisimulation proof method. *Mathematical Structures in Computer Science*, 8(5):447–479, 1998.
28. Mark R. Shinwell and Andrew M. Pitts. On a monadic semantics for freshness. *Theor. Comput. Sci.*, 342(1):28–55, 2005.
29. Ian Stark. *Names and Higher-Order Functions.* PhD thesis, University of Cambridge, December 1994. Also available as Technical Report 363, University of Cambridge Computer Laboratory.
30. Ian Stark. Categorical models for local names. *LISP Symb. Comput.*, 9(1):77–107, 1996.
31. Kristian Støvring and Søren B. Lassen. A complete, co-inductive syntactic theory of sequential control and state. In Jens Palsberg, editor, *Semantics and Algebraic Specification, Essays Dedicated to Peter D. Mosses on the Occasion of His 60th Birthday*, volume 5700 of *Lecture Notes in Computer Science*, pages 329–375. Springer, 2009.
32. Nikos Tzevelekos. *Nominal Game Semantics.* PhD thesis, University of Oxford, 2009.

33. Nikos Tzevelekos. Program equivalence with names. In Amal Ahmed, Nick Benton, Lars Birkedal, and Martin Hofmann, editors, *Modelling, Controlling and Reasoning About State, 29.08. - 03.09.2010*, volume 10351 of *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany, 2010.