Concurrent Visibility: Higher-Order Concurrency with First-Order Store

Iwan Quémerais
ENS de Lyon & University of Bologna, Italy

Guilhème Jaber
Nantes Université, France

Ken Sakayori
The University of Tokyo, Japan

Davide Sangiorgi
University of Bologna & INRIA, Italy

Abstract

We propose an Operational Game Semantics for a (call-by-value) concurrent higher-order language with first-order store (references may contain other references or first-order values such as integers or booleans; however they may not store higher-order values such as functions). We adapt the game-semantic notion of *visibility*, which semantically captures the absence of higher-order references and developed for sequential higher-order languages, to a concurrent setting. We thus define a *complete-trace* preorder, and prove it sound for the contextual preorder, by introducing a synchronization-based composition of semantic configurations and establishing an observational adequacy result. We also prove completeness for the subset of the language in which functions return first-order values.

In contrast to the case of sequential visibility, in the labeled transition semantics we have to account for the presence of multiple active threads with possibly different visibilities, and of a tree-like structure for managing the dependencies among the threads so created. Moreover, we have to reason on families of traces, rather than single traces, as in concurrent setting the order among certain actions cannot be enforced.

2012 ACM Subject Classification Theory of computation → Program semantics

Keywords and phrases Operational game semantics, higher-order effectful programs, mutable store

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2026.45

Funding *Ken Sakayori*: JSPS KAKENHI Grant Number JP24K20731.

1 Introduction

A central question in programming language semantics is how the presence or absence of a given program construct, such as mutable store, control operators, parallelism, or nondeterministic choice, is reflected in the semantics. From the point of view of *contextual equivalence*, this question can be approached by asking whether adding a construct gives contexts more power to distinguish programs. For instance, in sequential higher-order languages, local store allows a context to distinguish $\lambda f.\, f()$ from $\lambda f.\, f(); f()$, by counting how many times f is called.

In interactive models of computation, programs are described through their possible interactions with the environment, such as the calls to f in the example above. The absence of a program construct can then often be characterized semantically as a restriction on the allowed interactions. *Game semantics* [13, 1], which emerged and grew in the realm of sequential higher-order languages, has provided several striking examples: the absence of non-local control operators corresponds to *well-bracketing* [22], the absence of higher-order store to *visibility* [2], and stronger restrictions on state access give rise to conditions

such as *innocence* [13]. In operational presentations of game semantics [15, 24], based on labelled transition systems, such conditions are typically enforced by enriching the states with additional components recording suitable information about past interactions.

When contexts have access to parallelism, they can distinguish more programs than in the sequential setting. Consider the following example:

▶ **Example 1.**

$\mathtt{M}_1 \triangleq \mathtt{let}\ x = \mathtt{ref}\ \mathtt{true}\ \mathtt{in}\ f\,(\lambda\_.x \coloneqq \mathtt{false}); \mathtt{false}$

$\mathtt{M}_2 \triangleq \mathtt{let}\ x = \mathtt{ref}\ \mathtt{true}\ \mathtt{in}\ f\,(\lambda\_.x \coloneqq \mathtt{false}); !x\ \&\&\ \mathtt{not}\,!x$

Both terms first create a boolean reference stored in x, before calling a higher-order function f with a setter function as an argument. In a sequential setting, the two terms are contextually equivalent, since the expression $!x$ && $\mathtt{not}\,!x$ always evaluates to `false`: the value stored in the reference x may not be changed between the two reads of x. In contrast, in a language combining parallelism with higher-order references, the following context `E` and instance of f distinguishes $\mathtt{M}_1$ and $\mathtt{M}_2$, by storing the setter function $\lambda\_.x \coloneqq \mathtt{false}$ passed to f in the higher-order reference ℓ, and invoking it in a separate thread; this invocation can be performed between the two reads performed by the program.

$\mathtt{E} \triangleq \mathsf{par}\ (\mathsf{let}\, x = \bullet\, \mathsf{in}\, \mathtt{if}\ x\ \mathtt{then}\ ()\ \mathtt{else}\ \Omega)\ \mathsf{and}\ !\ell()$ $\qquad f \triangleq \lambda x.\ell \coloneqq x$

Thus $\mathtt{E}[\mathtt{M}_1]$ always diverges while $\mathtt{E}[\mathtt{M}_2]$ can terminate. Higher-order storage is essential here: once the call to f has returned, the setter is no longer in scope and would not otherwise be available. This suggests that, even in the presence of concurrency, the terms $\mathtt{M}_1$ and $\mathtt{M}_2$ should remain equivalent provided that higher-order references are unavailable. The question is then how to capture this semantically.

As a language, we adopt a call-by-value higher-order language with parallel composition and full ground references (full ground meaning that references may contain other references or first-order values such as integers or booleans; however they may not contain higher-order values such as functions). We then adapt the game-semantic notion of *visibility* to a concurrent setting. For this we first define an *interactive labelled transition system* (LTS), expressing possible interaction between a term and its environment. Following the tradition of operational game semantics as well as of higher-order concurrency, the action of this LTS may not contain higher-order values; higher-order values are replaced by abstract values, technically, names, that provide access to the actual values. In this setting, the *visibility* of a thread is intuitively the set of names of functions that the thread is allowed to call (i.e., the functions that are in its scope). We then couple this interactive LTS with two further components: a *typing LTS*, controlling the introduction of fresh names together with their types; a *visibility LTS*, tracking the possibly different visibilities of the current active threads. The use of the visibility LTS allows us to take into account the absence of higher-order references in the model. Such LTS also records the call-return well-bracketed discipline of interaction among terms of the language, reflecting the fact that the language has no non-local control operators. In the sequential case, well-bracketing is managed using a stack; in our concurrent case, we have to adopt a tree-like structure, recording dependencies among the threads created.

On top of the composite LTS, we define a *complete-trace* preorder (a complete trace intuitively representing a "completed" sequence of interactions between a term and its environment). Our first main result is soundness of the complete-trace preorder with respect to the contextual preorder. This is obtained by introducing a synchronization-based composition of semantic configurations and establishing an observational adequacy result.

Our second main result is completeness for the subset of the language in which functions return first-order values. The usual strategy of completeness proofs is to consider two terms in the context and a complete trace for one of them. Then one constructs a context that produces the complement trace, followed by a success signal. From the contextual relation between the two terms one then concludes that the second term also has that trace. The difficulty in our concurrent setting is that the order among certain actions of a trace (produced by a context or a term) cannot be enforced. Indeed, certain consecutive actions of traces may always be commuted. Thus we have to reason about families of traces, rather than single traces. This complicates the proof significantly, and brings in the requirement, discussed in the paper, that functions return first-order values.

In this paper, we do not consider the possibility for the program and its environment to share a store. While game semantics provides a way to represent such shared store by enriching the actions of the LTS [24, 29], this complexifies the models in a direction that is mostly orthogonal to the treatment of visibility.

2 Target Language and Operational Semantics

We work with a call-by-value λ-calculus extended with references and parallel compositions. A reference may contain references to other references – allowing programmers to write cyclic data structures – but may not store functions; such references are called *full ground* [30]. Types, values, terms and (sequential) evaluation contexts are defined by the following grammars:

$$
\begin{array}{rrl}
\text{Ground types} & \iota & ::= \mathtt{Unit} \mid \mathtt{Bool} \mid \mathtt{Int} \\
\text{Reference types} & \rho & ::= \mathbf{ref}_{\iota} \mid \mathbf{ref}_{\rho} \\
\text{Types} & \sigma, \tau & ::= \iota \mid \rho \mid \sigma \times \tau \mid \sigma \to \tau \\
\text{Values} & \mathtt{V}, \mathtt{W} & ::= x \mid \ell \mid () \mid \mathtt{b} \mid \mathtt{k} \mid \langle \mathtt{V}, \mathtt{W} \rangle \mid \lambda x.\mathtt{M} \\
\text{Terms} & \mathtt{M}, \mathtt{N} & ::= \mathtt{V} \mid \mathtt{V}\,\mathtt{M} \mid \mathsf{proj}_i\,\mathtt{V} \mid \mathtt{if\ V\ then\ M\ else\ N} \mid \mathtt{rec}\, f(x).\mathtt{M} \\
 & & \quad \mid \mathtt{ref\ V} \mid \mathtt{!V} \mid \mathtt{V} := \mathtt{W} \mid \mathsf{par}\ \mathtt{M}\ \mathsf{and}\ \mathtt{N} \mid \mathtt{CAS\ V\ V\ V} \\
\text{Sequential contexts} & \mathtt{E} & ::= \bullet \mid \mathtt{V\,E} \\
\text{Evaluation contexts} & \mathtt{D} & ::= \bullet \mid \mathtt{V\,D} \mid \mathsf{par}\ \mathtt{D}\ \mathsf{and}\ \mathtt{M} \mid \mathsf{par}\ \mathtt{M}\ \mathsf{and}\ \mathtt{D}
\end{array}
$$

We let x range over variables, ℓ over locations, $\mathtt{b} \in \{\mathtt{true}, \mathtt{false}\}$, and $\mathtt{k}$ over integers. The term $\mathsf{par}\ \mathtt{M}\ \mathsf{and}\ \mathtt{N}$ is a "parallel pair" that executes the first and the second element in parallel. Such parallel pair construct can be found in [4], which was a source of inspiration for our language. A compare-and-swap operation $\mathtt{CAS}\ \ell\ \mathtt{V}_1\mathtt{V}_2$ *atomically* compares the value stored at the location ℓ with $\mathtt{V}_1$, and swaps it with $\mathtt{V}_2$ if they are the same. Other constructs, from λ-calculus with store, are standard. By restricting the place where we can have terms rather than values in the syntax, we force the order of evaluation to be fully explicit, as is done with administrative normal forms [31]. Doing so, sequential evaluation contexts $\mathtt{E}$ and parallel ones $\mathtt{D}$, used in defining the operational semantics and the OGS, have a particularly simple definition. As a syntax sugar, we sometimes write: $\mathsf{let}\, x = \mathtt{M}\, \mathsf{in}\, \mathtt{N}$ to mean $(\lambda x.\mathtt{N})\,\mathtt{M}$; and $\mathtt{M\,N}$ (resp. $\langle \mathtt{M}, \mathtt{N} \rangle$) to mean $\mathsf{let}\, x = \mathtt{M}\, \mathsf{in}\, x\,\mathtt{N}$ (resp. $\mathsf{let}\, x = \mathtt{M}\, \mathsf{in}\, \mathsf{let}\, y = \mathtt{N}\, \mathsf{in}\, \langle x, y \rangle$).

The type judgment for terms is of the form $\Sigma; \Gamma \vdash \mathtt{M} : \sigma$, where Γ (resp. Σ) is a *typing context* (resp. *store typing context*), which is a finite map from variables (resp. locations) to types. Similarly, the type judgment for evaluation contexts is of the form $\Sigma; \Gamma \vdash \mathtt{D} : \tau \rightsquigarrow \sigma$, where τ represents the type of the hole and σ represents the type of the result. The typing rules are standard, see Appendix B.1

$$((\lambda x.M)\,\mathtt{V};\mathtt{S}) \mapsto_{\mathsf{red}} (M\{x := \mathtt{V}\};\mathtt{S})$$
$$(\mathtt{if\ true\ then\ M\ else\ N};\mathtt{S}) \mapsto_{\mathsf{red}} (\mathtt{M};\mathtt{S})$$
$$(\mathtt{if\ false\ then\ M\ else\ N};\mathtt{S}) \mapsto_{\mathsf{red}} (\mathtt{N};\mathtt{S})$$
$$(\mathtt{ref\ V};\mathtt{S}) \mapsto_{\mathsf{red}} (\ell;\mathtt{S}\cdot[\ell \mapsto \mathtt{V}]) \quad (\text{if } \ell \notin \mathrm{dom}(\mathtt{S}))$$
$$(!\ell;\mathtt{S}) \mapsto_{\mathsf{red}} (\mathtt{S}(\ell);\mathtt{S}) \quad (\text{if } \ell \in \mathrm{dom}(\mathtt{S}))$$
$$(\ell := \mathtt{V};\mathtt{S}) \mapsto_{\mathsf{red}} (();\mathtt{S}\cdot[\ell \mapsto \mathtt{V}])$$

$$(\mathsf{proj}_i\,\langle \mathtt{V}_1, \mathtt{V}_2\rangle;\mathtt{S}) \mapsto_{\mathsf{red}} (\mathtt{V}_i;\mathtt{S})$$
$$(\mathtt{CAS}\ \ell\ \mathtt{V}_1\ \mathtt{V}_2;\mathtt{S}\cdot[\ell \mapsto \mathtt{V}_1]) \mapsto_{\mathsf{red}} (\mathtt{true};\mathtt{S}\cdot[\ell \mapsto \mathtt{V}_2])$$
$$(\mathtt{CAS}\ \ell\ \mathtt{V}_1\ \mathtt{V}_2;\mathtt{S}\cdot[\ell \mapsto \mathtt{V}_3]) \mapsto_{\mathsf{red}} (\mathtt{false};\mathtt{S}\cdot[\ell \mapsto \mathtt{V}_3]) \quad (\text{if } \mathtt{V}_1 \neq \mathtt{V}_3)$$
$$(\mathsf{par}\ \mathtt{V}\ \mathsf{and}\ \mathtt{W};\mathtt{S}) \mapsto_{\mathsf{red}} (\langle \mathtt{V}, \mathtt{W}\rangle;\mathtt{S})$$
$$\frac{(\mathtt{M};\mathtt{S}) \mapsto_{\mathsf{red}} (\mathtt{M}';\mathtt{T})}{(\mathtt{D}[\mathtt{M}];\mathtt{S}) \mapsto_{\mathsf{red}} (\mathtt{D}[\mathtt{M}'];\mathtt{T})}$$

Figure 1 Reduction semantics.

We now define the main operational semantics and behavioural equivalence, namely reduction semantics and contextual equivalence. As standard for reduction relations of languages with references, we use *stores* (or heaps), as maps from locations to values. We write $[\ell \mapsto \mathtt{V}]$ for the singleton store mapping ℓ to $\mathtt{V}$; $\mathtt{S}\cdot\mathtt{T}$ for the disjoint concatenation of two stores; and $\mathtt{S}[\ell \mapsto \mathtt{V}]$ for the store that maps ℓ to $\mathtt{V}$ and $\ell' \neq \ell$ to $S(\ell')$. The judgment $\vdash \mathtt{S} : \Sigma$ denotes that, for each $\ell \in \mathrm{dom}(\mathtt{S})$, we have $\Sigma \vdash \mathtt{S}(\ell) : \Sigma(\ell)$. A (reduction) *configuration* is a pair of a term and a store written $(\mathtt{M};\mathtt{S})$.[1] We write $\Sigma;\Gamma \vdash (\mathtt{M};\mathtt{S}) : \sigma$ if $\vdash \Sigma : \mathtt{S}$ and $\Sigma;\Gamma \vdash \mathtt{M} : \sigma$. The reduction rules are shown in Figure 1. The left column shows standard operational rules for a sequential programming language with stores. Rules on the right column are related with concurrency. A parallel pair is treated as an ordinary pair when both components are values; otherwise, the two components may reduce in any order. We write $(\mathtt{M},\mathtt{S}) \Downarrow \mathtt{V}$ if $(\mathtt{M},\mathtt{S}) \mapsto^*_{\mathsf{red}} (\mathtt{V},\mathtt{S}')$ for some $\mathtt{S}'$.

As usual, contextual equivalence relates terms that have the same termination properties in all closing contexts. A (Γ/τ)-*testing-context* is a term with a hole that (1) accepts a term $\varnothing;\Gamma \vdash \mathtt{M} : \tau$, and (2) becomes a closed term of type `Unit` once the hole is filled.

▶ **Definition 2** (Contextual equivalence). *Suppose that* $\Sigma;\Gamma \vdash \mathtt{M}_1, \mathtt{M}_2 : \tau$. *The* contextual preorder $\Sigma;\Gamma \vdash \mathtt{M}_1 \preceq_{ctx} \mathtt{M}_2 : \tau$ *holds if, for all* (Γ/τ)*-testing-context* $\mathtt{C}$ *and* $\mathtt{S} : \Sigma$, $(\mathtt{C}[\mathtt{M}_1];\mathtt{S}) \Downarrow ()$ *implies* $(\mathtt{C}[\mathtt{M}_2];\mathtt{S}) \Downarrow ()$. *We write* $\Sigma;\Gamma \vdash \mathtt{M}_1 \simeq_{ctx} \mathtt{M}_2 : \tau$ *if* $\mathtt{M}_1$ *is below* $\mathtt{M}_2$ *and vice versa.*

A standard simplification of contextual equivalence is *closed instances of uses* (CIU) equivalence [28], which makes use of *evaluation contexts*, rather than general contexts. For an assignment γ from variables to values, we write $\Sigma \vdash \gamma : \Gamma$ if for any $x \in \mathrm{dom}(\Gamma)$, $\Sigma;\emptyset \vdash \gamma(x) : \Gamma(x)$. We write $\mathtt{M}\{\gamma\}$ for the term obtained by substituting in parallel in $\mathtt{M}$ all the $x \in \mathrm{dom}(\gamma)$ by $\gamma(x)$.

▶ **Definition 3.** *For two terms* $\Sigma;\Gamma \vdash \mathtt{M}_1, \mathtt{M}_2 : \tau$ *we say that* $\mathtt{M}_1$ *is a* CIU approximation *of* $\mathtt{M}_2$*, written* $\Sigma;\Gamma \vdash \mathtt{M}_1 \preceq_{ciu} \mathtt{M}_2 : \tau$ *(or simply* $\mathtt{M}_1 \preceq_{ciu} \mathtt{M}_2$*), if for all* Σ', $\mathtt{S}$, γ *and* $\mathtt{D}$, *such that*

- $\Sigma \subseteq \Sigma'$ *and* $\vdash \mathtt{S} : \Sigma'$*;*
- $\Sigma' \vdash \gamma : \Gamma$ *and* $\Sigma';\emptyset \vdash \mathtt{D} : \tau \rightsquigarrow \mathtt{Unit}$*;*

$(\mathtt{D}[\mathtt{M}_1\{\gamma\}];\mathtt{S}) \Downarrow ()$ *implies* $(\mathtt{D}[\mathtt{M}_2\{\gamma\}];\mathtt{S}) \Downarrow ()$. *If both* $\mathtt{M}_1 \preceq_{ciu} \mathtt{M}_2$ *and* $\mathtt{M}_2 \preceq_{ciu} \mathtt{M}_1$ *hold, then we say that* $\mathtt{M}_1$ *and* $\mathtt{M}_2$ *are* CIU equivalent*, written* $\Sigma;\Gamma \vdash \mathtt{M}_1 \simeq_{ciu} \mathtt{M}_2 : \tau$ *(or simply* $\mathtt{M}_1 \simeq_{ciu} \mathtt{M}_2$*).*

CIU preorder coincides with contextual preorder. The proof is done by showing that the CIU preorder is a congruence, following a standard technique established in [11].

[1] Several notions of configuration will be introduced in this paper, but we will usually just call them configuration when the meaning is clear from the context.

▶ **Proposition 4.** *We have* $\varnothing; \Gamma \vdash \mathtt{M}_1 \preceq_{ctx} \mathtt{M}_2 : \tau$ *if and only if* $\varnothing; \Gamma \vdash \mathtt{M}_1 \preceq_{ciu} \mathtt{M}_2 : \tau$.

In the following, we work only on interactions between a program and a context that do not share any locations. To preserve this property during the reduction, we enforce that the typing at their interface do not contain any **ref** type constructor. So we say that a term $\Sigma; \Gamma \vdash \mathtt{M} : \tau$ is *ref-free* if no reference type occurs (even as a subexpression) in Γ and τ.

3 Operational Game Semantics

3.1 LTSs

Contextual equivalence, as well as CIU equivalence, makes use of a heavy universal quantification on contexts. Our goal is to follow the ideas of Operational Game Semantics so to derive a Labeled Transition System (LTS) and, on top of this, a trace semantics, which can be used to reason about contextual equivalence. We will obtain this through a few steps.

1. First we define an LTS $\mathcal{L}_{\mathsf{env}}$ that describes how terms interact with the environment. The LTS does not account for reductions internal to terms, for which we still refer to the basic reduction semantics (it is common in operational game semantics to separate internal reductions from interactions with the environment).
2. In the LTS $\mathcal{L}_{\mathsf{env}}$, actions may carry higher-order values. It is well-known that such kinds of actions complicate definitions of satisfying behavioural relations (see e.g. [32, 33]). In OGS such higher-order values are transformed into abstract values, intuitively pointers to the actual values. This is accomplished in the second step, namely LTS $\mathcal{L}_{\mathsf{I}}$ in Figure 4. In $\mathcal{L}_{\mathsf{I}}$ we also structure the states of the LTS as a pool of threads; each thread, say $[\mathsf{q}]\mathtt{M}$, is a running term $\mathtt{M}$ that will compute a value to be returned to the environment at q.
3. The two aspects that still remain to be taken into account are typing and visibility; that is, we have to make sure that only well-typed actions are performed, and that actions respect the constraints produced by visibility (the impossibility for the environment of storing higher-order values). To account for these two aspects we introduce, respectively, the LTS $\mathcal{L}_{\mathsf{Ty}}$ (Figure 5), ensuring well-typedness, and the LTS $\mathcal{L}_{\mathsf{vis}}$ (Figure 6), that keeps track of the evolution of the visibility view of the threads in the environment.
4. The final LTS is the combination of the above three LTS: $\mathcal{L}_{\mathsf{I}}$, $\mathcal{L}_{\mathsf{Ty}}$, $\mathcal{L}_{\mathsf{vis}}$; that is, the legal actions are those produced by $\mathcal{L}_{\mathsf{I}}$ and allowed by the typing LTS $\mathcal{L}_{\mathsf{Ty}}$ and the visibility LTS $\mathcal{L}_{\mathsf{vis}}$.

Partially suspended terms and the LTS $\mathcal{L}_{\mathsf{env}}$

The LTS $\mathcal{L}_{\mathsf{env}}$ describes how terms interact with the environment. The states of the LTS are extensions of terms called *partially suspended terms*, as they may include terms that, having interrogated the environment, are now waiting for an answer:

$$\text{Partially suspended terms} \quad \underline{\mathtt{M}}, \underline{\mathtt{N}} ::= \mathtt{M} \mid \{\mathtt{E}\}_{\mathsf{p}} \mid \mathtt{E}[\mathsf{par}\ \underline{\mathtt{M}}\ \mathsf{and}\ \underline{\mathtt{N}}]$$

We call names such as p *continuation names*. Term $\{\mathtt{E}\}_{\mathsf{p}}$ is suspended waiting for the environment to provide a value at p. We write $\underline{\mathtt{M}}[(\mathtt{N}, \mathsf{p})]$ for the (partially suspended) term obtained by replacing the subterm of $\underline{\mathtt{M}}$ suspended on p, say, $\{\mathtt{E}\}_{\mathsf{p}}$, with $\mathtt{E}[\mathtt{N}]$. The transition rules are shown in Figure 2. A transition is of the form $\underline{\mathtt{M}} \overset{\mu}{\mapsto}_{\mathsf{env}} \underline{\mathtt{N}}$ where the action μ is either of the form $\mathsf{f}(\mathtt{V}, \mathsf{p})$ or $\mathsf{p}(\mathtt{V})$. We omit the straightforward extension of the reduction relation $\mapsto_{\mathsf{red}}$ to partially suspended terms, which will be used in the following LTSs.

$$\frac{\mathsf{p} \text{ is fresh}}{\mathtt{E}[f\ \mathtt{V}] \overset{f(\mathtt{V},\mathsf{p})}{\longmapsto}_{\mathsf{env}} \{\mathtt{E}\}_{\mathsf{p}}} \qquad \frac{}{\{\mathtt{E}\}_{\mathsf{p}} \overset{\mathsf{p}(\mathtt{V})}{\longmapsto}_{\mathsf{env}} \mathtt{E}[\mathtt{V}]} \qquad \frac{\underline{\mathtt{M}} \overset{\mu}{\longmapsto}_{\mathsf{env}} \underline{\mathtt{M}}'}{\mathsf{par}\ \underline{\mathtt{M}}\ \mathsf{and}\ \underline{\mathtt{N}} \overset{\mu}{\longmapsto}_{\mathsf{env}} \mathsf{par}\ \underline{\mathtt{M}}'\ \mathsf{and}\ \underline{\mathtt{N}}}$$

$$\frac{\underline{\mathtt{N}} \overset{\mu}{\longmapsto}_{\mathsf{env}} \underline{\mathtt{N}}'}{\mathsf{par}\ \underline{\mathtt{M}}\ \mathsf{and}\ \underline{\mathtt{N}} \overset{\mu}{\longmapsto}_{\mathsf{env}} \mathsf{par}\ \underline{\mathtt{M}}\ \mathsf{and}\ \underline{\mathtt{N}}'} \qquad \frac{\mathsf{par}\ \underline{\mathtt{M}}_1\ \mathsf{and}\ \underline{\mathtt{M}}_2 \overset{\mu}{\longmapsto}_{\mathsf{env}} \underline{\mathtt{N}}}{\mathtt{E}[\mathsf{par}\ \underline{\mathtt{M}}_1\ \mathsf{and}\ \underline{\mathtt{M}}_2] \overset{\mu}{\longmapsto}_{\mathsf{env}} \mathtt{E}[\underline{\mathtt{N}}]}$$

Figure 2 Definition of $\mathcal{L}_{\mathsf{env}}$.

$$\frac{}{\mathsf{f} : \sigma \to \tau \Vdash \mathsf{f} : \sigma \to \tau} \qquad \frac{\mathtt{B} \text{ a base value of ground type } \iota}{\varnothing \Vdash \mathtt{B} : \iota}$$

$$\frac{\Delta_1 \Vdash \mathtt{A}_1 : \sigma_1 \qquad \Delta_2 \Vdash \mathtt{A}_2 : \sigma_2}{\Delta_1 \cdot \Delta_2 \Vdash \langle \mathtt{A}_1; \mathtt{A}_2 \rangle : \sigma_1 \times \sigma_2}$$

Figure 3 Definition of the type judgment for abstract values.

Abstract values and the LTS $\mathcal{L}_{\mathsf{I}}$

In OGS, values exchanged between a term (*Proponent*) and the an environment (*Opponent*) are abstracted away, by transforming them into *abstract values* (aka *ultimate patterns* [26]) $\mathtt{A}$ defined thus:

$$\mathtt{A} ::= \mathtt{B} \mid \mathsf{f} \mid \langle \mathtt{A}, \mathtt{A} \rangle$$

where $\mathsf{f} \in \mathrm{FNames}$ is a function name, and $\mathtt{B}$ is a base value. Each syntactic value $\mathtt{V}$ can be decomposed into an abstract value $\mathtt{A}$ (its positive skeleton) together with a corresponding name assignment map γ (its negative filling) such that $\gamma(\mathtt{A}) = \mathtt{V}$. Typing rules of abstract values are reported in Figure 3. Abstract values are exchanged between Proponent and Opponent through *moves* $\mathbf{m}$ of the following shape:

P-question	P-answer	O-question	O-answer
$\overline{\mathsf{f}}(\mathtt{A}, \mathsf{p})_{\mathsf{q}}$	$\overline{\mathsf{p}}(\mathtt{A})$	$\mathsf{f}(\mathtt{A}, \mathsf{p})_{\mathsf{q}}$	$\mathsf{p}(\mathtt{A})$

Such moves are the labels of the *Interactive LTS* $\mathcal{L}_{\mathsf{I}}$, in Figure 4. Question moves are indexed with continuation names, which is not standard in OGS. The indices resemble thread identifiers sometimes used in labelled transition semantics [19, 20], allowing us to identify the thread from which an action originates. They are introduced for technical convenience to define well-bracketing, but are treated as unobservable and removed when considering traces.

The states of the LTS are *interactive configurations* $\mathbb{I}, \mathbb{J}$ which are of the shape $\langle \mathtt{P}; \mathtt{S}; \gamma \rangle$, with $\mathtt{P}$ a *pool of threads*, $\mathtt{S}$ a store, and γ an assignment from names in FNames to either variables or λ-abstractions. A pool is a parallel composition of threads, of the form $[\mathsf{p}_1]\underline{\mathtt{M}}_1 | \cdots | [\mathsf{p}_n]\underline{\mathtt{M}}_n$. Thus there are two levels of parallelism: the pool runs multiple threads in parallel, and each thread may contain parallelism caused by the parallel-pair construct. A new thread is created when a OQ-transition is performed. This intuitively corresponds to invoking a closure in parallel with the other threads.

$$\frac{(\underline{\mathtt{M}};\mathtt{S}) \mapsto_{\mathsf{red}} (\underline{\mathtt{N}};\mathtt{T})}{\langle \mathtt{P}|[\mathtt{p}]\underline{\mathtt{M}}|\mathtt{Q};\mathtt{S};\gamma\rangle \to_{\mathsf{I}} \langle \mathtt{P}|[\mathtt{p}]\underline{\mathtt{N}}|\mathtt{Q};\mathtt{T};\gamma\rangle}$$

$$\mathrm{PQ}\ \frac{\mathtt{V} = \mathtt{A}\{\gamma'\} \qquad \underline{\mathtt{M}} \overset{\mathtt{f}(\mathtt{V},\mathtt{p})}{\longmapsto}_{\mathsf{env}} \underline{\mathtt{N}}}{\langle \mathtt{P}|[\mathtt{q}]\underline{\mathtt{M}}|\mathtt{Q};\mathtt{S};\gamma\rangle \xrightarrow{\bar{\mathtt{f}}(\mathtt{A},\mathtt{p})_{\mathtt{q}}}_{\mathsf{I}} \langle \mathtt{P}|[\mathtt{q}]\underline{\mathtt{N}}|\mathtt{Q};\mathtt{S};\gamma\cdot\gamma'\rangle} \qquad \frac{\mathtt{V} = \mathtt{A}\{\gamma'\}}{\langle \mathtt{P}|[\mathtt{p}]\mathtt{V}|\mathtt{Q};\mathtt{S};\gamma\rangle \xrightarrow{\overline{\mathtt{p}}(\mathtt{A})}_{\mathsf{I}} \langle \mathtt{P}|\mathtt{Q};\mathtt{S};\gamma\cdot\gamma'\rangle}\ \mathrm{PA}$$

$$\mathrm{OQ}\ \frac{}{\langle \mathtt{P};\mathtt{S};\gamma\rangle \xrightarrow{\mathtt{f}(\mathtt{A},\mathtt{p})_{\mathtt{q}}}_{\mathsf{I}} \langle \mathtt{P}|[\mathtt{p}](\gamma(\mathtt{f})\,\mathtt{A});\mathtt{S};\gamma\rangle} \qquad \frac{\underline{\mathtt{M}} \overset{\mathtt{p}(\mathtt{A})}{\longmapsto}_{\mathsf{env}} \underline{\mathtt{N}}}{\langle \mathtt{P}|[\mathtt{q}]\underline{\mathtt{M}};\mathtt{S};\gamma\rangle \xrightarrow{\mathtt{p}(\mathtt{A})}_{\mathsf{I}} \langle \mathtt{P}|[\mathtt{q}]\underline{\mathtt{N}};\mathtt{S};\gamma\rangle}\ \mathrm{OA}$$

Figure 4 Definition of $\mathcal{L}_{\mathsf{I}}$.

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

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

Figure 5 Definition of $\mathcal{L}_{\mathsf{Ty}}$.

The typing LTS $\mathcal{L}_{\mathsf{Ty}}$ and the visible LTS $\mathcal{L}_{\mathsf{vis}}$

The typing LTS $\mathcal{L}_{\mathsf{Ty}}$ tracks the type of new names introduced and ensures that everything remains well-typed. *Type-context configurations*, ranged over by $\mathbb{S}, \mathbb{T}$, are of the shape $\langle \Delta_{\mathsf{O}} \mid \Delta_{\mathsf{P}}\rangle$, where $\Delta_{\mathsf{O}}, \Delta_{\mathsf{P}}$ are disjoint typing contexts mapping function names to function types $\sigma \to \tau$, and continuation names to context types $\neg\sigma$. The typing LTS $\mathcal{L}_{\mathsf{Ty}}$, whose states are type-context configurations, is in Figure 5. Names in Δ_{O}, called Opponent names, are the names introduced by Opponent. Symmetrically, names in Δ_{P} are the one introduced by Proponent.

The *Visible* LTS $\mathcal{L}_{\mathsf{vis}}$, in Figure 6, allows us to manage the different visibilities of the Opponent names. States are configurations $\langle \mathcal{F}, \mathcal{G}, \mathcal{T}\rangle$, ranged over by $\mathbb{V}, \mathbb{W}$, where:

- $\mathcal{F}$: Names $\to \mathcal{P}_{\mathrm{fin}}$(Names) is a function from Opponent names to a set of Proponent names which tells us the visibility of each Opponent name (intuitively, which Proponent names were known by the Opponent name when it was introduced; or, in other words, which functions introduced by Proponent may be used when the Opponent name is called);
- $\mathcal{G}$: CNames $\to \mathcal{P}_{\mathrm{fin}}$(Names) is a function from Proponent continuation names, seen as thread identifiers, to a set of Proponent names, and tells us the visibility of each thread. The premise $\mathtt{f} \in \mathcal{G}(\mathtt{q})$ of the rule OQ is the key condition that forces the environment to respect visibility.
- $\mathcal{T}$: `Tree` is a tree that represents the "multi-threaded stack" of continuation names, and accounts for the "well-bracketed" structure of return-call interactions among terms. A well-bracketing discipline is common in higher-order languages without control operators. In sequential languages, the discipline is formalised by means of a stack of continuation

$$\text{PQ}\ \frac{}{\langle \mathcal{F};\mathcal{G};\mathcal{T}\rangle \xrightarrow{\overline{\mathtt{f}(\mathtt{A},\mathtt{p})_{\mathtt{q}}}}_{\mathsf{vis}} \langle \mathcal{F};\mathcal{G}\cdot[\mathsf{p}\mapsto \mathcal{F}(\mathsf{f})\cup \mathsf{supp}(\mathtt{A})];\mathtt{addLeaf}(\mathcal{T},\mathsf{p},\mathsf{q})\rangle}$$

$$\text{PA}\ \frac{\mathtt{removeLeaf}(\mathcal{T},\mathsf{p}) = \mathtt{Some}(\mathcal{T}',\mathsf{q}^?)}{\langle \mathcal{F};\mathcal{G};\mathcal{T}\rangle \xrightarrow{\overline{\mathsf{p}(\mathtt{A})}}_{\mathsf{vis}} \langle \mathcal{F};\mathcal{G}[\mathsf{q}^?\mapsto \mathcal{G}(\mathsf{q})\cup\mathsf{supp}(\mathtt{A})];\mathcal{T}'\rangle}$$

$$\text{OQ}\ \frac{\mathsf{f}\in\mathcal{G}(\mathsf{q})}{\langle \mathcal{F};\mathcal{G};\mathcal{T}\rangle \xrightarrow{\mathsf{f}(\mathtt{A},\mathsf{p})_{\mathsf{q}}}_{\mathsf{vis}} \langle \mathcal{F}\cdot\left(\prod_{\mathsf{g}\in\mathsf{supp}(\mathtt{A})}[\mathsf{g}\mapsto\mathcal{G}(\mathsf{q})]\right);\mathcal{G};\mathtt{addLeaf}(\mathcal{T},\mathsf{p},\mathsf{q})\rangle}$$

$$\text{OA}\ \frac{\mathtt{removeLeaf}(\mathcal{T},\mathsf{p}) = \mathtt{Some}(\mathcal{T}',\mathsf{q}^?)}{\langle \mathcal{F};\mathcal{G}\cdot[\mathsf{p}\mapsto\mathcal{V}];\mathcal{T}\rangle \xrightarrow{\mathsf{p}(\mathtt{A})}_{\mathsf{vis}} \langle \mathcal{F}\cdot\prod_{\mathsf{f}\in\mathsf{supp}(\mathtt{A})}[\mathsf{f}\mapsto\mathcal{V}];\mathcal{G};\mathcal{T}'\rangle}$$

Figure 6 Definition of $\mathcal{L}_{\mathsf{vis}}$.

names, and operations for handling them. Since our language is concurrent, we need tree-shaped stacks, with operations for dealing with such trees. The grammar for the trees is:

$$\mathtt{Tree} \triangleq \mathtt{Tree\_aux} \mid \varepsilon \qquad \mathtt{Tree\_aux} \triangleq \mathrm{CNames} \times (\mathtt{list}\ \mathtt{Tree\_aux})$$

We use two functions to manipulate trees:

- $\mathtt{addLeaf}(\mathcal{T},\mathsf{p},\mathsf{q})$ that adds p to the tree $\mathcal{T}$ as a leaf with parent q;
- $\mathtt{removeLeaf}(\mathcal{T},\mathsf{p})$ that returns an option type; intuitively, used to remove the leaf p from $\mathcal{T}$ and obtain p's parent. Formally:
  - if p is a leaf in $\mathcal{T}$, the function returns $\mathtt{Some}(\mathcal{T}',\mathsf{q}^?)$ where $\mathcal{T}'$ is tree with p removed; and $\mathsf{q}^? = \mathtt{None}$ if p is the root of $\mathcal{T}$, otherwise $\mathsf{q}^? = \mathtt{Some}(\mathsf{q})$ where q is p's parent;
  - if p is not a leaf the function returns $\mathtt{None}$.

The OGS LTS $\mathcal{L}_{\mathsf{OGS}}$

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 $(\mathbb{I},\mathbb{S},\mathbb{V})$, and its transition relation, the conjunction of the transitions of the three previous LTSs, is defined by the rules:

$$\frac{\mathbb{I}\xrightarrow{\mathbf{a}}_{\mathsf{I}}\mathbb{J} \qquad \mathbb{S}\xrightarrow{\mathbf{a}}_{\mathsf{Ty}}\mathbb{T} \qquad \mathbb{V}\xrightarrow{\mathbf{a}}_{\mathsf{vis}}\mathbb{W}}{(\mathbb{I},\mathbb{S},\mathbb{V})\xrightarrow{\mathbf{a}}_{\mathsf{ogs}}(\mathbb{J},\mathbb{T},\mathbb{W})} \qquad \frac{\mathbb{I}\rightarrow_{\mathsf{I}}\mathbb{J}}{(\mathbb{I},\mathbb{S},\mathbb{V})\rightarrow_{\mathsf{ogs}}(\mathbb{J},\mathbb{S},\mathbb{V})}$$

An OGS configuration is *valid* when all its components are consistent with each other, including types and structure of the tree-stack of continuation names with respect to the pool of threads; see Appendix C, where we also show that validity is preserved by transition.

$$\text{PI}\ \frac{\Gamma = \overrightarrow{(x_i : \tau_i)} \qquad \Delta_i \Vdash \mathtt{A}_i : \tau_i}{\langle \Gamma \vdash (\mathtt{M}, \mathtt{S}) : \tau \rangle \xrightarrow{?(\overrightarrow{\mathtt{A}_i}, \mathtt{p})}_{\mathsf{ogs}} \left(\begin{array}{l} \langle [\mathtt{p}]\mathtt{M}\{\overrightarrow{\mathtt{A}_i/x_i}\}; \mathtt{S}; \varepsilon \rangle; \\ \langle \overrightarrow{\Delta_i}, \mathtt{p} : \neg\tau | \varepsilon \rangle; \\ \langle \prod_{\mathtt{f} \in \mathsf{supp}(\overrightarrow{\mathtt{A}_i})} [\mathtt{f} \mapsto \emptyset]; \varepsilon; (\mathtt{p}, []) \rangle \end{array} \right)}$$

$$\text{OI}\ \frac{\Gamma = \overrightarrow{(x_i : \tau_i)} \qquad \Delta_i \Vdash \mathtt{A}_i : \tau_i \qquad \mathtt{A}_i\{\gamma_i\} = \delta(x_i) \qquad \mathtt{D} \xrightarrow{?(\mathtt{p})}_{\mathsf{env}} \underline{\mathtt{M}}}{\langle \vdash (\mathtt{D}, \delta, \mathtt{S}) : (\tau \rightsquigarrow \mathtt{Unit}; \Gamma) \rangle \xrightarrow{!(\overrightarrow{\mathtt{A}_i}, \mathtt{p})}_{\mathsf{ogs}} \left(\begin{array}{l} \langle [\mathtt{p}_f]\underline{\mathtt{M}}; \mathtt{S}; \overrightarrow{\gamma_i} \rangle; \\ \langle \mathtt{p}_f : \neg\mathtt{Unit} | \overrightarrow{\Delta_i}, p : \neg\tau \rangle; \\ \langle \varepsilon; [\mathtt{p} \mapsto \mathsf{supp}(\overrightarrow{\mathtt{A}_i})]; (\mathtt{p}_f, [(\mathtt{p}, [])]) \rangle \end{array} \right)}$$

Figure 7 Initial moves for $\mathcal{L}_{\mathsf{OGS}}$.

▶ **Example 5** (Example of transitions). From $\mathtt{V} \triangleq \lambda\mathtt{f}.\mathtt{g}(\lambda\_.\mathtt{f}())$, we have the following execution, omitting the store component of interactive configurations and the typing configurations:

$$\begin{array}{lll} \langle [\mathtt{p}_0]\mathtt{V}; \varepsilon \rangle; & \xrightarrow{\overline{\mathtt{p}_0}(\mathtt{h})}_{\mathsf{ogs}} & \langle \varepsilon; \overbrace{[\mathtt{h} \mapsto \mathtt{V}]}^{\gamma_1} \rangle; \\ \langle \mathcal{F}_0; \mathcal{G}_0; (\mathtt{q}_0, [(\mathtt{p}_0, [])]) \rangle & & \langle \mathcal{F}_0; \underbrace{[\mathtt{q}_0 \mapsto \{\mathtt{h}\}]}_{\mathcal{G}_1}; (\mathtt{q}_0, []) \rangle \\ & & \langle [\mathtt{p}_1]\mathtt{V}\ \mathtt{f}; \gamma_1 \rangle \\ & \xrightarrow{\mathtt{h}(\mathtt{f};\mathtt{p}_1)_{\mathtt{q}_0}}_{\mathsf{ogs}} & \langle \underbrace{\mathcal{F}_0 \cdot [\mathtt{f} \mapsto \{\mathtt{h}\}]}_{\mathcal{F}_1}; \mathcal{G}_1; (\mathtt{q}_0, [(\mathtt{p}_1, [])]) \rangle \text{ since } \mathtt{h} \in \mathcal{G}_1(\mathtt{q}_0) \\ & & \langle [\mathtt{p}_1]\mathtt{g}\ (\lambda\_.\mathtt{f}()); \gamma_1 \rangle \\ & \rightarrow_{\mathsf{ogs}} & \langle \mathcal{F}_1; \mathcal{G}_1; (\mathtt{q}_0, [(\mathtt{p}_1, [])]) \rangle \\ & & \langle [\mathtt{p}_1]\{\bullet\}_{\mathtt{q}_1}; \overbrace{\gamma_1 \cdot [v \mapsto \lambda\_.\mathtt{f}()]}^{\gamma_2} \rangle \\ & \xrightarrow{\overline{\mathtt{g}}(v;\mathtt{q}_1)_{\mathtt{p}_1}}_{\mathsf{ogs}} & \langle \mathcal{F}_1; \mathcal{G}_1 \cdot [\mathtt{q}_1 \mapsto \{v\}]; (\mathtt{q}_0, [(\mathtt{p}_1, [(\mathtt{q}_1, [])])]) \rangle \end{array}$$

with $\mathcal{F}_0 \triangleq [\mathtt{g}, \mathtt{p}_0 \mapsto \emptyset]$ and $\mathcal{G}_0 \triangleq [\mathtt{q}_0 \mapsto \emptyset]$.

3.2 Complete-trace preorder

In order to define OGS-traces of terms, we have also to add initial moves, so to transform typed terms into appropriate configurations of the $\mathcal{L}_{\mathsf{OGS}}$ LTS. This is done in the rules of Figure 7. In rule PI, $\mathtt{A}_i$'s are the abstract values that replace the free variables of the initial term; $\mathtt{p}$ is the name along which the final return value will be delivered; the tree is initialised with just $\mathtt{p}$ as a root; the visibility components are initially empty. Rule OI is similar, for environment used in the ciu preorder, and will be useful in the soundness proofs. To transform a parallel evaluation context $\mathtt{D}$ into a suspended term, we add to the LTS $\mathcal{L}_{\mathsf{env}}$ the rule $\mathtt{E} \xrightarrow{?(\mathtt{p})}_{\mathsf{env}} \{\mathtt{E}\}_{\mathtt{p}}$, so that we can suspend the maximal sequential evaluation context $\mathtt{E}$ appearing around the hole of a parallel context. This rule also uses a distinguished continuation name $\mathtt{p}_f$ used by Opponent to perform the final answer.

While defining traces, we erase the thread labels as they are not supposed to be observed; for instance $\mathtt{f}(\mathtt{A}, \mathtt{p})_{\mathtt{q}}$ becomes $\mathtt{f}(\mathtt{A}, \mathtt{p})$ and $\overline{\mathtt{f}}(\mathtt{A}, \mathtt{p})_{\mathtt{q}}$ becomes $\overline{\mathtt{f}}(\mathtt{A}, \mathtt{p})$. Weak transitions are defined in the expected manner. Thus $\Longrightarrow_{\mathsf{ogs}}$ is the reflexive and transitive closure of

$\rightarrow_{\mathsf{ogs}}$; and $\overset{\mathbf{a}}{\Longrightarrow}_{\mathsf{ogs}} \triangleq \Longrightarrow_{\mathsf{ogs}} \overset{\mathbf{a}}{\rightarrow}_{\mathsf{ogs}} \Longrightarrow_{\mathsf{ogs}}$; finally for a trace $\mathbf{t} = \mathbf{a}_1 \dots \mathbf{a}_n$, we write $\overset{\mathbf{t}}{\Longrightarrow}_{\mathsf{ogs}}$ for $\overset{\mathbf{a}_1}{\Longrightarrow}_{\mathsf{ogs}} \cdots \overset{\mathbf{a}_n}{\Longrightarrow}_{\mathsf{ogs}}$. A *complete trace* is a trace in which "all questions have been answered"; formally, at the end of the trace, the tree-stack should be empty.

▶ **Definition 6.** *A complete trace* $\mathbf{t}$ *of* $\langle \Gamma \vdash (\mathtt{M}, \mathtt{S}) : \tau \rangle$ *is a trace of* $(\mathtt{M}, \mathtt{S})$ *such that there exists* $\mathbb{I}, \mathbb{S}, \mathcal{F}, \mathcal{G}$ *such that* $\langle \Gamma \vdash (\mathtt{M}, \mathtt{S}) : \tau \rangle \overset{\mathbf{t}}{\Longrightarrow}_{\mathsf{ogs}} (\mathbb{I}; \mathbb{S}; \langle \mathcal{F}, \mathcal{G}, \varepsilon \rangle)$.
We write $\mathbf{CTr}_{\mathsf{OGS}}(\langle \Gamma \vdash (\mathtt{M}, \mathtt{S}) : \tau \rangle)$ *for the set of complete traces of* $\langle \Gamma \vdash (\mathtt{M}, \mathtt{S}) : \tau \rangle$.

▶ **Definition 7.** *Given* $\Sigma, \Gamma \vdash \mathtt{M}_1, \mathtt{M}_2 : \tau$, *term* $\mathtt{M}_1$ *is a* trace approximation *of* $\mathtt{M}_2$, *written* $\Sigma, \Gamma \vdash \mathtt{M}_1 \preceq_{\mathsf{tr}} \mathtt{M}_2 : \tau$, *if for all* $\mathtt{S}$ *with* $\vdash \mathtt{S} : \Sigma$, *we have* $\mathbf{CTr}_{\mathsf{OGS}}(\langle \Gamma \vdash (\mathtt{M}_1, \mathtt{S}) : \tau \rangle) \subseteq \mathbf{CTr}_{\mathsf{OGS}}(\langle \Gamma \vdash (\mathtt{M}_2, \mathtt{S}) : \tau \rangle)$.

Examples

Example 8 shows that the visibility constraints can be used to perform some garbage collection, exploiting the property that the store is not higher-order, hence certain function names, while free, cannot be used by the environment.

▶ **Example 8.** Consider the interactive configuration $\mathbb{I} \triangleq \langle [\mathsf{p}] \mathsf{let}\, x = \mathsf{f}\,(\lambda y.\mathtt{M})\, \mathsf{in}\, \mathtt{N}; \mathtt{S}; \gamma \rangle$: for any typing configuration $\mathbb{S}$ and visible configuration $\mathbb{V}$, if $(\mathbb{I}, \mathbb{S}, \mathbb{V}) \xrightarrow{\bar{\mathsf{f}}(\mathsf{g},\mathsf{q})_{\mathsf{p}}}_{\mathsf{ogs}} \xrightarrow{\mathsf{q}(n)}_{\mathsf{ogs}} (\langle \mathtt{P}; \mathtt{S}'; \gamma' \rangle, \mathbb{S}', \mathbb{V}')$, then $\mathbf{Tr}_{\mathsf{OGS}}(\langle \mathtt{P}; \mathtt{S}'; \gamma' \rangle, \mathbb{S}', \mathbb{V}') = \mathbf{Tr}_{\mathsf{OGS}}(\langle \mathtt{P}; \mathtt{S}'; \gamma' \setminus \{\mathsf{g}\} \rangle, \mathbb{S}' \setminus \{\mathsf{g}\}, \mathbb{V}')$. This holds because visibility ensures us that g is forgotten after the answer $\bar{\mathsf{q}}(n)$.

More generally considering a configuration $(\langle \mathtt{P}; \mathtt{S}; \gamma \rangle, \mathbb{S}, \langle \mathcal{F}; \mathcal{G}; \mathcal{T} \rangle)$, and a proponent name $\mathsf{g} \in \mathrm{dom}(\gamma)$, if $\mathsf{g} \notin \mathsf{supp}(\mathcal{F}, \mathcal{G})$ then $\mathbf{Tr}_{\mathsf{OGS}}(\langle \mathtt{P}; \mathtt{S}; \gamma \rangle, \mathbb{S}, \langle \mathcal{F}; \mathcal{G}; \mathcal{T} \rangle) = \mathbf{Tr}_{\mathsf{OGS}}(\langle \mathtt{P}; \mathtt{S}; \gamma \setminus \{\mathsf{g}\} \rangle, \mathbb{S} \setminus \{\mathsf{g}\}, \langle \mathcal{F}; \mathcal{G}; \mathcal{T} \rangle)$

▶ **Example 9.** Back to the terms in Example 1 of the introduction, with $!x$ && `not` $!x$ defined as `if` $!x$ `then` (`if` $!x$ `then false else true`) `else false`: in our concurrent language, $\mathtt{M}_1$ and $\mathtt{M}_2$ are trace equivalent. If we had higher-order store, then a distinguishing trace could begin thus: $?((), \mathsf{r}).\bar{\mathsf{f}}(\mathsf{g}, \mathsf{p}).\mathsf{p}().\mathsf{g}(\mathsf{q}).\bar{\mathsf{q}}()$. However, as seen in Example 8, after the answer $\mathsf{p}()$ we can garbage collect g meaning that the question $\mathsf{g}(\mathsf{q})$ is not allowed by the visibility LTS. More generally, after the answer to $\bar{\mathsf{f}}(\mathsf{g}, \mathsf{p})$, name g cannot be used anymore and thus Opponent cannot access x; hence $!x$ && `not` $!x$ always returns `false`.

4 Soundness of OGS

The goal of this section is to sketch the proof of the following theorem relating the trace-semantics preorder with the ciu-preorder.

▶ **Theorem 10** (Soundness). *If* $\varnothing, \Gamma \vdash \mathtt{M}_1 \preceq_{\mathsf{tr}} \mathtt{M}_2 : \tau$ *with* Γ, τ *ref-free, then* $\varnothing, \Gamma \vdash \mathtt{M}_1 \preceq_{ciu} \mathtt{M}_2 : \tau$.

Composition via synchronization

The first step to prove this theorem is to introduce a way to compose the embedding of a term $\mathtt{M}$ as an active OGS configuration, with the embedding of an evaluation context $\mathtt{D}$ as a passive configuration. This composition is defined by a form of synchronization between transitions of $\mathcal{L}_{\mathsf{OGS}}$ triggered respectively by a passive and an active configuration. As standard in process algebra, this synchronization is defined via a form of parallel composition. Since our goal is to compose two OGS configurations that are "closing" each other (as in the composition of $\mathtt{D}[\mathtt{M}\{\gamma\}]$ that provides a closed term), we do not allow the composed configuration to interact with the outside world. Thus the parallel composition is defined as a reduction relation.

We reflect this idea that the two OGS configurations are closing each other by introducing the notion of pairs of *composable* configurations $(\mathbb{G}_\mathsf{P}, \mathbb{G}_\mathsf{O})$. It enforces, among other things, that each suspended contexts present in one configuration corresponds to a thread in the other, and that their respective typing configurations can be written as $\langle \Delta_O \mid \Delta_P \rangle$ and $\langle \Delta_P, \mathtt{p}_f : \neg\mathtt{Unit} \mid \Delta_O \rangle$. The extra continuation name $\mathtt{p}_f$ in Opponent typing configuration is used to perform the final Opponent answer.

▶ **Definition 11** (Parallel composition). *We define the reduction relation* $\mapsto_{\mathsf{so}}$ *on pairs of* OGS *composable configurations* $(\mathbb{G}_1 \parallel \mathbb{G}_2)$*, defined as* $\mapsto_{\mathsf{so}}^{\mathsf{red}} \cup \mapsto_{\mathsf{so}}^{\mathsf{sync}}$*, where:*

$$\frac{\mathbb{G}_\mathsf{P} \to_{\mathsf{ogs}} \mathbb{H}_\mathsf{P} \qquad \mathsf{supp}(\mathbb{H}_\mathsf{P}) \cap \mathsf{supp}(\mathbb{G}_\mathsf{O}) \subseteq \mathsf{supp}(\mathbb{G}_\mathsf{P}) \cap \mathsf{supp}(\mathbb{G}_\mathsf{O})}{(\mathbb{G}_\mathsf{P} \parallel \mathbb{G}_\mathsf{O}) \mapsto_{\mathsf{so}}^{\mathsf{red}} (\mathbb{H}_\mathsf{P} \parallel \mathbb{G}_\mathsf{O})}$$

$$\frac{\mathbb{G}_\mathsf{O} \to_{\mathsf{ogs}} \mathbb{H}_\mathsf{O} \qquad \mathsf{supp}(\mathbb{H}_\mathsf{O}) \cap \mathsf{supp}(\mathbb{G}_\mathsf{P}) \subseteq \mathsf{supp}(\mathbb{G}_\mathsf{O}) \cap \mathsf{supp}(\mathbb{G}_\mathsf{P})}{(\mathbb{G}_\mathsf{P} \parallel \mathbb{G}_\mathsf{O}) \mapsto_{\mathsf{so}}^{\mathsf{red}} (\mathbb{G}_\mathsf{P} \parallel \mathbb{H}_\mathsf{O})}$$

$$\frac{\mathbb{G}_\mathsf{P} \xrightarrow{\mathbf{m}}_{\mathsf{ogs}} \mathbb{H}_\mathsf{P} \qquad \mathbb{G}_\mathsf{O} \xrightarrow{\overline{\mathbf{m}}}_{\mathsf{ogs}} \mathbb{H}_\mathsf{O}}{(\mathbb{G}_\mathsf{P} \parallel \mathbb{G}_\mathsf{O}) \mapsto_{\mathsf{so}}^{\mathsf{sync}} (\mathbb{H}_\mathsf{P} \parallel \mathbb{H}_\mathsf{O})}$$

Following [25], the nominal side-condition for $\mapsto_{\mathsf{so}}^{\mathsf{red}}$ ensures us that freshly-allocated locations in one configuration do not appear in the other configuration. Thus composability is preserved by $\mapsto_{\mathsf{so}}^{\mathsf{sync}}$. We use parallel composition to observe the synchronization process between two OGS configurations. So we write $(\mathbb{G}_\mathsf{P} \parallel \mathbb{G}_\mathsf{O}) \Downarrow \mathsf{p}$, when there exist two OGS configurations $\mathbb{H}_\mathsf{P}, \mathbb{H}_\mathsf{O}$ such that $(\mathbb{G}_\mathsf{P} \parallel \mathbb{G}_\mathsf{O}) \mapsto_{\mathsf{so}}^{*} (\mathbb{H}_\mathsf{P} \parallel \mathbb{H}_\mathsf{O})$, with the interactive configuration of $\mathbb{H}_\mathsf{P}$ of the shape $\langle \mathtt{S}_\mathsf{P}; \gamma_\mathsf{P} \rangle$, and the interactive configuration of $\mathbb{H}_\mathsf{O}$ of the shape $\langle [\mathsf{p}](); \mathtt{S}_\mathsf{O}; \gamma_\mathsf{O} \rangle$.

Observational adequacy

As the parallel composition $(\mathbb{G}_\mathsf{P} \parallel \mathbb{G}_\mathsf{O})$ is formed by two composable configurations that are closing each other, we introduce a way to merge them into an operational configuration, written $(\mathbb{G}_\mathsf{P} \between \mathbb{G}_\mathsf{O})$, which is of the shape $(\mathtt{M}\{\gamma\}, \mathtt{S})$ with:

- $\mathtt{M}$ is defined by shuffling the thread pools of $\mathbb{G}_\mathsf{P}$ and $\mathbb{G}_\mathsf{O}$ into one, feeding their respective holes. It uses the trees $\mathcal{T}_\mathsf{P}$ and $\mathcal{T}_\mathsf{O}$ of the visible configuration to substitute the threads in their corresponding suspended contexts.
- γ is defined by shuffling the value assignments of $\mathbb{G}_\mathsf{P}$ and $\mathbb{G}_\mathsf{O}$ into one, following the order in which the function names in their domain have been introduced. One thus needs to enforce that such an order does exist, which is done in the definition of composability, and preserved by the reduction semantics of parallel composition.
- $\mathtt{S}$ is simply the concatenation $\mathtt{S}_\mathsf{P} \cdot \mathtt{S}_\mathsf{O}$ of the respective stores of $\mathbb{G}_\mathsf{P}$ and $\mathbb{G}_\mathsf{O}$, as they are disjoint since we do not allow shared store, and hold only first-order values so that we do not need to use γ on them.

While the shuffling of γ_P and γ_O is the same construction as the one used in sequential OGS (e.g. [15, 5]), the one for the shuffling of thread pools is new. Indeed, in a sequential well-bracketed setting, rather than two tree structures, we simply have two stacks of same length.

Our goal is then to relate the observation over $(\mathbb{G}_\mathsf{P} \parallel \mathbb{G}_\mathsf{O})$ defined above via the reduction semantics of the parallel composition, with the termination observation on $(\mathbb{G}_\mathsf{P} \between \mathbb{G}_\mathsf{O})$ defined via the operational reduction.

▶ **Theorem 12.** *Taking* $\mathbb{G}_\mathsf{P}, \mathbb{G}_\mathsf{O}$ *two composable OGS configurations, then* $(\mathbb{G}_\mathsf{P} \parallel \mathbb{G}_\mathsf{O}) \Downarrow \mathsf{p}_f$ *if and only if* $(\mathbb{G}_\mathsf{P} \between \mathbb{G}_\mathsf{O}) \Downarrow ()$.

The left-to-right direction follows the standard proof of OGS, namely that if $(\mathbb{G}_\mathsf{P} \parallel \mathbb{G}_\mathsf{O}) \mapsto^{\mathsf{sync}}_{\mathsf{so}} (\mathbb{H}_\mathsf{P} \parallel \mathbb{H}_\mathsf{O})$ then $(\mathbb{G}_\mathsf{P} \between \mathbb{G}_\mathsf{O}) = (\mathbb{H}_\mathsf{P} \between \mathbb{H}_\mathsf{O})$. The right-to-left direction is more challenging in presence of parallelism in the programming language. Indeed, when an operational reduction is available for $(\mathbb{G}_\mathsf{P} \between \mathbb{G}_\mathsf{O})$, some synchronisation steps may be required for $(\mathbb{G}_\mathsf{P} \parallel \mathbb{G}_\mathsf{O})$ to perform the same reduction. We then prove that we can always perform the synchronization steps eagerly, without losing the possibility of making a reduction step when available. Finally, we show that there are always only a finite number of synchronization steps between two reduction steps.

From $(\langle \Sigma \vdash (\mathtt{M}, \mathtt{S}) : \tau \rangle \parallel \langle \vdash (\mathtt{D}, \gamma, \mathtt{S}') : (\tau \rightsquigarrow \mathtt{Unit}; \Gamma) \rangle) \mapsto^{\mathsf{sync}}_{\mathsf{so}} (\mathbb{G}_\mathsf{P} \parallel \mathbb{G}_\mathsf{O})$, we derive $(\mathbb{G}_\mathsf{P} \between \mathbb{G}_\mathsf{O}) = (\mathtt{D}[\mathtt{M}\{\gamma\}], \mathtt{S} \cdot \mathtt{S}')$; we then deduce the following corollary of Theorem 12 on initial configurations.

▶ **Corollary 13.** $(\mathtt{D}[\mathtt{M}\{\gamma\}], \mathtt{S} \cdot \mathtt{S}') \Downarrow ()$ *if and only if* $(\langle \Sigma \vdash (\mathtt{M}, \mathtt{S}) : \tau \rangle \parallel \langle \vdash (\mathtt{D}, \gamma, \mathtt{S}') : (\tau \rightsquigarrow \mathtt{Unit}; \Gamma) \rangle) \Downarrow \mathsf{p}_f$.

Proof of Soundness

The final ingredient to prove the soundness statement is to relate the observation defined from the reduction relation of parallel composition with the existence of a complete trace.

▶ **Theorem 14.** *Taking* $\mathbb{G}_\mathsf{P}, \mathbb{G}_\mathsf{O}$ *two composable OGS configurations, then* $(\mathbb{G}_\mathsf{P} \parallel \mathbb{G}_\mathsf{O}) \Downarrow \mathsf{p}_f$ *if and only if there exists a complete trace* $\mathbf{t}$ *such that* $\mathbf{t} \in \mathbf{CTr}_{\mathsf{OGS}}(\mathbb{G}_\mathsf{P})$ *and* $\overline{\mathbf{t} \cdot \mathsf{p}_f()} \in \mathbf{CTr}_{\mathsf{OGS}}(\mathbb{G}_\mathsf{O})$.

▶ **Corollary 15.** $(\langle \Sigma \vdash (\mathtt{M}, \mathtt{S}) : \tau \rangle \parallel \langle \vdash (\mathtt{D}, \gamma, \mathtt{S}') : (\tau \rightsquigarrow \mathtt{Unit}; \Gamma) \rangle) \Downarrow \mathsf{p}_f$ *if and only if there exist a trace* $\mathbf{t}$ *such that* $\mathbf{t} \in \mathbf{CTr}_{\mathsf{OGS}}(\langle \Sigma \vdash (\mathtt{M}, \mathtt{S}) : \tau \rangle)$ *and* $\overline{\mathbf{t} \cdot \mathsf{p}_f()} \in \mathbf{CTr}_{\mathsf{OGS}}(\langle \vdash (\mathtt{D}, \gamma, \mathtt{S}') : (\tau \rightsquigarrow \mathtt{Unit}; \Gamma) \rangle)$

We now have all is needed to prove Theorem 10.

Proof. Take $\Sigma, \Gamma, \tau, \mathtt{M}_1, \mathtt{M}_2$ such that $\Sigma, \Gamma \vdash \mathtt{M}_1 \preceq_{\mathsf{tr}} \mathtt{M}_2 : \tau$. Let $\Sigma', \gamma, \mathtt{D}, \mathtt{S}, \mathtt{S}'$ as in Definition 3 of the ciu-preorder, with $(\mathtt{D}[\mathtt{M}_1\{\gamma\}]; \mathtt{S} \cdot \mathtt{S}') \Downarrow ()$. We show that $(\mathtt{D}[\mathtt{M}_2\{\gamma\}]; \mathtt{S} \cdot \mathtt{S}') \Downarrow ()$. By Corollary 13, we have $(\langle \Sigma \vdash (\mathtt{M}_1, \mathtt{S}) : \tau \rangle \parallel \langle \vdash (\mathtt{D}, \gamma, \mathtt{S}') : (\tau \rightsquigarrow \mathtt{Unit}; \Gamma) \rangle) \Downarrow \mathsf{p}_f$. From Corollary 15, we infer that there is a trace $\mathbf{t}$ such that $\mathbf{t} \in \mathbf{CTr}_{\mathsf{OGS}}(\langle \Sigma \vdash (\mathtt{M}_1, \mathtt{S}) : \tau \rangle)$ and $\overline{\mathbf{t} \cdot \mathsf{p}_f()} \in \mathbf{CTr}_{\mathsf{OGS}}(\langle \vdash (\mathtt{D}, \gamma, \mathtt{S}') : (\tau \rightsquigarrow \mathtt{Unit}; \Gamma) \rangle)$.

We have $\Sigma, \Gamma \vdash \mathtt{M}_1 \preceq_{\mathsf{tr}} \mathtt{M}_2 : \tau$, therefore $\mathbf{t} \in \mathbf{CTr}_{\mathsf{OGS}}(\langle \Sigma \vdash (\mathtt{M}_2, \mathtt{S}) : \tau \rangle)$. By Corollary 15, $(\langle \Sigma \vdash (\mathtt{M}_2, \mathtt{S}) : \tau \rangle \parallel \langle \vdash (\mathtt{D}, \gamma, \mathtt{S}') : (\tau \rightsquigarrow \mathtt{Unit}; \Gamma) \rangle) \Downarrow \mathsf{p}_f$; finally, by Corollary 13, $(\mathtt{D}[\mathtt{M}_2\{\gamma\}]; \mathtt{S} \cdot \mathtt{S}') \Downarrow ()$. ◀

5 Completeness

We prove completeness for a subset of our language. Before delving into the result, we discuss the problem for completeness on the whole language. In the usual strategy for completeness in OGS, one takes two contextually-equivalent term, and a trace $\mathbf{t}$ for one of them. Then one builds a context that generates the complement trace; thus from the contextual equivalence between the two terms combined with Corollaries 13 and 15, one infers that also the other term can produce the trace $\mathbf{t}$. Now, suppose $\mathbf{t}$ is as follows, using "$\mathtt{V}$" to designate an abstract value such that questions on names in "$\mathtt{V}$" would trigger questions on names in $\mathtt{V}$ (and _ or ... for non-relevant subterms):

$$\mathsf{f}(\_,\mathsf{p})\cdot\mathsf{f}(\_,\mathsf{q})\cdot\overline{\mathsf{p}}(\mathsf{g})\cdot\mathsf{f}(\text{“}\mathsf{g}\text{”},\mathsf{r})\cdot\overline{\mathsf{q}}(\mathsf{h})\cdot\mathsf{f}(\text{“}\langle\mathsf{g},\mathsf{h}\rangle\text{”},\mathsf{s})\cdot\overline{\mathsf{r}}()\cdot\overline{\mathsf{s}}()$$

Trace $\mathbf{t}$ can indeed be generated by a configuration. We explain why its dual trace $\overline{\mathbf{t}}$ cannot be generated by a context. The two first actions of $\overline{\mathbf{t}}$ should necessarily emanate from parallel components, because, in $\mathbf{t}$, the second question is asked before the first one is answered. That is, the context should be of the form $\mathsf{par}\ \mathsf{let}\,\mathsf{g}=\mathsf{f}\,\_\ \mathsf{in}\ \ldots\ \mathsf{and}\ \mathsf{let}\,\mathsf{h}=\mathsf{f}\,\_\ \mathsf{in}\ \ldots$.
Moreover, in $\mathbf{t}$, question $\mathsf{f}(\text{“}\mathsf{g}\text{”},\mathsf{r})$ precedes the answer at q, and uses a name introduced in the answer at p; this forces the context to have the form $\mathsf{par}\ \mathsf{let}\,\mathsf{g}=\mathsf{f}\,\_\,\mathsf{in}\,\mathsf{f}\ \mathsf{g}\ \mathsf{and}\ \mathsf{let}\,\mathsf{h}=\mathsf{f}\,\_\ \mathsf{in}\ \ldots$.
Correspondingly, $\overline{\mathbf{t}}$ should exhibit the action $\overline{\mathsf{f}}(\text{“}\langle\mathsf{g},\mathsf{h}\rangle\text{”},\mathsf{s})$; this is impossible, as no subterm of the context knows both g and h.

In the remainder of the section we focus on the restricted language, obtained by imposing that functions at the boundary only return first-order values, which we call the *fo-returning fragment*. Notwithstanding the restriction, concurrency introduces significant challenges. First of all, the order in which two consecutive actions occur in a trace cannot always be enforced. That is, a term, or a context, that produces a given trace will also produce other traces, obtained by commuting certain actions. A commutation is possible, intuitively, when the two actions are independent; that is, they are syntactically unrelated and there is no internal causality between them (i.e., via use of references). Concretely this may occur when the two actions: are both Proponent moves; are both Opponent moves; the first is a Proponent move $\mathbf{p}$ and the second an Opponent move $\mathbf{o}$ that is not *justified* by the Proponent move, written $\mathbf{p}\not\curvearrowright\mathbf{o}$, in that the Opponent move does not use names introduced in the Proponent move[2]. We write $\mathbf{t}'\preceq\mathbf{t}$ if $\mathbf{t}'$ is a legal permutation of $\mathbf{t}$, where $\preceq$ is the least precongruence on traces induced by the following relation between consecutive actions:

$$\frac{}{\mathbf{o}\cdot\mathbf{o}'\preceq\mathbf{o}'\cdot\mathbf{o}}\qquad\frac{}{\mathbf{p}\cdot\mathbf{p}'\preceq\mathbf{p}'\cdot\mathbf{p}}\qquad\textsc{Crt}\ \frac{\mathbf{p}\not\curvearrowright\mathbf{o}}{\mathbf{o}\cdot\mathbf{p}\preceq\mathbf{p}\cdot\mathbf{o}}$$

▶ Remark 16. The dual of the courtesy rule `Crt` does not appear, meaning that an Opponent move followed by a Proponent move cannot be commuted. The reason is that, in these cases, an internal causality may be created using references [9]. To see an example, consider a pool of threads $[\mathsf{p}]\{\mathtt{E}\}_\mathsf{r}|[\mathsf{q}]\mathsf{f}\,\mathtt{V}$: the answer on r and the question on f may look independent; however we can enforce an order between them thus:

$$[\mathsf{p}]\{(\lambda x.\ \ell:=1;x)\mathtt{E}\}_\mathsf{r}|[\mathsf{q}]\mathtt{if}\ !\ell=1\ \mathtt{then}\ \mathsf{f}\,\mathtt{V}\ \mathtt{else}\ \Omega$$

If the `if then else` is evaluated before the answer on r, then the process diverges; this is avoided if the question on f happens after the answer on r.

▶ Remark 17 (Divergence). As in the previous remark, it is a standard practice to use divergence to throw away undesired behaviours of some terms. This takes advantage of the use of complete traces instead of general traces, when any part of a term has a divergence, the term as a whole cannot terminate and the execution cannot be completed into a complete trace and this execution does not need to be considered.

We also have to take into account, however, that our language does not allow higher-order store, and hence only the *visible traces*, that is, intuitively, traces produced by the visible LTS on some given visible configuration, are relevant. Lemma 18 shows that, on visible traces, the permutations defined above are indeed possible.

[2] This is referred to as “courtesy” in concurrent game semantics [7]

▶ **Lemma 18.** *If* $\mathbf{t} \in \mathbf{CTr}_{\mathsf{OGS}}(\mathbb{G})$ *then for all* $\mathbf{t}' \preceq \mathbf{t}$ *such that* $\mathbf{t}'$ *is visible from* $\mathbb{V}_{\mathbb{G}}$, $\mathbf{t}' \in \mathbf{CTr}_{\mathsf{OGS}}(\mathbb{G})$.

In the proof of the lemma, we reason by induction on the proof of $\mathbf{t}' \preceq \mathbf{t}$. The difficult case is the courtesy rule `Crt` where, sometimes, a permutation may imply a modification on the Opponent actions involved, such as changing the thread label $\mathtt{q}$ in $\mathtt{f}(\mathtt{A},\mathtt{p})_{\mathtt{q}}$. The final and crucial step is to show that, for any given complete visible trace, a context exists that generates its complement trace and all its family of legal permutations.

▶ **Lemma 19.** *For all* $\mathtt{M},\mathtt{S},\Gamma,\tau$ *in the fo-returning fragment with* Γ,τ *ref-free, for all complete trace* $\mathbf{t} \in \mathbf{CTr}_{\mathsf{OGS}}(\langle \Gamma \vdash (\mathtt{M},\mathtt{S}) : \tau \rangle)$, *there exists a context* $\mathtt{E}$, *a value assignment* γ *and a store* $\mathtt{S}'$ *disjoint from* $\mathtt{S}$ *such that* $\mathbf{CTr}_{\mathsf{OGS}}(\langle \vdash (\mathtt{E},\gamma,\mathtt{S}') : (\tau \to \mathtt{Unit};\Gamma) \rangle) = \{\mathbf{t}' | \mathbf{t}' \preceq \overline{\mathbf{t} \cdot \mathtt{p}_f}$ *with* $\mathbf{t}'$ *visible* $\}$.

The context built in this definability result is a sequential context, which does not use any parallel composition before the first interaction with the term. We sketch the reasoning, taking the point of view of the environment $\mathtt{E},\gamma$ we build. Some care is needed for the following issues:

- As shown in Remark 16, an Opponent move cannot be commuted with a following Proponent move. This can be enforced along the lines of what suggested in that remark.
- For each Opponent function name `f`, if there are n questions with subject `f`, then `f` should be called exactly n times.

We handle these issues using locks, modeled by means of references. A lock may be tested to know its state (open/closed); and its state may be atomically changed, exploiting the `CAS` construct. When something unexpected is encountered in a branch, we force a divergence, so to rule out the possibility of taking that branch.

- A lock is modeled by means of a reference, set to 0 to mean that the lock is closed, or 1 if open. For each Opponent move $\mathbf{o}$ we create a lock $\ell_{\mathbf{o}}$, initially closed. After the move $\mathbf{o}$ is performed, we open the lock. For each Proponent move $\mathbf{p}$, we consider the set of Opponent moves that should occur before $\mathbf{p}$ and the set $L_{\mathbf{p}}$ of their locks. Before performing any Proponent move $\mathbf{p}$, we check that the references in $L_{\mathbf{p}}$ are all set to 1; if they are not, we diverge: $\mathtt{check\_lock}_{\mathbf{p}} \triangleq \mathsf{Par}_{\ell \in L_{\mathbf{p}}}\,(\mathtt{if}\ !\ell = 1\ \mathtt{then}\ ()\ \mathtt{else}\ \Omega)$
- Using the same locks, we ensure that each Opponent question occurs exactly once. After an Opponent question $\mathbf{o}$, and before any Proponent action triggered by $\mathbf{o}$, we check that the lock $\ell_{\mathbf{o}}$ is set to 0: if it is, we set it to 1 and continue, otherwise we diverge. To ensure that no interleaving can bypass the test, we use the atomic construct $\mathtt{CAS}\ \ell_{\mathbf{o}}\ 0\ 1$.

A further technicality to handle is about multiple Opponent questions for the same function name. For this we introduce, in the definition of these functions, a separate branch for each of them; to choose the branch we start by inspecting the structure of the abstract value received; when some abstract values are indistinguishable, we set a non-deterministic branching using a race between the different choices.

Exploiting the above ideas, we build an evaluation context $\mathtt{E}_{\mathbf{t}}$, a value assignment $\delta_{\mathbf{t}}$, and a store $\mathtt{S}_{\mathbf{t}}$ that can produce the expected trace. We then need to show three results: the context has the expected type; the context generates the trace; the context does not generate unexpected complete traces (i.e., traces $\mathbf{t}'$ with $\neg(\mathbf{t}' \preceq \mathbf{t})$). For the last result, intuitively, the use of locks makes sure that the context diverges as soon an unwanted prefix trace is generated.

▶ **Theorem 20** (Completeness wrt fo-returning fragment). *If* $\varnothing, \Gamma \vdash \mathtt{M}_1 \preceq_{ciu} \mathtt{M}_2 : \tau$ *with* $\mathtt{M}_1$, $\mathtt{M}_2$ *being terms of the fo-returning fragment and* Γ, τ *ref-free, then* $\varnothing, \Gamma \vdash \mathtt{M}_1 \preceq_{\mathsf{tr}} \mathtt{M}_2 : \tau$.

Proof. We take a trace $\mathbf{t}$ generated by $\mathtt{M}_1$. Using the existence of a context $(\mathtt{E}, \gamma, \mathtt{S}')$ generating $\{\mathbf{t}' | \mathbf{t}' \preceq \overline{\mathbf{t} \cdot \mathsf{p}_f}$ with $\mathbf{t}'$ visible $\}$, ciu approximation, Corollaries 13 and 15 in both directions, we have $\big(\langle \Gamma \vdash (\mathtt{M}_2, \mathtt{S}) : \tau \rangle \mid\mid \langle \vdash (\mathtt{E}, \gamma, \mathtt{S}') : (\tau \rightsquigarrow \mathtt{Unit}; \Gamma) \rangle\big) \Downarrow \mathsf{p}_f$. Hence there is some visible $\mathbf{t}'$ generated by $\mathtt{M}_2$ such that $\overline{\mathbf{t}' \cdot \mathsf{p}_f}$ is generated by the context. This means that $\overline{\mathbf{t}' \cdot \mathsf{p}_f} \preceq \overline{\mathbf{t} \cdot \mathsf{p}_f}$; we can conclude that $\mathbf{t} \preceq \mathbf{t}'$ thus $\mathbf{t}$ is a complete trace generated by $\mathtt{M}_2$. ◀

6 Conclusions, Related and Future Work

We have investigated how visibility, a central notion in operational game semantics (OGS), can be accommodated within a concurrent setting.

We currently do not know whether completeness holds on the full language. Our OGS can be thought of as a concurrent extension of [15], which gives a systematic development of visibility and well-bracketing for OGS in the sequential case. To our knowledge, the first OGS that took (O-)visibility into account is [14]. A difference between [14] and our work (as well as [15]) is that we do not allow locations to be exchanged between the program and the environment. To handle location exchange, and the resulting shared stores, one needs to generalise moves to moves-with-store [24, 29], where the attached store contains the values for the disclosed locations. In our concurrent setting, we leave such issues for future work. Although not an instance of OGS, the trace semantics for concurrent objects [20] also employs concepts used in game semantics such as well-bracketing and courtesy. Their definition of well-bracketing makes essential use of thread identifiers and requires that each subtrace with the same identifier is well-bracketed in the standard sense.

One of the closest denotational game models related to our OGS is that by Ghica and Murawski [9]. They developed a game semantics for Idealised Parallel Algol – essentially the call-by-name variant of our target language – that is fully abstract with respect to may-testing equivalence. While the relationship between operational and denotational game semantics has been made formal in some cases (for example, [14] relates [24] to [29]) there is no general method to relate operational and denotational approaches. It would be an interesting future work to formally relate our OGS with the denotational model of Ghica and Murawski. The Ghica-Murawski model has also been extended to a causal model based on event structures [7, 6]. Both in the sequential and the causal case, no notion of visibility is explicitly given; we note that the visibility condition of causal models [6] is a concept closer to innocence rather than the absence of higher-order references. On the other hand, Laird has introduced a notion of visibility in a concurrent setting by introducing *concurrency pointers* [23], an internal thread information attached to plays. Investigating the relationship between concurrent pointers and our labeling by continuation names is left for future work.

The concept of visibility has been incorporated into the π-calculus as a type system [10]. However, computation is purely sequential, and how to capture concurrent visibility as a type system remains open. This makes it hard to link our OGS with process encodings, unlike the case for pure λ-calculus against which a formal connection is given [17].

Other operational techniques to reason about program equivalence of higher-order languages with references or asynchronous channels and/or concurrency includes Kripke logical relation [4, 8] and various form of bisimulation – such as contextual and normal [32, 19, 27], normal form [3] and environmental [33, 21] bisimulations – or their hybrids [12, 18, 16]. In these works, references, if considered, are usually full higher-order references rather than ground references. An exception is [8] where the absence of higher-order references in a

higher-order sequential language is captured as the ability to "backtrack" the Kripke frame. This technique was imported to Kripke normal-form bisimulations in [16], via a formal connection with the notion of visibility. Extending this "backtrack" proof technique to a concurrent setting and relating it to our notion of concurrent visibility is an exciting research direction.

References

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

2 Samson Abramsky and Guy McCusker. Linearity, sharing and state: a fully abstract game semantics for idealized algol with active expressions. In Jean-Yves Girard, Mitsuhiro Okada, and Andre Scedrov, editors, *Linear Logic Tokyo Meeting 1996, Keio University, Mita Campus, Tokyo, Japan, March 29 - April 2, 1996*, Electronic Notes in Theoretical Computer Science, pages 2–14. Elsevier, 1996. doi:10.1016/S1571-0661(05)80398-6.

3 Dariusz Biernacki, Sergueï Lenglet, and Piotr Polesiuk. A complete normal-form bisimilarity for state. In Mikolaj Bojanczyk and Alex Simpson, editors, *Foundations of Software Science and Computation Structures - 22nd International Conference, FOSSACS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings*, Lecture Notes in Computer Science, pages 98–114. Springer, 2019. doi:10.1007/978-3-030-17127-8_6.

4 Lars Birkedal, Filip Sieczkowski, and Jacob Thamsborg. A concurrent logical relation. In Patrick Cégielski and Arnaud Durand, editors, *Computer Science Logic - 26th International Workshop / 21st Annual Conference of the EACSL, CSL 2012, Fontainebleau, France, September 3-6, 2012*, volume 16 of *LIPIcs*, pages 107–121. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPIcs.CSL.2012.107.

5 Peio Borthelle, Tom Hirschowitz, Guilhem Jaber, and Yannick Zakowski. An abstract, certified account of operational game semantics. In Viktor Vafeiadis, editor, *Programming Languages and Systems - 34th European Symposium on Programming, ESOP 2025, Held as Part of the International Joint Conferences on Theory and Practice of Software, ETAPS 2025, Hamilton, ON, Canada, May 3-8, 2025, Proceedings, Part I*, Lecture Notes in Computer Science, pages 172–199. Springer, 2025. doi:10.1007/978-3-031-91118-7_7.

6 Simon Castellan and Pierre Clairambault. Disentangling parallelism and interference in game semantics. *Log. Methods Comput. Sci.*, 20(3), 2024. doi:10.46298/LMCS-20(3:24)2024.

7 Simon Castellan, Pierre Clairambault, and Glynn Winskel. Thin games with symmetry and concurrent Hyland-Ong games. *Log. Methods Comput. Sci.*, 15(1), 2019. doi:10.23638/LMCS-15(1:18)2019.

8 Derek Dreyer, Georg Neis, and Lars Birkedal. The impact of higher-order state and control effects on local relational reasoning. *J. Funct. Program.*, 22(4-5):477–528, 2012. doi:10.1017/S095679681200024X.

9 Dan R. Ghica and Andrzej S. Murawski. Angelic semantics of fine-grained concurrency. *Ann. Pure Appl. Log.*, 151(2-3):89–114, 2008. doi:10.1016/J.APAL.2007.10.005.

10 Daniel Hirschkoff, Iwan Quémerais, and Davide Sangiorgi. First-order store and visibility in name-passing calculi. In Patricia Bouyer and Jaco van de Pol, editors, *36th International Conference on Concurrency Theory, CONCUR 2025, Aarhus, Denmark, August 26-29, 2025*, LIPIcs, pages 23:1–23:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.CONCUR.2025.23.

11 Furio Honsell, Ian A Mason, Scott Smith, and Carolyn Talcott. A variable typed logic of effects. *Information and computation*, 119(1):55–90, 1995. doi:10.1006/INCO.1995.1077.

12 Chung-Kil Hur, Derek Dreyer, Georg Neis, and Viktor Vafeiadis. The marriage of bisimulations and kripke logical relations. In John Field and Michael Hicks, editors, *Proceedings of the 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2012, Philadelphia, Pennsylvania, USA, January 22-28, 2012*, pages 59–72. ACM, 2012. doi:10.1145/2103656.2103666.

13 J. M. E. Hyland and C.-H. Luke Ong. On full abstraction for PCF: i, ii, and III. *Inf. Comput.*, 163(2):285–408, 2000. doi:10.1006/INCO.2000.2917.

14 Guilhem Jaber. Operational nominal game semantics. In Andrew M. Pitts, editor, *Foundations of Software Science and Computation Structures - 18th International Conference, FoSSaCS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*, Lecture Notes in Computer Science, pages 264–278. Springer, 2015. doi:10.1007/978-3-662-46678-0_17.

15 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. doi:10.1007/978-3-030-72019-3_13.

16 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. doi:10.1109/LICS52264.2021.9470524.

17 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, Göttingen, Germany (Virtual Conference), February 14-19, 2022*, LIPIcs, pages 25:1–25:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.CSL.2022.25.

18 Guilhem Jaber and Nicolas Tabareau. Kripke open bisimulation - A marriage of game semantics and operational techniques. In Xinyu Feng and Sungwoo Park, editors, *Programming Languages and Systems - 13th Asian Symposium, APLAS 2015, Pohang, South Korea, November 30 - December 2, 2015, Proceedings*, Lecture Notes in Computer Science, pages 271–291. Springer, 2015. doi:10.1007/978-3-319-26529-2_15.

19 Alan Jeffrey and Julian Rathke. A theory of bisimulation for a fragment of concurrent ML with local names. *Theor. Comput. Sci.*, 323(1-3):1–48, 2004. doi:10.1016/J.TCS.2004.03.005.

20 Alan Jeffrey and Julian Rathke. A fully abstract may testing semantics for concurrent objects. *Theor. Comput. Sci.*, 338(1-3):17–63, 2005. doi:10.1016/J.TCS.2004.10.012.

21 Vasileios Koutavas and Mitchell Wand. Small bisimulations for reasoning about higher-order imperative programs. In J. Gregory Morrisett and Simon L. Peyton Jones, editors, *Proceedings of the 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2006, Charleston, South Carolina, USA, January 11-13, 2006*, pages 141–152. ACM, 2006. doi:10.1145/1111037.1111050.

22 James Laird. Full abstraction for functional languages with control. In *Proceedings, 12th Annual IEEE Symposium on Logic in Computer Science, Warsaw, Poland, June 29 - July 2, 1997*, pages 58–67. IEEE Computer Society, 1997. doi:10.1109/LICS.1997.614931.

23 James Laird. A game semantics of idealized CSP. In Stephen D. Brookes and Michael W. Mislove, editors, *Seventeenth Conference on the Mathematical Foundations of Programming Semantics, MFPS 2001, Aarhus, Denmark, May 23-26, 2001*, Electronic Notes in Theoretical Computer Science, pages 232–257. Elsevier, 2001. doi:10.1016/S1571-0661(04)80965-4.

24 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*, Lecture Notes in Computer Science, pages 667–679. Springer, 2007. doi:10.1007/978-3-540-73420-8_58.

25 James Laird. A Curry-style semantics of interaction: From untyped to second-order lazy $\lambda\mu$-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*, Lecture Notes in Computer Science, pages 422–441. Springer, 2020. doi:10.1007/978-3-030-45231-5_22.

26 Søren B. Lassen and Paul Blain Levy. Typed normal form bisimulation for parametric polymorphism. In *Proceedings of the Twenty-Third Annual IEEE Symposium on Logic in Computer Science, LICS 2008, 24-27 June 2008, Pittsburgh, PA, USA*, pages 341–352. IEEE Computer Society, 2008. doi:10.1109/LICS.2008.26.

27 Sergueï Lenglet, Alan Schmitt, and Jean-Bernard Stefani. Characterizing contextual equivalence in calculi with passivation. *Information and Computation*, 209(11):1390–1433, 2011. doi:10.1016/J.IC.2011.08.002.

28 Ian Mason and Carolyn Talcott. Equivalence in functional languages with effects. *Journal of Functional Programming*, 1(3):287–327, 1991. doi:10.1017/S0956796800000125.

29 Andrzej S. Murawski and Nikos Tzevelekos. Game semantics for good general references. In *Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science, LICS 2011, June 21-24, 2011, Toronto, Ontario, Canada*, pages 75–84. IEEE Computer Society, 2011. doi:10.1109/LICS.2011.31.

30 Andrzej S. Murawski and Nikos Tzevelekos. Algorithmic games for full ground references. In Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts, and Roger Wattenhofer, editors, *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part II*, volume 7392 of *Lecture Notes in Computer Science*, pages 312–324. Springer, 2012. doi:10.1007/978-3-642-31585-5_30.

31 Amr Sabry and Matthias Felleisen. Reasoning about programs in continuation-passing style. In Jon L. White, editor, *Proceedings of the Conference on Lisp and Functional Programming, LFP 1992, San Francisco, California, USA, 22-24 June 1992*, pages 288–298. ACM, 1992. doi:10.1145/141471.141563.

32 Davide Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*. PhD thesis CST–99–93, Department of Computer Science, University of Edinburgh, 1992.

33 Davide Sangiorgi, Naoki Kobayashi, and Eijiro Sumii. Environmental bisimulations for higher-order languages. *ACM Trans. Program. Lang. Syst.*, 33(1):5:1–5:69, 2011. doi:10.1145/1889997.1890002.

A Summary of main notations

Notation	Meaning	Definition
$\mathtt{M}, \mathtt{N}$	Terms	Section 2
σ, τ	Types	Section 2
Γ, Σ	Typing context and store typing context	Section 2
$\mathtt{S}, \mathtt{S}[\ell \mapsto \mathtt{V}], \mathtt{S} \cdot \mathtt{T}$	Store, store update and disjoint concatenation of stores	Section 2
$\mapsto_{\mathsf{red}}$	Reduction semantic of terms with store	Figure 1
$\preceq_{ciu}, \simeq_{ciu}$	CIU approximation and CIU equivalence	Definition 3
$\{\mathtt{E}\}_{\mathsf{p}}, \underline{\mathtt{M}}$	Suspended context and partially suspended term	Section 3.1
$\overset{\mu}{\longmapsto}_{\mathsf{env}}, \mathcal{L}_{\mathsf{env}}$	Reduction and LTS for interactions with the environment	Figure 2
$\mathtt{A}$	Abstract values	Section 3.1
$\mathbf{m}, \overline{\mathtt{f}}(\mathtt{A}, \mathsf{p}), \overline{\mathsf{p}}(\mathtt{A}), \mathtt{f}(\mathtt{A}, \mathsf{p}), \mathsf{p}(\mathtt{A})$	Moves	Section 3.1
γ	Substitutions	Section 3.1
$\mathtt{P}, \mathtt{Q}, [\mathsf{p}]\underline{\mathtt{M}}$	Pool of threads	Section 3.1
$\mathbb{I}, \mathbb{J}, \langle \mathtt{P}; \mathtt{S}; \gamma \rangle$	Interactive configurations	Section 3.1
$\overset{\mathbf{m}}{\longrightarrow}_{\mathsf{I}}, \mathcal{L}_{\mathsf{I}}$	Interactive reduction and LTS	Figure 4
$\mathbb{S}, \mathbb{T}, \langle \Delta_{\mathsf{O}} \mid \Delta_{\mathsf{P}} \rangle$	Typing configurations	Section 3.1
$\overset{\cdot}{\rightarrow}_{\mathsf{Ty}}\ \mathcal{L}_{\mathsf{Ty}}$	Typing reduction and LTS	Figure 5
$\mathbb{V}, \mathbb{W}, \langle \mathcal{F}; \mathcal{G}; \mathcal{T} \rangle$	Visible configurations	Section 3.1
$\overset{\cdot}{\rightarrow}_{\mathsf{vis}}\ \mathcal{L}_{\mathsf{vis}}$	Visible reduction and LTS	Figure 6
$\mathbb{G}, \mathbb{H}, (\mathbb{I}, \mathbb{S}, \mathbb{V})$	OGS configurations	Section 3.1
$\rightarrow_{\mathsf{ogs}}, \mathcal{L}_{\mathsf{OGS}}$	OGS reduction and LTS	Section 3.1
$?(\overrightarrow{\mathtt{A}_i}, \mathsf{p}), !(\overrightarrow{\mathtt{A}_i}, \mathsf{p})$	Initial OGS moves	Figure 7
$\preceq_{ctx}, \simeq_{ctx}$	Contextual preorder and contextual equivalence	Section 2
$\preceq_{\mathsf{tr}}, \simeq_{\mathsf{tr}}$	Trace approximation and trace equivalence	Section 3.2
$\|\|, \mapsto_{\mathsf{so}}^{\mathsf{red}}, \mapsto_{\mathsf{so}}^{\mathsf{sync}}$	Parallel composition, operational reduction and synchronisations	Definition 11
$\mathbin{\text{⩓}}$	Composition by merging of configurations	Section 4
$\preceq$	Preorder for commutation of traces	Section 5

B Target Language and Operational Semantics

B.1 Typing

The typing rules are shown in Figure 8

C Operational Game Semantics

▶ **Definition 21.** *An* OGS *configuration* $(\mathbb{I}, \mathbb{S}, \mathbb{V})$*, with:*

- $\mathbb{I} = \langle [\mathsf{p}_1]\underline{\mathtt{M}}_1 | \cdots | [\mathsf{p}_k]\underline{\mathtt{M}}_k; \mathtt{S}; \gamma \rangle$*;*
- $\mathbb{S} = \langle \Delta_{\mathsf{O}} \mid \Delta_{\mathsf{P}} \rangle$*;*
- $\mathbb{V} = \langle \mathcal{F}; \mathcal{G}; \mathcal{T} \rangle$*;*

is said to be valid *when:*

- *there exists a store typing context* Σ *such that:*
  - $\Sigma; \Delta_{\mathsf{O}} \vdash \gamma : \Delta_{\mathsf{P}}$*;*
  - $\vdash \mathtt{S} : \Sigma$*;*
  - *for all* $i \in \{1, \ldots, k\}$*,* $\Sigma; \Delta_{\mathsf{O}} \vdash \underline{\mathtt{M}}_i : \sigma_i$ *with* $\Delta_{\mathsf{O}}(\mathsf{p}_i) = \neg\sigma_i$*.*
- $\mathrm{dom}(\mathcal{F}) = \mathrm{dom}(\Delta_{\mathsf{O}})$ *and* $\mathrm{im}(\mathcal{F}) \subseteq \mathcal{P}_{\mathrm{fin}}(\mathrm{dom}(\Delta_{\mathsf{P}}))$*;*
- $\mathrm{dom}(\mathcal{G}) \subseteq \mathrm{dom}(\Delta_{\mathsf{P}})$ *and* $\mathrm{im}(\mathcal{G}) \subseteq \mathcal{P}_{\mathrm{fin}}(\mathrm{dom}(\Delta_{\mathsf{P}}))$*;*

$\boxed{\Sigma;\Gamma \vdash \mathtt{M} : \sigma}$

$$\textsc{Var}\ \frac{\Gamma(x) = \sigma}{\Sigma;\Gamma \vdash x : \sigma} \qquad \textsc{Loc}\ \frac{\Sigma(\ell) = \sigma}{\Sigma;\Gamma \vdash \ell : \mathbf{ref}_\sigma} \qquad \textsc{Unit}\ \frac{}{\Sigma;\Gamma \vdash () : \mathtt{Unit}} \qquad \textsc{Bool}\ \frac{}{\Sigma;\Gamma \vdash \mathtt{b} : \mathtt{Bool}} \qquad \textsc{Int}\ \frac{}{\Sigma;\Gamma \vdash \mathtt{k} : \mathtt{Int}}$$

$$\textsc{Pair}\ \frac{\Sigma;\Gamma \vdash \mathtt{V} : \sigma \qquad \Sigma;\Gamma \vdash \mathtt{W} : \tau}{\Sigma;\Gamma \vdash \langle \mathtt{V}, \mathtt{W} \rangle : \sigma \times \tau} \qquad \textsc{Proj}\ \frac{\Sigma;\Gamma \vdash \mathtt{V} : \sigma_1 \times \sigma_2 \qquad i \in \{1,2\}}{\Sigma;\Gamma \vdash \mathsf{proj}_i\ \mathtt{V} : \sigma_i}$$

$$\textsc{Abs}\ \frac{\Sigma;\Gamma, x : \sigma \vdash \mathtt{M} : \tau}{\Sigma;\Gamma \vdash \lambda x.\mathtt{M} : \sigma \to \tau} \qquad \textsc{App}\ \frac{\Sigma;\Gamma \vdash \mathtt{V} : \sigma \to \tau \qquad \Sigma;\Gamma \vdash \mathtt{M} : \sigma}{\Sigma;\Gamma \vdash \mathtt{V}\,\mathtt{M} : \tau} \qquad \textsc{Rec}\ \frac{\Sigma;\Gamma, f : \sigma \to \tau, x : \sigma \vdash \mathtt{M} : \tau}{\Sigma;\Gamma \vdash \mathtt{rec}\ f(x).\mathtt{M} : \tau}$$

$$\textsc{Ite}\ \frac{\Sigma;\Gamma \vdash \mathtt{V} : \mathtt{Bool} \qquad \Sigma;\Gamma \vdash \mathtt{M} : \sigma \qquad \Sigma;\Gamma \vdash \mathtt{N} : \sigma}{\Sigma;\Gamma \vdash \mathtt{if\ V\ then\ M\ else\ N} : \sigma}$$

$$\textsc{New}\ \frac{\Sigma;\Gamma \vdash \mathtt{V} : \sigma}{\Sigma;\Gamma \vdash \mathtt{ref\ V} : \mathbf{ref}_\sigma} \qquad \textsc{Deref}\ \frac{\Sigma;\Gamma \vdash \mathtt{V} : \mathbf{ref}_\sigma}{\Sigma;\Gamma \vdash !\mathtt{V} : \sigma} \qquad \textsc{Assign}\ \frac{\Sigma;\Gamma \vdash \mathtt{V} : \mathbf{ref}_\sigma \qquad \Sigma;\Gamma \vdash \mathtt{W} : \sigma}{\Sigma;\Gamma \vdash \mathtt{V} \coloneqq \mathtt{W} : \mathtt{Unit}}$$

$$\textsc{Par}\ \frac{\Sigma;\Gamma \vdash \mathtt{M} : \sigma \qquad \Sigma;\Gamma \vdash \mathtt{N} : \tau}{\Sigma;\Gamma \vdash \mathsf{par}\ \mathtt{M}\ \mathsf{and}\ \mathtt{N} : \sigma \times \tau} \qquad \textsc{Cas}\ \frac{\Sigma;\Gamma \vdash \mathtt{W} : \mathbf{ref}_\sigma \qquad \Sigma;\Gamma \vdash \mathtt{V}_1 : \sigma \qquad \Sigma;\Gamma \vdash \mathtt{V}_2 : \sigma}{\Sigma;\Gamma \vdash \mathtt{CAS\ W\ V}_1\ \mathtt{V}_2 : \mathtt{Bool}}$$

$\boxed{\Sigma;\Gamma \vdash \mathtt{D} : \tau \rightsquigarrow \sigma}$

$$\textsc{Hole}\ \frac{}{\Sigma;\Gamma \vdash \bullet : \sigma \rightsquigarrow \sigma} \qquad \frac{\Sigma;\Gamma \vdash \mathtt{D} : \tau \rightsquigarrow \sigma \qquad \Sigma;\Gamma \vdash \mathtt{V} : \sigma \to \sigma'}{\Sigma;\Gamma \vdash \mathtt{V}\,\mathtt{D} : \tau \rightsquigarrow \sigma'}$$

$$\frac{\Sigma;\Gamma \vdash \mathtt{D} : \tau \rightsquigarrow \sigma_1 \qquad \Sigma;\Gamma \vdash \mathtt{M} : \sigma_2}{\Sigma;\Gamma \vdash \mathsf{par}\ \mathtt{D}\ \mathsf{and}\ \mathtt{M} : \tau \rightsquigarrow \sigma_1 \times \sigma_2} \qquad \frac{\Sigma;\Gamma \vdash \mathtt{M} : \sigma_1 \qquad \Sigma;\Gamma \vdash \mathtt{D} : \tau \rightsquigarrow \sigma_2}{\Sigma;\Gamma \vdash \mathsf{par}\ \mathtt{M}\ \mathsf{and}\ \mathtt{D} : \tau \rightsquigarrow \sigma_1 \times \sigma_2}$$

Figure 8 Typing rules.

- *The tree $\mathcal{T}$ is alternating, that is for all $\mathsf{p} \in \mathcal{T}$:*
  - *either p is the root and $\mathsf{p} \in \mathrm{dom}(\Delta_{\mathsf{O}})$*
  - *or $\mathsf{p} \in \mathrm{dom}(\Delta_{\mathsf{O}})$ and $\mathsf{q} \in \mathrm{dom}(\Delta_{\mathsf{P}})$ where q is the parent of p;*
  - *or $\mathsf{p} \in \mathrm{dom}(\Delta_{\mathsf{P}})$ and $\mathsf{q} \in \mathrm{dom}(\Delta_{\mathsf{O}})$ where q is the parent of p;*
- *The set of nodes of $\mathcal{T}$ that are in Δ_{O} is equal to $\{\mathsf{p}_1, \dots, \mathsf{p}_k\}$;*
- *$\{\mathtt{E}\}_{\mathsf{p}_j}$ appears in $\underline{\mathtt{M}_i}$ if and only if p_j is a child of p_i.*

▶ **Lemma 22.** *Taking $\mathbb{G}$ a valid* OGS *configuration such that $\mathbb{G} \xrightarrow{\mathbf{a}}_{\mathsf{ogs}} \mathbb{H}$ then $\mathbb{G}$ is valid.*