Reachability Types, Traces and Full Abstraction

Benedict Bunting
University of Oxford, UK

Andrzej S. Murawski
University of Oxford, UK

***Abstract*—Reachability types are a recent approach to modelling sharing in higher-order languages, aiming to provide separation guarantees through typability. The contextual equivalence problem in such a setting is exacerbated by the need to consider reachability-related constraints on the allowable interactions. In particular, they might weaken the ability of contexts to observe sequentiality.**

In this paper, we investigate contextual equivalence for reachability types through the lens of operational game semantics. We provide a sound trace model for a language equipped with reachability types, and show how to refine it to a fully abstract one, which captures a natural notion of equivalence based on allowing terms to share functions and locations consistently with the assigned reachability annotations. We also discuss the corresponding problem of contextual approximation, along with an inequational full abstraction result.

This is a first attempt at defining a fully abstract semantics for reachability types.

***Index Terms*—reachability types, game semantics, trace models, full abstraction**

I. Introduction

Type systems that offer control over sharing are seen as a promising technique for improving program safety and performance. This has been demonstrated by the recent success of Rust, whose core is based on the *shared XOR mutable* principle, i.e. sharing is restricted to immutable variables. This policy turns out to be quite restrictive when it comes to expressing common programming patterns involving higher-order functions and state, like those expressible in languages such as ML or Scala. Reachability types [1, 2] are a recent proposal to address this gap and provide a type system that is capable of collecting information about sharing as well as lack thereof, i.e. separation.

The key idea of reachability types is to track reachable variables/locations by annotating types with *type qualifiers*, which contain functions or locations that may be reachable from a given term. For example, the term $h \triangleq \mathbf{let}\, r = \mathbf{ref}\,(0)\ \mathbf{in}\ \lambda f.\lambda x.(!\ell + f(!r) + g(x))$, where ℓ is a memory location, ! stands for dereferencing, $\tau \triangleq \mathsf{Int} \to \mathsf{Int}$ and $g : \tau$ is free, would normally be typed as $\tau \to \tau$. With reachability types, it can be given the more accurate type below.

$$(\mu h.(f : (\mathsf{Int} \to \mathsf{Int})^{Q}) \to (\mathsf{Int} \to \mathsf{Int})^{\{f,g,h,\ell\}})^{\{g,\ell\}}$$

The presence of g, ℓ in qualifiers indicates that both the whole term and its functional result may directly reach the unknown function g and location ℓ. In contrast, f corresponds to a dependency on resources contributed by the input. h in turn is a *self-reference*, which allows one to express the fact that the functional result of h may reach (private) locations (such as r) created by h itself. In addition, the argument type τ^{Q} can be used to specify the degree of overlap between h and its arguments. Setting Q to $\emptyset$ corresponds to demanding that what the argument can reach must be disjoint from what h can reach, while $\{\ell\}$ would allow for scenarios in which the argument may also reach ℓ, but not g. Similarly, $Q = \{\ell, g\}$ would permit h to share ℓ, g with its arguments. Overall, the idea of tracking reachability at the type level turns out very powerful and can be used to express many common programming scenarios such as non-interference, non-escaping, non-accessibility and scoped borrowing [1].

The use of qualifiers in reachability types complicates arguments about contextual equivalence, such as may be needed justify reachability-type-based program transformations and optimisations [3]. For example, whether the term $\mathbf{let}\ f = h()\ \mathbf{in}\ (\mathbf{let}\, g = h()\, \mathbf{in}\, f(); g())$ is equivalent to $\mathbf{let}\ f = h()\ \mathbf{in}\ (\mathbf{let}\, g = h()\, \mathbf{in}\, g(); f())$ depends on how $h : \mathsf{Unit} \to (\mathsf{Unit} \to \mathsf{Unit})$ is typed. If $h : (\mu h.\mathsf{Unit} \to (\mathsf{Unit} \to \mathsf{Unit})^{\{h\}})^{\emptyset}$ then f, g may share the private state of h and the terms will not be equivalent, because the order of calls can be detected through the state. In contrast, for $h : (\mu h.\mathsf{Unit} \to (\mathsf{Unit} \to \mathsf{Unit})^{\emptyset})^{\emptyset}$ the terms will behave in the same way.

In this paper, we employ operational game semantics [4, 5] to provide a formal and precise account of the underpinning theory of contextual equivalence (and the associated notion of approximation). Game semantics is already known for providing a wide range of fully abstract models for various programming paradigms [6, 7]. They rely on representing interactions between programs and contexts as a dialogue between two players: O (context) and P (program). A characteristic feature of operational game semantics is that these dialogues are generated as traces of a carefully crafted labelled transition system (LTS), which uses names to represent unknown functions in the spirit of open/normal-form bisimulation [8, 9]. In addition, to capture the sharing of state, traces of our LTS are decorated with sets of abstract states revealed by the environment.

In the example equivalence above, the first term is represented by traces like $\overline{h^l}((), c_1)\ \ c_1(f^m)\ \ \overline{h^l}((), c_3)\ \ c_3(g^n)\ \ \overline{f^m}((), c_4)\ \ c_4(())\ \ \overline{g^n}((), c_5)\ \ c_5(())\ \ \overline{c^l}(())$. In this trace, *names* such as f, g are used to abstractly represent the functions returned by the context for the calls to h, while l, m, n are *abstract states*. In this way, the trace captures the way the term uses these functions. The distinctiveness of l, m, n captures formally the fact f, g, h may not share state of h, argued intuitively above. The other term is represented by $\overline{h^l}((), c_1)$

For the purpose of Open Access, the author has applied a CC BY public copyright licence to any Author Accepted Manuscript (AAM) version arising from this submission. This work was supported in part by the Engineering and Physical Sciences Research Council [EP/W524311/1, studentship 2742896]

$c_1(f^m)$ $\overline{h^l}((),c_3)$ $c_3(g^n)$ $\overline{g^n}((),c_5)$ $c_5(())$ $\overline{f^m}((),c_4)$ $c_4(())$ $\overline{c^l}(())$, which captures the difference in the order of the calls.

In order to demonstrate our approach, in Section II we introduce a minimalistic language with reachability types, modelled after λ^* [1]. As our methods exploit the ability to η-expand terms, the calculus is based on a notion of *well-behaved* types, for which η-expansion is guaranteed to be type-preserving. In Sections III and IV, we introduce the LTS $\mathcal{L}_P$, which is sound. The technical concepts underpinning the analysis of the model and the soundness result are covered in Sections V and VI. In particular, the LTS non-trivially adapts the classic notion of visibility (originally used to characterise functions that are reachable/visible to players in a language with first-order references) [10] to the setting of reachability types. Intuitively, this is done by identifying a family of subtle technical conditions that further restrict visibility to make it compatible with types. In Section VII, building on a definability result, we refine the trace model to a fully abstract one by introducing rearrangement relations. We show that one can capture contextual equivalence via complete trace equivalence up to allowable trace permutations. For example, these permutations equate the traces for the two terms in the equivalence above. Program approximation in turn can be characterised through an ordering that allows one to omit certain actions. For example, it turns out that the term **let** $f = h()$ **in** $f(1);f(2)$ approximates **let** $f = h()$ **in** $f(1)$ when $h : (\mu h.\mathsf{Unit} \to (\mathsf{Int} \to \mathsf{Unit})^{\emptyset})^{\emptyset}$, because $f(1)$ and $f(2)$ cannot reach any state other than f's private one and, consequently, $f(2)$ does not interfere with the subsequent computation.

At a high level, we see our work as the beginning of a semantic study of reachability types. So far, the most closely related work in this direction is [11], which provides a sound logical relation for contextual equivalence.

II. Language

We shall study λ^*_{wb}, a simple ML-like language, which is a variant of λ^* [1] inspired by the formalisation [12] and notation of [11]. We present its syntax in Figure 1. At its core, it is a λ-calculus enriched with constants of base type, primitive operations, conditionals, integer-valued references and recursion. It has a standard, small-step call-by-value operational semantics, expressed as a reduction relation $(M,h) \to (M',h')$ between pairs of terms and heaps (mapping locations to integers). This uses ***evaluation contexts***, K, to identify the next redex. We write $(M,h) \Downarrow$ to mean $(M,h) \to^* (V,h')$ for some value V (i.e. it terminates).

Types What makes this language interesting is its types. They are denoted by τ, with σ being used for ***qualified types*** τ^Q, in which the annotation Q is referred to as a ***qualifier***. In our case, qualifiers are either $\bot$ (for base types) or a set of captured variables or locations (for references and functions). Functions are given a dependent type of the form $\mu f.(x : \tau_1^{Q_1}) \to \tau_2^{Q_2}$. Here x is the argument to the function. When τ_1 is a reference or function type, the argument could contribute locations/functions to the result. Accordingly, if $Q_1 \neq \bot$ then x may appear in Q_2, but not in τ_2 or $\tau_1^{Q_1}$. f is called the ***self-reference***, which signifies dependence of the result on the function: it may appear in Q_2, but not in τ_2 or $\tau_1^{Q_1}$. For clarity, when x does not appear in Q_2, such as when it is of base type, we may omit it ($\mu f.\tau_1^{Q_1} \to \tau_2^{Q_2}$). Similarly, we will often drop the binding when the self-reference is unused ($(x : \tau_1^{Q_1}) \to \tau_2^{Q_2}$). If neither is used, we simply write $\tau_1^{Q_1} \to \tau_2^{Q_2}$.

The qualifiers can be partially ordered by $Q_1 \sqsubseteq Q_2 \triangleq Q_1 = \bot \vee Q_1 \subseteq Q_2$, which essentially lifts the subset ordering to include $\bot$ as the least element. Joins ($\sqcup$) can then be computed by $\bot \sqcup Q \triangleq Q$, $Q \sqcup \bot \triangleq Q$, and $Q_1 \sqcup Q_2 \triangleq Q_1 \cup Q_2$ (otherwise), with meets ($\sqcap$) given by $\bot \sqcap Q \triangleq \bot$, $Q \sqcap \bot \triangleq \bot$, and $Q_1 \sqcap Q_2 \triangleq Q_1 \cap Q_2$ (otherwise).

We will be concerned with types that satisfy a certain well-formedness condition.

Definition 1. A qualified type σ is ***well-behaved*** if $\sigma = \beta^Q$ or $\sigma = \mathsf{Ref}^Q$ or $\sigma = \mu f.(x : \tau_1^{Q_1}) \to \tau_2^{Q_2})^{Q_f}$, where $\tau_1^{Q_1}, \tau_2^{Q_2}$ are well-behaved, $Q_1 \sqsubseteq Q_f$, $Q_2 \sqsubseteq Q_f \sqcup \{f,x\}$, and $x \in Q_2$ implies $Q_1 \sqsubseteq Q_2$. We shall write $wb(\sigma)$ to insist that σ be well-behaved.

These conditions reflect the intuitive reading of qualified types. In the type $(\mu f.(x : \tau_1^{Q_1}) \to \tau_2^{Q_2})^{Q_f}$, Q_1 represents the permitted overlap of the argument with the function, and so should be within what is observed by the function. Similarly, Q_2 expresses what the result of the function might observe, which must be either observed by the function (Q_f), the argument (x), or something not observable from outside the function (f). All examples in [1] are well-behaved.

Typing rules We will have typing judgments of the form $\Sigma;\Gamma \vdash^{\varphi} M : \sigma$. Σ denotes a heap typing environment, written $\ell_1 : \mathsf{Ref}, \cdots, \ell_n : \mathsf{Ref}$. For variables, we will have type assignments **:**, which have either the standard form : (used to type arguments) or ⁚ for the self-references used when typing functions. Thus, Γ is a typing environment of the form $x_1 \mathbf{:} \tau_1^{Q_1}, \cdots, x_m \mathbf{:} \tau_m^{Q_m}$, where we forbid circular dependencies by stipulating that $\tau_i^{Q_i}$ may contain $\ell \in \Sigma$ and x_j provided $Q_j \neq \bot$ and $j < i$. We require $wb(\sigma)$ for any entry $x \mathbf{:} \sigma \in \Gamma$. The final element of the typing judgment is φ, called the ***filter***. Let us write $\mathrm{dom}_+(\Sigma;\Gamma)$ to be all locations in Σ and variables in Γ given non-$\bot$ qualifiers. Then $\varphi \subseteq \mathrm{dom}_+(\Sigma;\Gamma)$ and φ can be seen as specifying the locations and variables that can be observed directly by the term. In the case that $\varphi = \mathrm{dom}_+(\Sigma;\Gamma)$, then we will omit φ, and write simply $\Sigma;\Gamma \vdash M : \sigma$. To ease the presentation of the rules, we will occasionally write things like $\Sigma;\Gamma \vdash^{\varphi \cup \{x\}} M : \sigma$, where x may have a $\bot$-qualifier in Γ. In such cases, we consider x to be implicitly removed. We also write $\Sigma;\Gamma \vdash^{\varphi} M_1, M_2 : \sigma$ as shorthand for a pair of judgments for M_1 and M_2.

The type system is given in Figure 1. An important rule is T-App, which controls the separation between a function and its argument. ***Saturation***, written $Q*$, identifies variables and locations reachable from a qualifier. In an application, the intersection of the saturated qualifiers of the function and argument is compared with the permitted overlap, ensuring the

argument only reaches things that are permitted, or unobserved by the function. The rule T-SELF implements self-references, allowing the qualifier of an abstraction to be replaced by the self-reference while typing the function body. Sub-typing is included, allowing variation of qualifiers, with the usual contra-variance in function arguments.

Example 2. We write $\mathsf{let}\, x = M \,\mathsf{in}\, N$ for $(\lambda x.N)M$. As in [1], one can derive the following rule for $\mathsf{let}\, x = M \,\mathsf{in}\, N$.

T-LET
$$\frac{\Sigma;\Gamma \vdash^{}_{\varphi} M_1 : \tau_1^{Q_1} \quad \Sigma;\Gamma, x : \tau_1^{Q_1} \vdash^{}_{\varphi\cup\{x\}} M_2 : \tau_2^{Q_2} \quad wb(\tau_2^{Q_2}) \quad x \notin \mathrm{FV}(\tau_2)}{\Sigma;\Gamma \vdash^{}_{\varphi} \mathsf{let}\, x = M_1 \,\mathsf{in}\, M_2 : \tau_2^{Q_2\{Q_1/x\}}}$$

For example, $\vdash \mathsf{let}\, r = \mathsf{ref}\,\widehat{0}\,\mathsf{in}\,\lambda x.r : (\mu f.\mathsf{Unit}^{\perp} \to \mathsf{Ref}^{\{f\}})^{\emptyset}$.

Properties λ^*_{wb} enjoys a number of desirable properties. The first is an 'open' version of type preservation [1].

Lemma 3 (Type Preservation)**.** *If $h : \Sigma$, $\Sigma;\Gamma \vdash^{}_{\varphi} M : \tau^Q$, and $(M, h) \to (M', h')$, then there exists Σ' such that $h' : \Sigma'$, $L = \mathrm{dom}(\Sigma') \setminus \mathrm{dom}(\Sigma)$, and $\Sigma';\Gamma \vdash^{}_{\varphi+L} M' : \tau^{Q+L}$, where $\perp + L \triangleq \perp$ and $Q + L \triangleq Q \sqcup L$ for $Q \neq \perp$.*

A key property of well-behaved types is the ability to eta-expand a value while preserving its type.

Lemma 4. *Given a value V and type τ, define the **full eta-expansion** $\eta_\tau(V)$ by $\eta_\tau(V) \triangleq V$ for $\tau \in \{\beta, \mathsf{Ref}\}$ and $\eta_\tau(V) \triangleq \lambda x.V\,\eta_{\tau_1}(x)$, where x is fresh and $\tau = \mu f.(y : \tau_1^{Q_1}) \to \sigma$. If $wb(\tau^Q)$ and $\Sigma;\Gamma \vdash^{}_{\widehat{Q}} V : \tau^Q$ then $\Sigma;\Gamma \vdash^{}_{\widehat{Q}} \eta_\tau(V) : \tau^Q$, where $\widehat{\perp} \triangleq \emptyset$ and $\widehat{Q} \triangleq Q$ otherwise.*

This property is necessary when trying to construct a model based upon operational game semantics, as this involves applying functions passed between the two players with fresh values, which is essentially what is occurring in full eta-expansion. The Lemma also provides enough flexibility to prove a definability result in Section VII.

In what follows, when we write $\Gamma \vdash M : \sigma$, we intend to refer to judgments in which Γ contains no $\circ$ entries (and so represents 'top-level' typings) and $wb(\sigma)$.

Contextual Testing The objects of our study are contextual approximation and equivalence, defined by testing termination of terms in every possible context. As λ^*_{wb} makes explicit the locations available to the context, it becomes natural to define possible contexts as being allowed additional locations, so long as they are consistent with the qualifier annotations. We formalise this below.

Definition 5. We write $\Sigma;\Gamma \vdash \sigma$ if all non-$\perp$ qualifiers in σ refer to $\mathrm{dom}(\Sigma) \cup \mathrm{dom}_+(\Gamma)$. Let $\Gamma = x_1 : \tau_1^{Q_1}, \cdots, x_m : \tau_m^{Q_m}$. We say that $I \subseteq \{i \mid x_i \in \mathrm{dom}_+(\Gamma)\}$ is ***eligible for enrichment*** if $X \neq \emptyset$ and $X \sqcap Y = \emptyset$ for $X = \bigsqcap_{i\in I}(x_i*)$ and $Y = \bigsqcup_{i\notin I}(x_i*)$. Given such I, let Γ_I be Γ in which, for each $1 \leq i \leq m$, we add fresh ℓ to every qualifier in $\tau_i^{Q_i}$ that already contains x_j for some $j \in I$, and to Q_i if $i \in I$. σ_I is defined analogously. We then write $(\Sigma;\Gamma \vdash \sigma) \prec ((\Sigma, \ell : \mathsf{Ref});\Gamma_I \vdash \sigma_I)$ for one-step enrichment, and $(\Sigma;\Gamma \vdash \sigma) \preceq (\Sigma';\Gamma' \vdash \sigma')$ for its reflexive and transitive closure.

$\preceq$ captures the typings that can be obtained by extending the environment with additional locations in a way which satisfies the separation imposed by the original typing.

Lemma 6. *If $\Sigma;\Gamma \vdash \sigma \preceq \Sigma';\Gamma' \vdash \sigma'$ then $\Sigma;\Gamma \vdash M : \sigma$ implies $\Sigma';\Gamma' \vdash M : \sigma'$.*

Definition 7 (Context Typing)**.** We type a context C as $\Sigma;\Gamma \vdash C : (\Sigma';\Gamma' \vdash \sigma') \Rightarrow \sigma$ when for every $\Sigma';\Gamma' \vdash M : \sigma'$, we can type $\Sigma;\Gamma \vdash C[M] : \sigma$ in such a way that the typing derivation of M is unchanged.

Contextual testing is then defined over all possible contexts which respect the imposed overlaps.

Definition 8 (Contextual Approximation and Equivalence)**.** Given $\Gamma \vdash M_1, M_2 : \sigma$, we define $\Gamma \vdash M_1 \lesssim_{ctx} M_2 : \sigma$ to hold, when for all $\Gamma \vdash \sigma \preceq \Sigma';\Gamma' \vdash \sigma'$, $h : \Sigma$ and contexts $\Sigma' \vdash C : (\Sigma';\Gamma' : \sigma') \Rightarrow \mathsf{Unit}$, if $(C[M_1], h) \Downarrow$ then $(C[M_2], h) \Downarrow$. We say $\Gamma \vdash M_1 \simeq_{ctx} M_2 : \sigma$ when $\Gamma \vdash M_1 \lesssim_{ctx} M_2 : \sigma$ and $\Gamma \vdash M_2 \lesssim_{ctx} M_1 : \sigma$

Remark 9. We consider our definition of contextual equivalence to be natural, in the sense that not enriching the typing judgments with locations leads to strange phenomena. Consider $\tau = \mathsf{Unit}^{\perp} \to \mathsf{Unit}^{\perp}$, the context $\Gamma = h\colon \tau^{\emptyset}, f : \tau^{\{h\}}, g\colon \tau^{\{h\}}$ and terms $\Gamma \vdash f(); g() : \mathsf{Unit}^{\perp}$ and $\Gamma \vdash g(); f() : \mathsf{Unit}^{\perp}$. The qualifiers of f and g indicate that they may communicate through h, so it is natural to want contextual equivalence to distinguish these two terms. However, a notion of contextual equivalence that does not provide the extra location to h would equate them. This is because of h's type $((\mathsf{Unit}^{\perp} \to \mathsf{Unit}^{\perp})^{\emptyset})$, which cannot be used to transmit any meaningful information between f and g, apart from generating divergence after a number of calls. But this does not suffice to separate $f(); g()$ from $g(); f()$ even if both f and g use h, because both terms generate the same number of calls to h. In contrast, with an extra location available to h, the terms can be distinguished.

Surprisingly, changing the type of h to $(\mathsf{Unit}^{\perp} \to \mathsf{Bool}^{\perp})^{\emptyset}$ does allow one to differentiate between the terms (without extra locations), because a function at this type could communicate whether or not it has been called for the first time, which could then be used to separate the terms. Consequently, without extra locations provided to h, the equivalence of terms depends on the *type* (not *qualifier*) of variables not appearing in it, and we view this non-uniformity as undesirable. In any case, the notion proposed above is more discriminating than one without extra locations, i.e. our results would then amount to soundness rather than full abstraction.

Note that our typing judgments are economical in that, in general, they do not require that qualifiers Q be equal to the full reachability sets $Q*$, e.g. variables can be typed as $\tau^{\{x\}}$. While this leads to more concise annotations, some of our results assume saturation.

Syntax

Base Types	β	$\triangleq$	$\mathsf{Unit} \mid \mathsf{Bool} \mid \mathsf{Int}$
Types	τ	$\triangleq$	$\beta \mid \mathsf{Ref} \mid \mu f.(x:\sigma_1) \to \sigma_2$
Qualified Types	σ	$\triangleq$	$\beta^{\perp} \mid \mathsf{Ref}^{\alpha} \mid (\mu f.(x:\sigma_1) \to \sigma_2)^{\alpha}$ $\qquad Q, R \triangleq \perp \mid \alpha$ $\qquad \alpha \in \mathcal{P}_{\mathrm{fin}}(\mathbf{Var} \cup \mathbf{Loc})$
Values	U, V	$\triangleq$	$() \mid \mathbf{tt} \mid \mathbf{ff} \mid \widehat{n} \mid x \mid \ell \mid \lambda x.M \mid \mathbf{rec}\ y(x).M$
Terms	M, N	$\triangleq$	$V \mid MN \mid \mathbf{ref}\ M \mid !M \mid M := N \mid \mathbf{if}\ M_1\ \mathbf{then}\ M_2\ \mathbf{else}\ M_3 \mid M \oplus N \mid M \boxdot N$
Eval. Ctxt.	K	$\triangleq$	$\bullet \mid VK \mid KM \mid \mathbf{ref}\ K \mid !K \mid V := K \mid K := M \mid \mathbf{if}\ K\ \mathbf{then}\ M\ \mathbf{else}\ N$ $\mid K \oplus M \mid V \oplus K \mid K \boxdot M \mid V \boxdot K$
Contexts	C	$\triangleq$	$\bullet \mid \lambda x.C \mid \mathbf{rec}\ y(x).C \mid MC \mid CM \mid \mathbf{ref}\ C \mid !C \mid C := M \mid M := C \mid \mathbf{if}\ C\ \mathbf{then}\ M\ \mathbf{else}\ N$ $\mid \mathbf{if}\ M\ \mathbf{then}\ C\ \mathbf{else}\ N \mid \mathbf{if}\ M\ \mathbf{then}\ N\ \mathbf{else}\ C \mid C \oplus M \mid M \oplus C \mid C \boxdot M \mid M \boxdot C$

Notational conventions: $x, y \in \mathbf{Var}$, $\ell \in \mathbf{Loc}$, $n \in \mathbb{Z}$, $i \in \{1, 2\}$, $\oplus \in \{+, -, *\}$, $\boxdot \in \{=, <\}$

Qualifier operations

$\perp \sqcup Q \triangleq Q$, $Q \sqcup \perp \triangleq Q$, $Q_1 \sqcup Q_2 \triangleq Q_1 \cup Q_2$ (otherwise) $\qquad \perp + Q \triangleq \perp$, $Q + Q' \triangleq Q \sqcup Q'$ (otherwise)

$\perp \sqcap Q \triangleq \perp$, $Q \sqcap \perp \triangleq \perp$, $Q_1 \sqcap Q_2 \triangleq Q_1 \cap Q_2$ (otherwise) $\qquad \perp + f \triangleq \perp$, $Q + f \triangleq Q \cup \{f\}$ (otherwise)

$\perp\{Q/x\} \triangleq \perp$, $Q_1\{Q_2/x\} \triangleq (Q_1 \setminus \{x\}) \sqcup Q_2$ if $x \in Q_1$ and Q_1 otherwise $\qquad \widehat{\perp} \triangleq \emptyset$, $\widehat{Q} \triangleq Q$ (otherwise)

Term typing

$$\text{T-Unit}\ \frac{}{\Sigma;\Gamma \vdash_{\varphi} () : \mathsf{Unit}^{\perp}} \qquad \text{T-Int}\ \frac{}{\Sigma;\Gamma \vdash_{\varphi} \widehat{n} : \mathsf{Int}^{\perp}} \qquad \text{T-Bool}\ \frac{t \in \{\mathbf{tt}, \mathbf{ff}\}}{\Sigma;\Gamma \vdash_{\varphi} t : \mathsf{Bool}^{\perp}} \qquad \text{T-Var}\perp\ \frac{x : \tau^{\perp} \in \Gamma}{\Sigma;\Gamma \vdash_{\varphi} x : \tau^{\perp}} \qquad \text{T-Var}\ \frac{x : \tau^{Q} \in \Gamma \qquad x \in \varphi}{\Sigma;\Gamma \vdash_{\varphi} x : \tau^{\{x\}}}$$

$$\text{T-Loc}\ \frac{\ell : \mathsf{Ref} \in \Sigma \qquad \ell \in \varphi}{\Sigma;\Gamma \vdash_{\varphi} \ell : \mathsf{Ref}^{\{\ell\}}} \qquad \text{T-Ref}\ \frac{\Sigma;\Gamma \vdash_{\varphi} M : \mathsf{Int}^{\perp}}{\Sigma;\Gamma \vdash_{\varphi} \mathbf{ref}\ M : \mathsf{Ref}^{\emptyset}} \qquad \text{T-Assign}\ \frac{\Sigma;\Gamma \vdash_{\varphi} M_1 : \mathsf{Ref}^{Q} \qquad \Sigma;\Gamma \vdash_{\varphi} M_2 : \mathsf{Int}^{\perp}}{\Sigma;\Gamma \vdash_{\varphi} M_1 := M_2 : \mathsf{Unit}^{\perp}} \qquad \text{T-Deref}\ \frac{\Sigma;\Gamma \vdash_{\varphi} M : \mathsf{Ref}^{Q}}{\Sigma;\Gamma \vdash_{\varphi} !M : \mathsf{Int}^{\perp}}$$

$$\text{T-OPlus}\ \frac{\Sigma;\Gamma \vdash_{\varphi} M_1, M_2 : \mathsf{Int}^{\perp}}{\Sigma;\Gamma \vdash_{\varphi} M_1 \oplus M_2 : \mathsf{Int}^{\perp}} \qquad \text{T-Ineq}\ \frac{\Sigma;\Gamma \vdash_{\varphi} M_1, M_2 : \mathsf{Int}^{\perp}}{\Sigma;\Gamma \vdash_{\varphi} M_1 \boxdot M_2 : \mathsf{Bool}^{\perp}} \qquad \text{T-If}\ \frac{\Sigma;\Gamma \vdash_{\varphi} N : \mathsf{Bool}^{\perp} \qquad \Sigma;\Gamma \vdash_{\varphi} M_1, M_2 : \tau^{Q}}{\Sigma;\Gamma \vdash_{\varphi} \mathbf{if}\ N\ \mathbf{then}\ M_1\ \mathbf{else}\ M_2 : \tau^{Q}}$$

$$\text{T-Abs}\ \frac{\tau = \mu f.(x : \tau_1^{Q_1}) \to \tau_2^{Q_2} \qquad (\Sigma;\Gamma, x : \tau_1^{Q_1},\ f \circ \tau^{Q_f}) \vdash_{\overline{Q_f \cup \{f,x\}}} M : \tau_2^{Q_2} \qquad Q_f \sqsubseteq \varphi \qquad f \notin \mathrm{FV}(M)}{\Sigma;\Gamma \vdash_{\varphi} \lambda x.M : \tau^{Q_f}}$$

$$\text{T-Fix}\ \frac{\tau = \mu g.(x : \tau_1^{Q_1}) \to \tau_2^{Q_2} \qquad (\Sigma;\Gamma, x : \tau_1^{Q_1},\ f \circ \tau^{Q_f}) \vdash_{\overline{Q_f \cup \{f,x\}}} M : \tau_2^{Q_2\{f/g\}} \qquad Q_f \sqsubseteq \varphi}{\Sigma;\Gamma \vdash_{\varphi} \mathbf{rec}\ f(x).M : \tau^{Q_f}}$$

$$\text{T-App}\ \frac{\Sigma;\Gamma \vdash_{\varphi} M_1 : (\mu f.(x : \tau_1^{Q}) \to \tau_2^{Q_2})^{Q_f} \qquad wb(\tau_1^{Q_1}) \qquad \Sigma;\Gamma \vdash_{\varphi} M_2 : \tau_1^{Q_1} \qquad Q_1 * \sqcap Q_f * \sqsubseteq Q * \qquad Q_2 \sqsubseteq \varphi, x, f}{\Sigma;\Gamma \vdash_{\varphi} M_1\, M_2 : \tau_2^{Q_2\{Q_1/x\}\{Q_f/f\}}}$$

$$\text{T-Sub}\ \frac{\Sigma;\Gamma \vdash_{\varphi} M : \tau_1^{Q_1} \qquad \Sigma;\Gamma \vdash \tau_1^{Q_1} <: \tau_2^{Q_2} \qquad Q_2 \sqsubseteq \varphi}{\Sigma;\Gamma \vdash_{\varphi} M : \tau_2^{Q_2}} \qquad \text{T-Self}\ \frac{\Sigma;\Gamma \vdash_{\varphi} M : \tau^{Q + Q_f} \qquad f \circ \tau_f^{Q_f} \in \Gamma \qquad f \in \varphi}{\Sigma;\Gamma \vdash_{\varphi} M : \tau^{Q+f}}$$

$$\text{T-Weakening}\ \frac{\Sigma;\Gamma \vdash_{\varphi} M : \tau^{Q} \qquad \varphi' = \mathrm{dom}_{+}(\Sigma', \Gamma')}{\Sigma, \Sigma';\Gamma, \Gamma' \vdash_{\varphi \cup \varphi'} M : \tau^{Q}} \qquad \text{T-Exchange}\ \frac{\Sigma;\Gamma, x : \sigma, y : \sigma', \Gamma' \vdash_{\varphi} M : \tau^{Q}}{\Sigma;\Gamma, y : \sigma', x : \sigma, \Gamma' \vdash_{\varphi} M : \tau^{Q}}$$

Subtype

$$\text{S-Trans}\ \frac{\Sigma;\Gamma \vdash \sigma_1 <: \sigma_2 \qquad \Sigma;\Gamma \vdash \sigma_2 <: \sigma_3}{\Sigma;\Gamma \vdash \sigma_1 <: \sigma_3} \qquad \text{S-Base}\ \frac{}{\Sigma;\Gamma \vdash B^{\perp} <: B^{\perp}} \qquad \text{S-Ref}\ \frac{Q_1 \sqsubseteq Q_2 \sqsubseteq \mathrm{dom}_{+}(\Sigma;\Gamma)}{\Sigma;\Gamma \vdash \mathsf{Ref}^{Q_1} <: \mathsf{Ref}^{Q_2}}$$

$$\text{S-Fun}\ \frac{Q_5 \sqsubseteq Q_6 \sqsubseteq \mathrm{dom}_{+}(\Sigma;\Gamma) \qquad \Sigma;\Gamma \vdash \sigma_3 <: \sigma_1 \qquad \Sigma;\Gamma, f \circ (\mu f.(x : \sigma_1) \to \sigma_2)^{Q_5}, x : \sigma_3 \vdash \sigma_2 <: \sigma_4}{\Sigma;\Gamma \vdash (\mu f.(x : \sigma_1) \to \sigma_2)^{Q_5} <: (\mu f.(x : \sigma_3) \to \sigma_4)^{Q_6}}$$

Reachability and saturation

$\Sigma;\Gamma \vdash x \rightsquigarrow y \Leftrightarrow x : \tau^{Q} \in \Sigma;\Gamma, y \in Q$ $\qquad \Sigma;\Gamma \vdash x* \triangleq \{y \mid x \rightsquigarrow^{*} y\}$ $\qquad \Sigma;\Gamma \vdash \perp* \triangleq \perp$ $\qquad \Sigma;\Gamma \vdash Q* \triangleq \bigcup_{x \in Q} x*$

Fig. 1: The system λ^{*}_{wb}

Definition 10. If $\Sigma;\Gamma \vdash \sigma$ is a type, we write $\Sigma;\Gamma \vdash \sigma*$ for the type obtained by replacing every qualifier Q in σ by $Q*$. The notation can be extended to an environment $\Sigma;\Gamma$ pointwise. A judgment $\Sigma;\Gamma \vdash M : \sigma$ is ***saturated*** if $\sigma = \Sigma;\Gamma \vdash \sigma*$ and $\Sigma;\Gamma = (\Sigma;\Gamma)*$.

The saturation procedure preserves the essentials of reachability types: separation between functions and arguments, and dependence of the result on the argument or self-reference.

Lemma 11. *We have that* $\Gamma \vdash M_1, M_2 : \sigma$ *implies* $\Gamma* \vdash M_1, M_2 : \sigma*$*, and* $\Gamma* \vdash M_1 \lesssim_{ctx} M_2 : \sigma*$ *implies* $\Gamma \vdash M_1 \lesssim_{ctx} M_2 : \sigma$.

The full abstraction results presented in the paper will apply to all saturated judgments $\Gamma \vdash M : \sigma$ with one caveat. As in [5], we will assume that Ref does not occur in Γ or σ, and call such judgments ***r-free***. We stress that this is just a restriction on the boundary (Γ and σ only), and subterms of M may contain arbitrary types and uses of **ref** . We impose the restriction due to the complications that arise in fully abstract modelling of reference-based interfaces [13]. They would be alleviated somewhat, if contexts could store references [4, 14], but then the reachability-type framework is more involved [1]. Consequently, the r-free case is a reasonable compromise for a first full abstraction result in the area.

III. LTS (BASIC NOTIONS)

In this section we set out a trace semantics for reachability types, which will lead to a full abstraction result. The model is built in the tradition of operational game semantics, i.e. we define an LTS whose traces provide an abstract account of interactions between the program (P, for Proponent) and its environment (O, for Opponent). The interactions will consist of sequences of actions that involve two sets of names.

Definition 12. Let FNames, CNames be countably infinite disjoint sets of ***function*** and ***continuation names*** respectively. We set $\mathrm{Names} \triangleq \mathrm{FNames} \uplus \mathrm{CNames}$.

Elements of Names will appear in structures throughout this work, and so $\nu(Z)$ refers to the set of names used in some entity Z. We will use f, c (and variants) to range over FNames and CNames respectively. We will now discuss the actions that will feature in traces. They will have a polarity, either O or P, depending on who plays them, and will be either a *question*, corresponding to a function call, or an *answer*, corresponding to returning a value. The values that appear in such calls and returns will be generated by the grammar $A \triangleq f \mid () \mid \widehat{n} \mid \mathbf{tt} \mid \mathbf{ff}$, and will be called ***abstract values***. Altogether we have the four types of actions detailed below.

Definition 13. An ***action***, written generically as $\mathbf{a}$, has one of the forms:

- ***Opponent Question*** (OQ) $f(A, c)$
- ***Player Question*** (PQ) $\overline{f}(A, c)$
- ***Opponent Answer*** (OA) $c(A)$
- ***Player Answer*** (PA) $\overline{c}(A)$

We write $\check{f}(A, c)$ to mean either $f(A, c)$ or $\overline{f}(A, c)$, and $\check{c}(A)$ similarly for $c(A)$ or $\overline{c}(A)$. We refer to f in $\check{f}(A, c)$, and c in $\check{c}(A)$ as the ***head names*** of $\mathbf{a}$, whereas any other names in an action are said to be ***introduced*** by the action. OQ and OA are O-actions, while PQ and PA are P-actions. We will often write X to refer to a specific player $X \in \{O, P\}$. Then $\overline{X}$ stands for the opposite player, i.e. $\overline{O} = P$ and $\overline{P} = O$.

The question actions correspond to calls to the function f with argument A whose result will be passed to continuation c, whereas the answer actions correspond to returning the result A to the continuation c. The sequences of actions that will constitute traces need to satisfy a number of technical conditions. First of all, they have to be typable. To formalise this, we start off with a notion of initial names and initial typing, which will provide an initial supply of typed names. In what follows we assume that free functional variables occurring in types are drawn from FNames.

Definition 14. An ***initial typing*** is a pair of partial functions (N_O, N_P), each mapping a subset of Names to qualified types such that function names are mapped to qualified function types; $\mathrm{dom}(N_O)$, $\mathrm{dom}(N_P)$ are finite and disjoint; for all $X \in \{O, P\}$, we have $\nu(\mathrm{img}(N_X)) \subseteq \mathrm{dom}(N_O) \cup \mathrm{dom}(N_P)$. We abuse notation to treat N_X as a set, writing N_X for $\mathrm{dom}(N_X)$.

Remark 15. Given an initial typing (N_O, N_P) and $Q \subseteq \mathrm{Names}$, we can define $Q^*_{N_O,N_P} \subseteq \mathrm{Names}$ in a manner analogous to saturation with respect to a typing context. That is $x \rightsquigarrow_{N_O,N_P} y \Leftrightarrow (N_O(x) = \tau^Q \vee N_P(x) = \tau^Q) \wedge y \in Q$, $x^*_{N_O,N_P} \triangleq \{y \mid x \rightsquigarrow^*_{N_O,N_P} y\}$ and $Q^*_{N_O,N_P} \triangleq \bigcup_{x \in Q} x^*_{N_O,N_P}$.

We now define what it means to be a basic trace.

Definition 16. Let (N_O, N_P) be an initial typing. An (N_O, N_P)-***trace*** is a sequence t of actions such that:

- actions alternate between P and O actions;
- no name is introduced twice and names from $N_O \cup N_P$ are never introduced;
- for any X-action $\mathbf{a}$ with head name d, we have $d \in N_{\overline{X}}$ or d must be introduced in t in an earlier action by $\overline{X}$;
- there exists a typing map Ty such that $\mathrm{dom}(\mathrm{Ty}) = N_O \cup N_P \cup \nu(t)$, $\mathrm{Ty}(d) = N_X(d)$ for $d \in N_X$ and, for any action of the form $\check{f}(A, c)$, if $\mathrm{Ty}(f) = (\mu g.(x : \tau_1^{Q_1}) \to \tau_2^{Q_2})^Q$ then $\mathrm{Ty}(A) = \tau_1^{Q_1}$ and $\mathrm{Ty}(c) = \tau_2^{Q_2}\{f/g\}\{\nu(A)/x\}$, and, similarly, for any action of the form $\check{c}(A)$, if $\mathrm{Ty}(c) = \sigma$ then $\mathrm{Ty}(A) = \sigma$.

Note that the definition requires each name in t either to come from $N_O \uplus N_P$ or to be introduced at a unique point in t. Thus, for each name d in t, we can specify its owner as $X \in \{O, P\}$ if $d \in N_X$ or d was introduced by X. A name owned by X is called an X-name. Since head names from outside $N_O \uplus N_P$ must be introduced in actions of opposite polarity, it follows that in a trace the players are calling each other's functions and also returning to each other's functions. As head names that are not in $N_O \uplus N_P$ must be introduced in earlier actions, it follows that Ty is unique. Thus, given a (N_O, N_P)-trace t and $d \in N_O \cup N_P \cup \nu(t)$, we can write $t \vdash d{:}\sigma$ if

$\mathrm{Ty}(d) = \sigma$. Finally, note that the definition requires names that are introduced to be fresh with respect to the preceding trace. This corresponds to testing a term with a functional parameter on a fresh name in the spirit of open bisimulation [8, 9].

Example 17. If N_O is $[h \mapsto (\mu h.\mathsf{Unit}^\perp \to (\mathsf{Unit}^\perp \to \mathsf{Unit}^\perp)^{\{h\}})^\emptyset, c \mapsto \mathsf{Unit}^\perp]$ then $\overline{h}((), c_1)\, c_1(f)\, \overline{h}((), c_2)\, c_2(g)\, \overline{f}((), c_3)\, c_3(())\, \overline{g}((), c_4)\, c_4(())\, \overline{c}(())$ is an $(N_O, \emptyset)$-trace with $\mathrm{Ty}(f) = \mathrm{Ty}(g) = (\mathsf{Unit}^\perp \to \mathsf{Unit}^\perp)^{\{h\}}$.

The traces as defined so far do not convey information about potential state sharing. To this end, we use another countably infinite set AStates of ***abstract states*** to decorate certain names in traces with finite subsets S of AStates. We will then say that the name is ***qualified*** by S and write f^S or c^S. Abusing notation, we will also write A^S on the understanding that, for $A \notin \mathrm{Names}$, A^S simply stands for (undecorated) A.

Abstract states can be viewed as locations created by the context, which can be shared between functions and continuations provided to the term, but not disclosed directly. For this reason, they are represented as a distinct resource for names and locations.

Definition 18. Let (N_O, N_P) be an initial typing, let $X \in \{O, P\}$ and let $\Upsilon : N_X \to \mathcal{P}(\mathrm{AStates})$. Υ is ***X-consistent*** with (N_O, N_P) provided, for all $f \in N_X$, $\Upsilon(f) \neq \emptyset$, and for any $s \in \bigcup \mathrm{img}(\Upsilon)$, there exists $g \in N_X$ such that, for all $f \in N_X$, we have $s \in \Upsilon(f)$ if and only if $g \in f^*_{N_O,N_P}$. Moreover, for all $c \in N_X$, $\Upsilon(c) = \Upsilon(N_X \cap \mathrm{FNames})$.

Intuitively, the allocation of abstract states by Υ matches the reachability relation implied by N_X, and initial continuation names get access to all states assigned to function names.

Definition 19. Suppose $X \in \{O, P\}$ and let Υ be X-consistent with (N_O, N_P). An (N_O, N_P)-trace t is called an (N_O, N_P, Υ)-***qualified*** ***X-trace*** if it starts with an X action, each $\overline{X}$-name in t is annotated (qualified) with a finite subset of AStates in such a way that each $d \in N_{\overline{X}}$ is qualified with $\Upsilon(d)$, and X-names remain unannotated. We will often omit (N_O, N_P, Υ) when it is clear from the context.

Example 20. If N_O is as in Example 17, and $\Upsilon(h) = \Upsilon(c) = \{l\}$, then $\mathtt{t}_1 = \overline{h^l}((), c_1)\ c_1(f^{l,m})\ \overline{h^l}((), c_3)\ c_3(g^{l,n})\ \overline{f^{l,m}}((), c_4)\ c_4(())\ \overline{g^{l,n}}((), c_5)\, c_5(())\ \overline{c^l}(())$ is an $(N_O, \emptyset, \Upsilon)$-qualified-P-trace, as is $\mathtt{t}_2 = \overline{h^l}((), c_1)\ c_1(f^m)\ \overline{h^l}((), c_3)\, c_3(g^n)\ \overline{f^m}((), c_4)\, c_4(())\ \overline{g^n}((), c_5)\ c_5(())\, \overline{c^l}(())$.

Traces interpreting terms will be qualified P-traces for some initial typing $(N_O, \emptyset)$ and O-consistent Υ, i.e. only O-names will be qualified. However, the more general definition will be useful when investigating contextual interactions of terms.

IV. LTS (DYNAMICS)

Building on the above notion of traces, we can define an LTS, called $\mathcal{L}_P$, which will generate the set of traces corresponding to a term, given some initial typing $(N_O, \emptyset)$. $\mathcal{L}_P$ will contain terms built from reachability type syntax, extended to allow function names to appear as values. We extend $\to$ to behave accordingly.

Configurations $\mathcal{L}_P$ is a stack-based transition system. Hence, its configurations will consist of a state paired up with a stack. There will be two kinds of states: $\langle \gamma, \phi, h, \mathrm{Ty}, \Psi, Fn, \Upsilon, Sn \rangle$ (*passive*, O to play) and $\langle M, c, \gamma, \phi, h, \mathrm{Ty}, \Psi, \Upsilon \rangle$ (*active*, internal or P to play). In both, ϕ contains all names introduced so far by both players, h is the current heap, and Ty is a map from names to types. Due to subtyping, $\mathrm{Ty}(d) = \tau^Q$ stands for an actual type $\tau_d^{Q_d}$ such that $\tau_d^{Q_d} <: \tau^{Q'}$ for some $Q \sqsubseteq Q'$.

γ is an *environment* mapping function names introduced by P to functions. In an active configuration, M is the *term* component, which captures the current behaviour of P, and c is the continuation O-name to produce the result to.

Ψ and Υ are introduced to account for reachability-related information about the potential behaviour of the environment (O), which needs to be explored exhaustively.

$\Psi : \mathrm{Names} \rightharpoonup \mathcal{P}(\mathrm{FNames})$ is a ***reachability map***, used to determine which function names are reachable from function O-names. In $\mathcal{L}_P$, this is extended to cover also continuation O-names, which will be mapped to the reachability set available at the time the name is introduced. In passive states, Fn represents the set of function names currently available to O, and will play a role analogous to the filter.

Υ is a similar mapping from O-names to finite sets of abstract states, and Sn represents the set of abstract states, which O could have access to. These are used so that the generated actions can explore all potential scenarios involving sharing, subject to satisfying constraints dictated by reachability types.

Finally, we use the stack to force the environment (O) to return to the most recent unreturned call when returning. The stack alphabet will consist of elements of the form $(c, (K, c'), (f, A))$, where c is a continuation P-name, K is an evaluation context in which to use the value produced to c, c' is an continuation O-name to return the result of K to, and (f, A) provide details of the call.

Transitions The transition rules of $\mathcal{L}_P$ are presented in Figure 2. They are labelled transitions between states, with $\mathbf{a}/m$ denoting pushing m to the stack when producing the label $\mathbf{a}$, and $\mathbf{a}, m$ denoting popping m when producing $\mathbf{a}$. We discuss the five cases in turn.

($\mathbf{P}\tau$) The $(P\tau)$ rule simply embeds the operational semantics of the language into $\mathcal{L}_P$, and makes it possible to reduce terms until a value V or a callback $K[fV]$ is reached, at which point the rules (PA) and (PQ) take over. They rely on an auxiliary function $\mathbf{AVal}$, which assigns the corresponding set of abstract values to a given value of Ref-free type: $\mathbf{AVal}(V) = \mathrm{FNames}$ (if V is of function type) and $\mathbf{AVal}(V) = \{V\}$ otherwise.

($\mathbf{PA}$) The (PA) rule handles values. If V is of function type then it will be included in γ' using a fresh name A (this is enforced by $\mathbf{AVal}(V)$, $\uplus$ and $[\nu(A) \mapsto V]$) and the type of A will be recorded in Ty'. The corresponding label $\overline{c^S}(A)$ announces the passing of A to c, where $S = \Upsilon(c)$ corresponds to the set of abstract states available at the point when c was introduced. Finally, the rule initialises the Fn, Sn components for use in the subsequent passive state. This is done again by referring to the point of their origin (via $\Psi(c), \Upsilon(c)$

$(P\tau)$ $\langle M, c, \gamma, \phi, h, \mathrm{Ty}, \Psi, \Upsilon\rangle \xrightarrow{\tau} \langle N, c, \gamma, \phi, h', \mathrm{Ty}, \Psi, \Upsilon\rangle$
when $(M, h) \to (N, h')$

(PA) $\langle V, c, \gamma, \phi, h, \mathrm{Ty}, \Psi, \Upsilon\rangle \xrightarrow{\overline{c^S}(A)} \langle \gamma \cdot \gamma', \phi \uplus \nu(A), h, \mathrm{Ty} \cdot \mathrm{Ty}', \Psi, \Psi(c) \cup \nu(A), \Upsilon, \Upsilon(c)\rangle$
when $S = \Upsilon(c),\ A \in \mathbf{AVal}(V),\ \gamma' = [\nu(A) \mapsto V],\ \mathrm{Ty}' = [\nu(A) \mapsto \mathrm{Ty}(c)]$

(PQ) $\langle K[f\,V], c', \gamma, \phi, h, \Psi, \mathrm{Ty}, \Upsilon\rangle \xrightarrow{\overline{f^S}(A,c)/(c,(K,c'),(f,A))} \langle \gamma \cdot \gamma', \phi \uplus \nu(A) \uplus \{c\}, h, \mathrm{Ty} \cdot \mathrm{Ty}', \Psi, \Psi(f) \cup \nu(A), \Upsilon, \Upsilon(f)\rangle$
when $S = \Upsilon(f),\ A \in \mathbf{AVal}(V),\ \gamma' = [\nu(A) \mapsto V],\ \mathrm{Ty}(f) = (\mu g.(x : \tau_1^{Q_1}) \to \tau_2^{Q_2})^Q,$
$\mathrm{Ty}' = [\nu(A) \mapsto \tau_1^{Q_1}, c \mapsto \tau_2^{Q_2\{f/g\}\{\nu(A)/x\}}]$

(OA) $\langle \gamma, \phi, h, \mathrm{Ty}, \Psi, Fn, \Upsilon, Sn\rangle \xrightarrow{c(A^S),(c,(K,c')),(f,A'))} \langle K[A], c', \gamma, \phi \uplus \nu(A), h, \mathrm{Ty} \cdot \mathrm{Ty}', \Psi \cdot \Psi', \Upsilon \cdot \Upsilon'\rangle$
when $\mathrm{Ty}(c) = \tau^Q,\ A \in \mathbf{AVal}_{ty}(\tau),\ \mathrm{Ty}' = [\nu(A) \mapsto \mathrm{Ty}(c)],\ \Psi' = [\nu(A) \mapsto F],\ \Upsilon' = [\nu(A) \mapsto S]$
- If $\nu(A) = \emptyset$ then F, S need not be specified.
- Otherwise $\mathrm{ArgQ}(\tau) \subseteq F \subseteq Fn$, $\mathcal{R}^{\Psi}_{\mathrm{Ty}}(F; Fn') \cap Fn' \subseteq Q\{\Psi(f)/f\}$, where $Fn' = \Psi(f) \cup \nu(A')$,
  $S \subseteq S'$, if $h \in \mathcal{R}^{\Psi}_{\mathrm{Ty}}(F; Fn') \cap (\mathrm{dom}(\Upsilon) \setminus Fn')$ then $\Upsilon(h) \subseteq S'$,
  where $S' = ((Sn \setminus \Upsilon(f)) \cup (\Upsilon(f) \cap \bigcup \Upsilon(Q \cap \mathrm{dom}(\Upsilon))) \uplus T$ and $T \neq \emptyset$ is disjoint from Υ.

(OQ) $\langle \gamma, \phi, h, \mathrm{Ty}, \Psi, Fn, \Upsilon, Sn\rangle \xrightarrow{f(A^S, c^{S'})} \langle V\,A, c, \gamma, \phi \uplus \nu(A) \uplus \{c\}, h, \mathrm{Ty} \cdot \mathrm{Ty}', \Psi \cdot \Psi', \Upsilon \cdot \Upsilon'\rangle$
when $f \in Fn,\ \gamma(f) = V,\ \mathrm{Ty}(f) = (\mu g.(x : \tau_1^{Q_1}) \to \tau_2^{Q_2})^Q,\ A \in \mathbf{AVal}_{ty}(\tau_1),\ Q_2 \sqsubseteq Fn \cup \{g, x\},$
$\mathrm{Ty}' = [\nu(A) \mapsto \tau_1^{Q_1}, c \mapsto \tau_2^{Q_2\{f/g\}\{\nu(A)/x\}}],\ \Psi' = [\nu(A) \mapsto F, c \mapsto Fn],\ \Upsilon' = [\nu(A) \mapsto S, c \mapsto S']$
- If $\nu(A) = \emptyset$ then $S' = Sn$ and F, S need not be specified.
- Otherwise $\mathrm{ArgQ}(\tau_1) \subseteq F \subseteq Fn,\ S' = Sn \uplus T,\ T \subseteq S \subseteq S'$, where $T \neq \emptyset$ is disjoint from Υ, and
  $(S \cup \bigcup_{g \in F} \mathcal{R}^{\Psi,\Upsilon}_{\mathrm{Ty}}(g)) \cap \mathcal{R}^{\Psi,\Upsilon}_{\mathrm{Ty}}(f) \subseteq \mathcal{R}^{\Psi,\Upsilon}_{\mathrm{Ty}}(Q_1)$.

Given $N \subseteq \mathrm{Names}$, $[N \mapsto \mathcal{V}]$ stands for the map $[n \mapsto \mathcal{V} \mid n \in N]$. For $A \notin \mathrm{Names}$, we take A^S to mean A.

Fig. 2: The transition system $\mathcal{L}_P$

respectively) and, if A is a name, it will also be included in Fn. Readers familiar with game semantics will notice that this is similar to how O-views are calculated. However, there will be a difference in calculations at O-actions, because not all names will be available due to reachability constraints.

(**PQ**) The (PQ) rule deals with callbacks $K[fV]$. As far as component updates are concerned, it is similar to the previous one. If V is of function type then it will be added to γ under a fresh name, and c will always be a fresh name. This time, the label is $\overline{f^S}(A, c)$, signifying a call to f, where $S = \Upsilon(f)$ is the set of abstract states available to f. The Fn, Sn components are prepared analogously using $\Psi(f)$, $\Upsilon(f)$ respectively. The main difference is that this action pushes $(c, (K, c'), (f, A))$ to the stack. This is done to enforce the stack discipline on environment returns as well as providing all the necessary information required to return to the call in (OA).

The (OQ) and (OA) rules, to be discussed next, are concerned with testing terms from γ (OQ) or continuations from the stack (OA) with arbitrary abstract values. In their definition, we use an auxiliary function $\mathbf{AVal}_{ty}$, which returns the set of abstract values corresponding to a given Ref-free type: $\mathbf{AVal}_{ty}(\mathsf{Unit}) = \{()\}$, $\mathbf{AVal}_{ty}(\mathsf{Bool}) = \{\mathbf{tt}, \mathbf{ff}\}$, $\mathbf{AVal}_{ty}(\mathsf{Int}) = \{\widehat{n} \mid n \in \mathbb{Z}\}$, and $\mathbf{AVal}_{ty}(\tau) = \mathrm{FNames}$ if τ is a function type.

(**OQ**) In the (OQ) rule, one of the currently available functions ($f \in Fn$) is retrieved from γ and applied to an abstract value A with continuation c, producing the $f(A^S, c^{S'})$ label representing a call. Note that fresh names will be used as A if the argument is of function type, which is enforced using $\uplus$. The rule can be seen through the lens of typing an application $f\,A$. In this view, Fn and Sn play a role similar to the *filter* (for function names/variables and abstract states/locations, respectively). Following the variable rule, we must check that f is with the filter ($f \in Fn$). From the application rule, we also have to check that the qualifier of the codomain of the function is within the filter, or the self-reference or argument ($Q_2 \sqsubseteq Fn \cup \{f, x\}$). Next, we need to consider the type given to A. As $\mathrm{Ty}(f) = (\mu g.(x : \tau_1^{Q_1}) \to \tau_2^{Q_2})^Q$, A must have type $\tau_A^{Q_A}$ with $\tau_A^{Q_A} <: \tau_1^{Q'}$ for some Q'. If τ_1 is a function type, i.e. $\nu(A) \neq \emptyset$, we can consider Q_A as consisting of two parts, function names (F) and abstract states (S). The qualifier Q_A must then fall within the filter: $F \subseteq Fn$ and $S \subseteq S'$, where $S' = Sn \uplus T$ and $T \subseteq S$. The role of T here is to model *fresh* abstract states, not captured by anything previously disclosed, corresponding to the growth of the filter during reduction.

Before we can discuss the next condition, we need to introduce $\mathrm{ArgQ}(\tau)$ as the combined qualifier of all the arguments in τ: $\mathrm{ArgQ}(\beta) = \mathrm{ArgQ}(\mathsf{Ref}) = \emptyset$ and $\mathrm{ArgQ}(\mu f.(x : \tau_1^{Q_1}) \to \tau_2^{Q_2}) = Q_1 \sqcup \mathrm{ArgQ}(\tau_2)$. One can show that $wb(\tau^Q)$ implies $\mathrm{ArgQ}(\tau) \subseteq \widehat{Q}$. Since $\mathrm{ArgQ}(\tau_1) \sqsubseteq \mathrm{ArgQ}(\tau_A)$ (by $\tau_A^{Q_A} <: \tau_1^{Q'}$) and $\mathrm{ArgQ}(\tau_A) \sqsubseteq Q_A$ (by $wb(\tau_A^{Q_A})$), it follows that $\mathrm{ArgQ}(\tau_1) \sqsubseteq Q_A$, which is reflected by $\mathrm{ArgQ}(\tau_1) \subseteq F$.

Most importantly, applications require that the overlap between function and argument is only that permitted by the qualifier on the function domain ($(S \cup \bigcup_{g \in F} \mathcal{R}^{\Psi,\Upsilon}_{\mathrm{Ty}}(g)) \cap \mathcal{R}^{\Psi,\Upsilon}_{\mathrm{Ty}}(f) \subseteq \mathcal{R}^{\Psi,\Upsilon}_{\mathrm{Ty}}(Q_1)$), where $\mathcal{R}^{\Psi,\Upsilon}_{\mathrm{Ty}}$ assigns the set of function names and abstract states reachable from a given function name by following Ψ, Υ (for O-names) or the qualifier from

Ty (for P-names) according to the definition below. Observe the asymmetry between O and P compared to the syntactic notion of reachability, which arises as game semantics captures the difference in knowledge between term and environment.

Definition 21. Let $\Psi : \mathrm{Names} \rightharpoonup \mathrm{Names}$ and $\mathrm{Ty} : \mathrm{Names} \rightharpoonup \mathrm{Types}$ be such that $\mathrm{dom}(\Psi) \cup \mathrm{img}(\Psi) \cup \nu(\mathrm{img}(\mathrm{Ty})) \subseteq \mathrm{dom}(\mathrm{Ty})$. Define a directed graph $\mathcal{G}^{\Psi}_{\mathrm{Ty}}$ by taking the set of vertices to be $\mathrm{dom}(\mathrm{Ty})$ and defining edges (v_1, v_2) if ($v_1 \in \mathrm{dom}(\Psi)$ and $v_2 \in \Psi(v_1)$) or ($v_1 \notin \mathrm{dom}(\Psi)$, $\mathrm{Ty}(v_1) = \tau^Q$ and $v_2 \in Q$). We shall write $\mathcal{R}^{\Psi}_{\mathrm{Ty}}(v)$ for the set of vertices of $\mathcal{G}^{\Psi}_{\mathrm{Ty}}$ reachable from v, and $\mathcal{R}^{\Psi}_{\mathrm{Ty}}(U)$ for $\bigcup_{v \in U} \mathcal{R}^{\Psi}_{\mathrm{Ty}}(v)$. Furthermore, given $\Upsilon : \mathrm{Names} \rightharpoonup \mathrm{AStates}$, let $\mathcal{R}^{\Psi,\Upsilon}_{\mathrm{Ty}}(v) \triangleq \mathcal{R}^{\Psi}_{\mathrm{Ty}}(v) \cup \bigcup \Upsilon(\mathcal{R}^{\Psi}_{\mathrm{Ty}}(v) \cap \mathrm{dom}(\Upsilon))$.

If we imagine the application $f\,A$ occurring in some evaluation context K, the filter for typing K is the original filter along with any of the fresh locations. Hence, $\Psi(c) = Fn$ and $\Upsilon(c) = S'$. The type map Ty is updated to give A the type $\tau_1^{Q_1}$ (if it is a function type), which corresponds to the fact that P will view A as having type $\tau_1^{Q_1}$, whereas O has some freedom in its choice of type.

(**OA**) The (OA) rule can be viewed as returning a result A (with $\mathrm{Ty}(A) = \tau^Q$) of an earlier call to an imaginary function f constructed by O, which was called on A'. To this end, we use the top continuation name c from the stack and pass an abstract value A to the corresponding evaluation context (K, c'), producing the $c(A^S)$ label. This leads to an active state with term $K[A]$ and continuation c'. Note that f, A' are also available from the stack. The remaining conditions consist of updating the components of the configuration to account for new names in $\nu(A)$ (if any), so below we assume $\nu(A) \neq \emptyset$.

We need to allow A to have types $\tau_A^{Q_A} <: \tau^{Q'}$ for some Q', where $Q_A = (F, S)$ represents the resources made available to A for subsequent use in applications. Thus, we need to stipulate $F \subseteq Fn$, where Fn represents the currently available function names. Because $wb(\tau_A^{Q_A})$ and $\tau_A^{Q_A} <: \tau^{Q'}$, we also require $\mathrm{ArgQ}(\tau) \subseteq F$.

We can classify the names in Fn as follows: those introduced before the function call to f (i.e. $\Psi(f)$, corresponding self-reference f in Q), the argument A' (if $\nu(A') \neq \emptyset$), and those introduced 'inside' the function f (i.e. $Fn \setminus Fn'$, where $Fn' = \Psi(f) \cup \nu(A')$). From F we should be allowed to reach only those variables from Fn' that are permitted by the qualifier $Q\{\Psi(f)/f\}$. This is expressed by $\mathcal{R}^{\Psi}_{\mathrm{Ty}}(F; Fn') \cap Fn' \subseteq Q\{\Psi(f)/f\}$, where $\mathcal{R}^{\Psi}_{\mathrm{Ty}}(U_1; U_2)$ is defined below.

Definition 22. Given Ψ, Ty as in Definition 21 and $U_1, U_2 \subseteq \mathrm{dom}(\mathrm{Ty})$, let $\mathcal{R}^{\Psi}_{\mathrm{Ty}}(U_1; U_2)$ consist of all $u \in \mathcal{R}^{\Psi}_{\mathrm{Ty}}(U_1)$ such that all vertices on the witnessing paths other than u must come from outside U_2.

Similarly, S may only consist of 'new' abstract states introduced since the corresponding question $(Sn \setminus \Upsilon(f))$, 'old' abstract states from $\Upsilon(f)$ that are permitted by Q (i.e. belong to $\bigcup \Upsilon(Q \cap \mathrm{dom}(\Upsilon))$) and, some fresh states T. This is captured by $S \subseteq S'$. Finally, we also need to ensure that the names in F do not lead to abstract states disallowed by S': if $h \in \mathcal{R}^{\Psi}_{\mathrm{Ty}}(F; Fn') \cap (\mathrm{dom}(\Upsilon) \setminus Fn')$ then $\Upsilon(h) \subseteq S'$. We use $(\mathrm{dom}(\Upsilon) \setminus Fn')$ above, because $\mathcal{R}^{\Psi}_{\mathrm{Ty}}(F; Fn') \cap Fn'$ has already been validated in the preceding test.

Remark 23. We note here how abstract states are involved in the generation of traces. Abstract state annotations are used to represent various patterns of sharing which can occur in a trace. These patterns are not known in advance, and the LTS generates (arbitrarily large) sets of abstract states to predict them. The choices made while generating a trace can rule out later actions occurring in the trace, due to the conditions enforced in OQ and OA actions. For example, for $c : (x : (\mathsf{Unit}^{\perp} \to \mathsf{Unit}^{\perp})^{\emptyset}) \to ((\mathsf{Unit}^{\perp} \to \mathsf{Unit}^{\perp})^{\emptyset} \to \mathsf{Unit}^{\perp})^{\{x\}}$, after generating the trace $\overline{c^{\emptyset}}(f)\ f(g_1^{l_1}, c_1^{l_1})\ \overline{c_1^{l_1}}(h_1)\ f(g_2^{l_1,l_2}, c_2^{l_1,l_2})\ \overline{c_2^{l_1,l_2}}(h_2)\ h_1(e^{l_3}, c_3^{l_1,l_2,l_3})\ \overline{e^{l_3}}((), c_4)$, the LTS cannot produce a question on h_2. For this to occur, we would have $h_2 \in \Psi(e)$, so when generating $h_1(e^{\{l_3\}}, c_3^{\{l_1,l_2,l_3\}})$, we would pick $h_2 \in F$. But this would violate the condition $(\{l_3\} \cup \bigcup_{g \in F} \mathcal{R}^{\Psi,\Upsilon}_{\mathrm{Ty}}(g)) \cap \mathcal{R}^{\Psi,\Upsilon}_{\mathrm{Ty}}(h_1) \subseteq \mathcal{R}^{\Psi,\Upsilon}_{\mathrm{Ty}}(\emptyset)$ of the (OQ) rule, as $l_1 \in \mathcal{R}^{\Psi,\Upsilon}_{\mathrm{Ty}}(h_2) \cap \mathcal{R}^{\Psi,\Upsilon}_{\mathrm{Ty}}(h_1)$ because $h_2 \to g_2$ and $h_1 \to g_1$ in $\mathcal{G}^{\Psi}_{\mathrm{Ty}}$. Violating this condition corresponds to allowing the function h_1 to overlap with its argument e.

Initialisation Recall that $\mathcal{L}_P$ involves a stack, i.e. its configurations consist of a state and a stack.

Definition 24. Given configurations $\mathbf{C}, \mathbf{C}'$ of $\mathcal{L}_P$, we write $\mathbf{C} \overset{\mathtt{a}}{\Rightarrow} \mathbf{C}'$ if $\mathbf{C} \overset{\tau}{\to}{}^* \mathbf{C}'' \overset{\mathtt{a}}{\to} \mathbf{C}'$, with $\overset{\tau}{\to}{}^*$ representing multiple (possibly none) τ-actions. We extend the notation to sequences of labels: given $\mathtt{t} = \mathtt{a}_1 \ldots \mathtt{a}_n$, we write $\mathbf{C} \overset{\mathtt{t}}{\Rightarrow} \mathbf{C}'$, if there exist $\mathbf{C}_1, \ldots, \mathbf{C}_{n-1}$ such that $\mathbf{C} \overset{\mathtt{a}_1}{\Rightarrow} \mathbf{C}_1 \cdots \mathbf{C}_{n-1} \overset{\mathtt{a}_n}{\Rightarrow} \mathbf{C}'$. We define $\mathbf{Tr}(\mathbf{C}) = \{\mathtt{t} \mid \text{there exists } \mathbf{C}' \text{ such that } \mathbf{C} \overset{\mathtt{t}}{\Rightarrow} \mathbf{C}'\}$.

Remark 25. Due to the freedom of name choice, note that $\mathbf{Tr}(\mathbf{C})$ is closed under renamings (of Names and AStates) that preserve the elements from $\mathbf{C}$.

In order to use $\mathcal{L}_P$ to generate qualified P-traces from terms, we need to specify initial configurations. Suppose $\Gamma = \{x_1 : \tau_1^{Q_1}, \cdots, x_k : \tau_k^{Q_k}\}$ and $\Gamma \vdash M : \sigma$ is r-free. A Γ-***assignment*** ρ is a map from $\{x_1, \cdots, x_k\}$ to the set of values and function names, such that, for all $1 \leq i \neq j \leq k$, if τ_i is not a function type, $\rho(x_i) : \tau_i$, and if τ_i is a function type, $\rho(x_i) \in \mathrm{FNames}$ and $\nu(\rho(x_i)) \cap \nu(\rho(x_j)) = \emptyset$. ρ simply creates values (for base-type variables) and function names (for function-type variables) corresponding to the context.

For a fixed ρ and a continuation name c, let us define $N_{\rho,c} = \mathrm{Ty} \triangleq [\nu(\rho(x_i)) \mapsto \rho(\tau_i^{Q_i}) \mid 1 \leq i \leq k] \cdot [c \mapsto \rho(\sigma)]$ and $\Psi \triangleq [\nu(\rho(x_i)) \mapsto \rho(Q_i) \mid 1 \leq i \leq k] \cdot [c \mapsto \mathrm{dom}(\mathrm{Ty}) \cap \mathrm{FNames}])$. Let Υ be O-consistent with $(N_{\rho,c}, \emptyset)$. We shall call (ρ, c, Υ) a ***qualified*** (Γ, c)-***assignment***. Then the initial configuration $\mathsf{C}^{\rho,c,\Upsilon}_{\Gamma \vdash M:\sigma}$ for $\mathcal{L}_P$, is defined to be $(\ \langle\ M\{\rho\},\ c,\ \emptyset,\ \mathrm{dom}(\mathrm{Ty}),\ \emptyset,\ \mathrm{Ty},\ \Psi,\ \Upsilon\ \rangle,\ \epsilon)$, where ϵ represents the empty stack.

Example 26. Consider $\vdash M_1 \triangleq \lambda g.g\,() : \sigma$, where $\sigma = ((\mathsf{Unit}^{\perp} \to \mathsf{Unit}^{\perp})^{\emptyset} \to \mathsf{Unit}^{\perp})^{\emptyset}$. The only $(\emptyset, c)$-assignment

is $(\emptyset, c, \Upsilon_0)$ with $\Upsilon_0 = [c \mapsto \emptyset]$. We can now derive a trace from $\mathsf{C}^{\emptyset,c,\Upsilon_0}_{\vdash M_1:\sigma}$. Taking $\Psi_0 = [c_0 \mapsto \emptyset]$,

$$\begin{aligned}&\mathsf{C}^{\emptyset,c,\Upsilon_0}_{\vdash M_1:\sigma} = (\langle M_1, c, \emptyset, \{c\}, \emptyset, [c \mapsto \sigma], \Psi_0, \Upsilon_0\rangle, \epsilon)\\ &\xrightarrow{\overline{c^{\emptyset}}(f)} \quad (\langle \gamma, \{c, f\}, \emptyset, \mathrm{Ty}_1, \Psi_0, \{f\}, \Upsilon_0, \emptyset\rangle, \epsilon)\\ &\text{where } \gamma = [f \mapsto M_1], \mathrm{Ty}_1 = [c \mapsto \sigma, f \mapsto \sigma].\end{aligned}$$

To continue, O must call f.

$$\begin{aligned}&\xrightarrow{f(g_1^{\{l_1\}}, c_1^{\{l_1\}})} (\langle M_1\, g_1, c_1, \gamma, \{c, f, g_1, c_1\}, \emptyset, \mathrm{Ty}_2, \Psi, \Upsilon\rangle, \epsilon)\\ &\text{where } \mathrm{Ty}_2 = \mathrm{Ty}_1 \cdot [g_1 \mapsto (\mathsf{Unit}^{\perp} \to \mathsf{Unit}^{\perp})^{\emptyset}, c_1 \mapsto \mathsf{Unit}^{\perp}],\\ &\Psi = \Psi_0 \cdot [g_1 \mapsto \emptyset, c_1 \mapsto \{f\}],\ \Upsilon = \Upsilon_0[g_1, c_1 \mapsto l_1]\end{aligned}$$

Note that this is the only possible choice of Ψ because f cannot be included for g_1 as it violate the overlap condition.

$$\begin{aligned}&\xrightarrow{\tau}{}^{*} \quad (\langle g_1\,(), c_1, \gamma, \{c, f, g_1, c_1\}, \emptyset, \Psi, \mathrm{Ty}_2, \Upsilon\rangle, \epsilon)\\ &\xrightarrow{\overline{g_1^{\{l_1\}}}((), d_1)} (\langle \gamma, \phi, \emptyset, \mathrm{Ty}_3, \Psi, \emptyset, \Upsilon, \{l_1\}\rangle, (d_1, (\bullet, c_1), g_1, ()))\\ &\phi = \{c, f, g_1, c_1, d_1\}, \mathrm{Ty}_3 = \mathrm{Ty}_2 \cdot [d_1 \mapsto \mathsf{Unit}^{\perp}]\end{aligned}$$

Fn is empty, so all O can do is answer the pending question.

$$\begin{aligned}&\xrightarrow{d_1(())} \quad (\langle (), c_1, \gamma, \phi, \emptyset, \mathrm{Ty}_3, \Psi, \Upsilon\rangle, \epsilon)\\ &\xrightarrow{\overline{c_1^{\{l_1\}}}(())} (\langle \gamma, \phi, \emptyset, \mathrm{Ty}_3, \Psi, \{f\}, \Upsilon, \{l_1\}\rangle, \epsilon)\end{aligned}$$

If O wishes to extend this trace, it can only do so by essentially repeating the last four actions with fresh names.

When defining the semantics of our terms, ultimately we will be interested only in qualified traces that correspond to terminating interactions. Recall that $\mathcal{L}_P$ uses a stack to ensure that O (the environment) only ever answers the last P-question. Because the language does not feature any control operators, answers from P (the term) satisfy an analogous property for free (we spell this out formally in the next section). Consequently, the traces that are relevant to contextual testing in this setting are those in which all questions have been answered. Note that this implies that the trace must end with an action by P. This gives rise to the notion of a complete qualified P-trace defined below.

Definition 27. A qualified P-trace from $\mathbf{Tr}(\mathsf{C}^{\rho,c,\Upsilon}_{\Gamma\vdash M:\sigma})$ is ***complete*** if it has odd length and $\mathcal{L}_P$ has empty stack after generating it.

Definition 28. The ***trace semantics*** of a r-free judgement $\Gamma \vdash M : \tau^Q$ is defined to be

$$\mathbf{Tr}(\Gamma \vdash M : \tau^Q) \triangleq \{\, ((\rho, c, \Upsilon), t) \mid c \in \mathrm{CNames}, (\rho, c, \Upsilon) \text{ is a qualified } \Gamma\text{-assignment}, \ t \in \mathbf{Tr}(\mathsf{C}^{\rho,c,\Upsilon}_{\Gamma\vdash M:\tau^Q}),\ t \text{ is complete} \,\}$$

One of our key results will be the soundness of this model (Theorem 48), which will imply that $\mathbf{Tr}(\Gamma \vdash M_1 : \tau^Q) = \mathbf{Tr}(\Gamma \vdash M_2 : \tau^Q)$ entails program equivalence.

Example 29. Returning to Example 26, the complete traces of $\mathbf{Tr}(\mathsf{C}^{\emptyset,c,\Upsilon_0}_{\vdash M_1:\sigma})$ all have the form $\overline{c^{\emptyset}}(f)\ f(g_1^{S_1}, c_1^{S'_1})\ \overline{g_1^{S_1}}((), d_1)\ d_1(())\ \overline{c_1^{S'_1}}(())\ \ldots\ f(g_n^{S_n}, c_1^{S'_n})\ \overline{g_n^{S_n}}((), d_n)\, d_n(())\ \overline{c_1^{S'_n}}(())$, where $n \geq 0$, $T_i \subseteq S_i \subseteq S'_i = \bigcup_{1\leq k\leq i} T_k$. If we consider $M_2 \triangleq \mathbf{let}\, x = \mathbf{ref}\, \widehat{0}\, \mathbf{in}\, \lambda g.\mathbf{if}\, (!x = \widehat{0})\, \mathbf{then}\, (x := \widehat{1};\, g\,();\, x := \widehat{0})\, \mathbf{else}\, \Omega$ then, by a similar process, the traces in $\mathbf{Tr}(\mathsf{C}^{\emptyset,c,\Upsilon_0}_{\vdash M_2:\sigma})$ have the same form. Thus, $\mathbf{Tr}(\mathsf{C}^{\emptyset,c,\Upsilon_0}_{\vdash M_1:\sigma}) = \mathbf{Tr}(\mathsf{C}^{\emptyset,c,\Upsilon_0}_{\vdash M_2:\sigma})$. Thus, $\mathbf{Tr}(\vdash M_1 : \sigma) = \mathbf{Tr}(\vdash M_2 : \sigma)$ and so $\vdash M_1 \simeq_{ctx} M_2 : \sigma$. This is somewhat surprising, as in the standard setting we have that $\vdash M_1 \not\cong M_2$. However, the separating context $\mathbf{let}\, f = \bullet\, \mathbf{in}\, f\,(\lambda x.f\,(\lambda y.()))$ cannot be typed in λ^*_{wb}, due to the overlap between f and $\lambda x.f\,(\lambda y.())$.

The next sections describe technical properties of traces generated by $\mathcal{L}_P$ and how soundness is proved. In Section VII we refine the model to a fully abstract one.

V. Properties

Here we discuss the technical properties of qualified traces, starting with those that concern the underlying traces. We begin with bracketing, which is essentially the same as in standard game semantics [15]: in an X-bracketed trace, X can only answer the 'top' $\overline{X}$-question. By an ***X-prefix*** we shall mean a prefix ending in an X-action.

Definition 30 (Bracketing)**.** A (N_O, N_P)-trace t is ***X-bracketed*** if for any X-prefix $t'\ \check{c}(A)$, we have $c \in \mathsf{Top}_X(t')$, where Top_X is defined below, and c is not used as a head name in t'. t is ***bracketed*** if it is both O- and P-bracketed.

$$\begin{aligned}\mathsf{Top}_X(\epsilon) &= N_{\bar{X}} \cap \mathrm{CNames}\\ \mathsf{Top}_X(t\ \check{c}(A)) &= N_{\bar{X}} \cap \mathrm{CNames} \quad c \in N_X\\ \mathsf{Top}_X(t\ \check{f}(A', c)\ t'\ \check{c}(A)) &= \mathsf{Top}_X(t)\\ \mathsf{Top}_X(t\ \check{f}(A, c)) &= \{c\}\end{aligned}$$

The following notion captures situations in which all O-questions have been answered.

Definition 31 (Complete traces)**.** An O-bracketed $(N_O, \emptyset)$-P-trace t is ***complete*** if it is of odd length (i.e. ends with a P-action) and $\mathsf{Top}_O(t) = \emptyset$ (note this subsumes Definition 27).

To describe the other trace properties, it is useful to have some notation relating a continuation name to the question it is answering. We extend FNames with a fictional name $f_!$ for the initial continuations to answer, letting $\mathrm{FNames}_! = \mathrm{FNames} \uplus \{f_!\}$ and $\mathrm{Names}_! = \mathrm{Names} \uplus \{f_!\}$.

Definition 32. For an (N_O, N_P)-trace t, let us write $t \vdash c{:}\tau^Q[f, Y]$ to mean $t \vdash c{:}\tau^Q$ and c was introduced in t in $\check{f}(A, c)$ with $Y = \nu(A)$. We write $t \vdash c{:}\tau^Q[f_!, \emptyset]$ for $N_X(c) = \tau^Q$, i.e. when c was not introduced in t.

Next we introduce terminology that will help us describe restrictions on the use of names in traces.

Definition 33. A ***reachability map*** $\Psi : \mathrm{FNames}_! \rightharpoonup \mathcal{P}(\mathrm{FNames})$ is a partial map from $\mathrm{FNames}_!$ to $\mathcal{P}(\mathrm{FNames})$. Given an (N_O, N_P)-trace t, we define the set $\mathrm{Vis}^{\Psi}_X(t)$ of ***names visible to*** X in t as follows.

$$\begin{aligned}\mathrm{Vis}^{\Psi}_X(\epsilon) &\triangleq (N_X \cup N_{\bar{X}}) \cap \mathrm{FNames}\\ \mathrm{Vis}^{\Psi}_X(t\ \check{c}(A)) &\triangleq \nu(A) \cup (N_X \cap \mathrm{FNames}) \quad c \in N_X\\ \mathrm{Vis}^{\Psi}_X(t\ \check{f}(A, c)) &\triangleq \nu(A) \cup \Psi(f)\\ \mathrm{Vis}^{\Psi}_X(t\ \check{f}(A', c)\ t'\ \check{c}(A)) &\triangleq \nu(A) \cup \mathrm{Vis}^{\Psi}_X(t)\end{aligned}$$

Let Ty be the typing corresponding to t. We write $\mathcal{R}^{\Psi}_t(v)$ for $\mathcal{R}^{\Psi}_{\mathrm{Ty}}(v)$ (i.e reachable names), and $\mathcal{R}^{\Psi}_t(f;U)$ for $\mathcal{R}^{\Psi}_{\mathrm{Ty}}(f;U)$ (i.e. names reachable without passing through U).

$\mathrm{Vis}^{\Psi}_X(t)$ will be used in subsequent definitions to restrict the range of names available to players to the visible ones, akin to the classic visibility condition [15]. Using $\mathcal{R}^{\Psi}_t(v)$ and $\mathcal{R}^{\Psi}_t(f;U)$, we will be able to formulate additional restrictions corresponding to qualifiers in reachability types.

Definition 34 (ΨX-Visibility)**.** Given a reachability map Ψ, an (N_O, N_P)-trace t is said to be ΨX***-visible*** if

- $\mathrm{dom}(\Psi)$ is $\{f_!\}$ and X-names in $(N_X \cup \nu(t)) \cap \mathrm{FNames}$;
- $\Psi(f_!) = (N_O \cup N_P) \cap \mathrm{FNames}$ and, for all $f \in \mathrm{dom}(\Psi) \cap N_X$, if $N_X(f) = \tau^Q$ then $\Psi(f) = Q$;
- for any X-prefix $t'\, \check{f}(A, c)$ of t with $t' \vdash f{:}(\mu f.(x : \tau_1^{Q_1}) \to \tau_2^{Q_2})^Q$, we have $f \in \mathrm{Vis}^{\Psi}_X(t')$, $Q_2 \sqsubseteq \mathrm{Vis}^{\Psi}_X(t') \cup \{f, x\}$, and if $A = g$, $\mathrm{ArgQ}(\tau_1) \subseteq \Psi(g) \subseteq \mathrm{Vis}^{\Psi}_X(t')$ and $\mathcal{R}^{\Psi}_{t'}(g) \cap \mathcal{R}^{\Psi}_{t'}(f) \subseteq \mathcal{R}^{\Psi}_{t'}(Q_1)$;
- for any X-prefix $t'\, \check{c}(h)$ of t with $t' \vdash c{:}\tau^Q[f, Y]$, we have $\mathrm{ArgQ}(\tau) \subseteq \Psi(h) \subseteq \mathrm{Vis}^{\Psi}_X(t')$, and $\mathcal{R}^{\Psi}_t(\Psi(h); Fn') \cap Fn' \subseteq Q\{\Psi(f)/f\}$ where $Fn' = \Psi(f) \cup Y$.

Definition 35. An (N_O, N_P)-trace t is said to be ***X-visible*** if there exists a reachability map Ψ such that t is ΨX-visible.

The next kind of conditions concern annotations with abstract states. Recall that in qualified traces we only required that initial names be annotated in a way dictated by Υ, leaving a lot of flexibility for other annotations. The conditions below describe how other names are annotated. To express them, we introduce auxiliary notation.

Definition 36. Let t be an (N_O, N_P, Υ)-qualified X-trace.

- If t ends in an X-action $\mathbf{a}$ then we write $\mathrm{Last}(t)$ to refer to the set of abstract names used in $\mathbf{a}$, i.e. $\mathrm{Last}(t'\, \check{f}^S(A, c)) \triangleq S$ and $\mathrm{Last}(t'\, \check{c}^S(A)) \triangleq S$.
- We define $\Upsilon^t : \mathrm{FNames}_! \rightharpoonup \mathcal{P}(\mathrm{AStates})$ by $\Upsilon^t(d) = S$ if d^S appears in t (or $d \in N_{\bar{X}}$ and $S = \Upsilon(d)$), and $\Upsilon^t(f_!) = \bigcup \mathrm{img}(\Upsilon)$. We also write $t \vdash d^S$ if $\Upsilon^t(d) = S$.
- Intuitively, $\mathrm{OK}_t(f, Q)$ is shorthand for the set of abstract states that can be returned after t as a result of f under qualifier Q. We set $\mathrm{OK}_t(f, Q) \triangleq (\mathrm{Last}(t) \backslash \Upsilon^t(f)) \cup (\Upsilon^t(f) \cap \bigcup \Upsilon^t(Q \cap \mathrm{dom}(\Upsilon^t))$. The first component represents states created after the call to f, while the second one 'filters' $\Upsilon^t(f)$ (states existing at the time of the call) through Q.

Below we specify how abstract state can evolve.

Definition 37 (Qualified X-Trace)**.** An (N_O, N_P, Υ)-qualified X-trace t is ***well-qualified*** if it satisfies the following.

- If $t'\, \check{f}(A, c^{S'})$ is an $\overline{X}$-prefix of t with $\nu(A) = \emptyset$ then $S' = \mathrm{Last}(t')$.
- If $t'\, \check{f}(g^S, c^{S'})$ is an $\overline{X}$-prefix of t then $\emptyset \neq T \subseteq S \subseteq S'$, and $S' = \mathrm{Last}(t') \uplus T$ for T disjoint from $\Upsilon^{t'}$.
- If $t'\, \check{c}(g^S)$ is an $\overline{X}$-prefix of t, then $\emptyset \neq T \subseteq S \subseteq \mathrm{OK}_{t'}(f, Q) \uplus T$ for T disjoint from $\Upsilon^{t'}$ and $t' \vdash c{:}\tau^Q[f, Y]$.

Example 38. Recall Example 20. Both $\mathtt{t}_1$ and $\mathtt{t}_2$ are $(N_O, \emptyset, \Upsilon)$-well-qualified P-traces. For $N'_O = [h \mapsto (\mu h.\mathsf{Unit}^{\perp} \to (\mathsf{Unit}^{\perp} \to \mathsf{Unit}^{\perp})^{\emptyset})^{\emptyset}, c : \mathsf{Unit}^{\perp}]$, $\mathtt{t}_2$ is a $(N'_O, \emptyset, \Upsilon)$-well-qualified P-trace, but $\mathtt{t}_1$ is not.

Finally, we formulate a condition that ensures the annotations with abstract state are compatible with reachability maps. Essentially, this condition requires that in a question the overlap in abstract state between the function and the argument is restricted according to the qualifier. In an answer, abstract state on names introduced after the call that are reachable from the returned value must be permitted by the qualifier.

Definition 39 (Ψ-compatibility)**.** Let t be an (N_O, N_P, Υ)-qualified X-trace that is well qualified, and let Ψ be a reachability map whose domain consists of $f_!$ and all $\overline{X}$-function-names from t. We shall call t is Ψ***-compatible*** provided the following conditions are satisfied.

- If $t \vdash f{:}(\mu f.(x : \tau_1^{Q_1}) \to \tau_2^{Q_2})^Q$ then, for any $\overline{X}$-prefix $t'\, \check{f}(g^S, c^{S'})$ of t, $\mathcal{R}^{\Psi,\Upsilon^t}_t(g) \cap \mathcal{R}^{\Psi,\Upsilon^t}_t(f) \subseteq \mathcal{R}^{\Psi,\Upsilon^t}_t(Q)$.
- For any $\overline{X}$-prefix $t'\, \check{c}(g^S)$ of t, if $h \in \mathcal{R}^{\Psi}_t(\Psi(g); Fn) \cap (\mathrm{dom}(\Upsilon^t) \setminus Fn)$ implies $\Upsilon^t(h) \subseteq \mathrm{OK}_{t'}(f, Q)$, where $t' \vdash c{:}\tau^Q[f, Y]$ and $Fn = \Psi(f) \cup Y$.

Now we can state how these properties relate to $\mathcal{L}_P$.

Lemma 40. *Let $\Gamma \vdash M : \sigma$ be r-free, (ρ, c, Υ) a qualified Γ-assignment and $N_{\rho,c}$ as in the definition of $\mathsf{C}^{\rho,c,\Upsilon}_{\Gamma \vdash M:\sigma}$ If $\mathsf{C}^{\rho,c,\Upsilon}_{\Gamma \vdash M:\sigma} \overset{t}{\Rightarrow} \mathbf{C}$ then t is a $(N_{\rho,c}, \emptyset, \Upsilon)$-well-qualified P-trace whose qualifiers match the qualifying map $\Upsilon_{\mathbf{C}}$ in $\mathbf{C}$ and*

1) t is bracketed;
2) t is ΨO-visible and Ψ-compatible, where Ψ is the reachability map of $\mathbf{C}$ restricted to FNames and extended to $f_!$ by $\Psi(f_!) = (N_O \cup N_P) \cap \mathrm{FNames}$;
3) t is $\Psi' P$-visible for some Ψ'.

That we obtain an $(N_{\rho,c}, \emptyset, \Upsilon)$-well-qualified P-trace follows the setup of our LTS. O-bracketing is enforced by our use of the stack, while P-bracketing is a known condition satisfied by terms that do not use control operators [5, 15, 16]. The properties in 2) are *enforced* by the LTS to simulate the permitted behaviours of contexts, e.g. O-visibility is achieved thanks to the $f \in Fn$ condition in (OQ) as well as maintaining the Fn component so that it corresponds to the set of visible names. The property 3) is the most interesting, as it arises out of the behaviour of the term being used to generate the trace. The proof amounts to showing how to construct a suitable choice of Ψ_P. This is done by typing the various terms and evaluation contexts found in any configuration on the path, depending crucially on the type preservation result (Lemma 3). From the qualifier of the value V used to define $\gamma(f)$ in (PQ) or (PA), we extract a suitable choice for $\Psi_P(f)$.

Remark 41. Now that we have identified the properties of qualified traces, we can observe that abstract state allocations are in some way independent from the choice of underlying actions in a trace. In particular, for a ΨO-visible trace t, we can compute all of the allocations of abstract states so the

qualified trace is Ψ-compatible. We can reconcile this with Remark 23, as that pertained to the *generation* of traces action by action. We could have defined an LTS which generated traces without abstract states, and then defined the semantics of a term by taking all the ways of adding abstract states to the traces so they are Ψ-compatible. Such a presentation would be less direct, and complicate the proofs in the next section.

VI. Soundness

We now show soundness of the model, which will involve reasoning about context behaviour. First we reduce testing with arbitrary contexts to testing with evaluation contexts and closing substitutions, i.e. Closed Instances of Use (CIU) [17]. Given $\Gamma = x_1 \colon \tau_1^{Q_1}, \cdots, x_m \colon \tau_m^{Q_m}$, we write $\Sigma; \Gamma \vdash \gamma$ for substitutions γ such that $\Sigma;\ x_1 \colon \tau_1^{Q_1}, \cdots, x_{i-1} \colon \tau_{i-1}^{Q_{i-1}} \vdash \gamma(x_i) : \tau_i^{Q_i}$ and $\mathrm{FV}(\gamma(x_i)) = \emptyset$.

Definition 42 (CIU Approximation)**.** Given $\Gamma \vdash M_1, M_2 : \sigma$, we let $\Gamma \vdash M_1 \lesssim_{ciu} M_2 : \sigma$, when for all $\Gamma \vdash \sigma \preceq \Sigma'; \Gamma' \vdash \sigma'$ and Σ, h, K, γ, such that $\Sigma' \subseteq \Sigma$, $h : \Sigma$, $\mathrm{FV}(K) = \emptyset$, $\Sigma; \Gamma' \vdash K : (\Sigma'; \Gamma' : \sigma') \Rightarrow \mathsf{Unit}^{\perp}$, and $\Sigma'; \Gamma' \vdash \gamma$, we have $(K[M_1]\{\gamma\}, h) \Downarrow$ implies $(K[M_2]\{\gamma\}, h) \Downarrow$.

$\lesssim_{ciu}$ can be shown to subsume $\lesssim_{ctx}$ in the following sense.

Lemma 43 (CIU)**.** *$\Gamma \vdash M_1 \lesssim_{ctx} M_2 : \sigma$ implies $\Gamma \vdash M_1 \lesssim_{ciu} M_2 : \sigma$. If $\Gamma \vdash M_1, M_2 : \sigma$ is saturated, $\Gamma \vdash M_1 \lesssim_{ciu} M_2 : \sigma$ implies $\Gamma \vdash M_1 \lesssim_{ctx} M_2 : \sigma$.*

Dual LTS The approach to proving soundness in trace semantics is to consider the *composite interaction* between traces generated for the term and the context [4, 5]. The LTS $\mathcal{L}_P$ represents traces for the term, so next we provide another 'dual' LTS, called $\mathcal{L}_O$, to correspond to the choice h, K, γ in CIU testing. Its transition rules are presented in Figure 3. The key difference is that Sn (the set of available abstract states) now appears in active (P) configurations, and Υ is updated in P-actions. The assignment of abstract states to P-names (Υ) is based upon the locations present in the corresponding value stored in γ. To enable that, $\mathcal{L}_O$ includes an additional component Φ, the ***location map***, mapping $\bigcup \mathrm{img}(\Upsilon)$ to *empty* or *singleton* sets of locations in $\mathrm{dom}(h)$. In essence, Φ determines what location a given abstract state grants access to. The rules (PA) and (PQ) rely on the following auxiliary function, which is used to extract the abstract states S from the locations used in V and allocate new abstract states for locations not yet covered by abstract state.

Definition 44. Given location map Φ and value V, let $S = \Phi^{-1}(\mathrm{Loc(V)})$ and $L = \mathrm{Loc(V)} \setminus \Phi(S)$. Define $\mathrm{QLoc}_{\Phi}(V)$ to consist of all pairs $(S \uplus S', \Phi')$ such that $S' \subseteq \mathrm{AStates}$ is non-empty (and disjoint from S), $\mathrm{dom}(\Phi') = S'$, and

- if $L = \emptyset$ then $\Phi'(s) = \emptyset$ for all $s \in S'$;
- if $L \neq \emptyset$ then Φ' is a bijection from S' to $\{\{\ell\} \mid \ell \in L\}$.

Initialisation Let us now fix an r-free judgement $\Gamma \vdash M : \sigma$. Following the pattern of CIU equivalence (Definition 42), we will now define an initial configuration for $\mathcal{L}_O$ for any h, K, γ, where $\Gamma \vdash \sigma \preceq \Sigma'; \Gamma' \vdash \sigma'$, $\Sigma' \subseteq \Sigma$, $\Sigma \vdash h$, $\mathrm{FV}(K) = \emptyset$, $\Sigma; \Gamma' \vdash K : (\Sigma'; \Gamma' : \sigma') \Rightarrow \mathsf{Unit}^{\perp}$, and $\Sigma'; \Gamma' \vdash \gamma$. Let us fix a continuation name $\circ$, which we use to determine when the context has returned. Just as we needed a qualified (Γ, c)-assignment to initialise $\mathcal{L}_P$, this time we shall need an analogous notion for $\mathcal{L}_O$, to be defined next, which also needs to include Φ and ensure it is consistent with qualifiers in Γ'.

A ***location qualified*** (Γ', c)***-assignment*** is $(\rho, c, \Upsilon, \Phi)$, where ρ is a Γ-assignment, $c \in \mathrm{CNames} \setminus \{\circ\}$, Υ is P-consistent with $(\emptyset, N_{\rho,c})$, and Φ is such that, for any $x : \tau^Q \in \Gamma$ of function type, if $x : (\tau')^{Q'} \in \Gamma'$ then $Q' = Q \cup \bigcup \Phi(\Upsilon(x))$. $(\rho, c, \Upsilon, \Phi)$ is said to be ***compatible*** with γ if ρ and γ agree on values of base type. The initial configuration $\mathsf{C}^{\rho,c,\Upsilon,\Phi}_{\Gamma',h,K,\gamma}$ for $\mathcal{L}_O$ is then defined to be $(\langle \gamma_\rho,\ \mathrm{dom}(N_{\rho,c}) \uplus \{\circ\},\ h,\ \mathrm{Ty},\ \Psi,\ \nu(\rho),\ \Upsilon,\ \Phi\rangle,\ (c, (K, \circ), f_!, ()))$, where $\gamma_\rho = [\rho(x) \mapsto \gamma(x) \mid \gamma(x) \in \mathrm{FNames}]$, $\mathrm{Ty} = N_{\rho,c} \cdot [\circ \mapsto \mathsf{Unit}^{\perp}]$, $\Psi = [\circ \mapsto \emptyset, f_! \mapsto \mathrm{dom}(\gamma_\rho)]$.

Dual properties We can now consider the properties of $\mathcal{L}_O$, and see how they 'line-up' with those of $\mathcal{L}_P$.

Lemma 45. *Let Γ, σ be r-free and $\Sigma, \Gamma', h, K, \gamma$ be as in CIU. For a location qualified (Γ', c)-assignment $(\rho, c, \Upsilon, \Phi)$ compatible with γ, if $\mathsf{C}^{\rho,c,\Upsilon,\Phi}_{\Gamma',h,K,\gamma} \stackrel{t}{\Rightarrow} \mathbf{C}$ then t is a $([\circ \mapsto \mathsf{Unit}^{\perp}], N_{\rho,c}, \Upsilon)$-well-qualified O-trace whose qualifiers match the qualifying map $\Upsilon_{\mathbf{C}}$ in $\mathbf{C}$ and*

1) *t is bracketed;*
2) *t is $\Psi' P$-visible and Ψ'-compatible for some Ψ';*
3) *t is ΨO-visible, where Ψ is the reachability map of $\mathbf{C}$ restricted to $\mathrm{FNames}_!$.*

Qualification and 1) follow from the way $\mathcal{L}_O$ is set up. This time, 3) are the constraints that we have imposed to mirror 3) in Lemma 40, to match the behaviour of the term being placed in the context. Well-qualification and 2) are now the interesting properties, obtained by considering how the terms and contexts produced during evaluation can be typed.

The other key result about $\mathcal{L}_O$ follows from its construction, and relates the abstract state qualifiers to the locations appearing in components of an dual LTS configuration. This is important as it allows us to say that names with disjoint abstract state qualifications can only access disjoint locations.

Lemma 46. *Let $\Gamma \vdash \tau^Q$ be r-free and $\Sigma, \Gamma', h, K, \gamma$ be as in CIU. For $(\rho, c, \Upsilon, \Phi)$ a location qualified (Γ', c)-assignment compatible with γ, if $\mathsf{C}^{\rho,c,\Upsilon,\Phi}_{\Gamma',h,K,\gamma} \stackrel{t}{\rightarrow} \mathbf{C}$ (either $(\langle \gamma, \phi, h', \mathrm{Ty}, \Psi, Fn, \Upsilon, \Phi\rangle, \mathbf{S})$ or $(\langle M, c', \gamma, \phi, h', \mathrm{Ty}, \Psi, \Upsilon, Sn, \Phi\rangle, \mathbf{S})$), then*

- $\mathrm{Loc(M)} \subseteq \Phi(Sn) \cup \mathrm{Loc}_{\Phi}(M)$
- *for any $f \in \mathrm{dom}(\gamma)$,* $\mathrm{Loc(\gamma(f))} \subseteq \bigcup \Phi(\Upsilon(f))$
- *if $(c', (K', c''), f, A) \in \mathbf{S}_t$ then* $\mathrm{Loc(K')} \subseteq \bigcup \Phi(\Upsilon(c)) \cup \mathrm{Loc}_{\Phi}(K')$.

Given an (N_O, N_P, Υ)-qualified X-trace t, let us write $t^{\perp}$ for the (N_P, N_O, Υ)-qualified $\overline{X}$-trace obtained by changing the polarity of each name. Equipped with Lemmata 40 and 45, we can arrive at a correctness result for contextual interactions:

$(P\tau)$ $\langle M, c, \gamma, \phi, h, \mathrm{Ty}, \Psi, \Upsilon, Sn, \Phi\rangle \xrightarrow{\tau} \langle N, c, \gamma, \phi, h', \mathrm{Ty}, \Psi, \Upsilon, Sn, \Phi\rangle$
when $(M, h) \to (N, h')$

(PA) $\langle V, c, \gamma, \phi, h, \mathrm{Ty}, \Psi, \Upsilon, Sn, \Phi\rangle \xrightarrow{\overline{c}(A^S)} \langle \gamma\cdot\gamma', \phi \uplus \nu(A), h, \mathrm{Ty}\cdot\mathrm{Ty}', \Psi, \Psi(c)\cup\nu(A), \Upsilon\cdot\Upsilon', \Phi\cdot\Phi'\rangle$
when $A \in \mathbf{AVal}(V)$, $\gamma' = [\nu(A) \mapsto V]$, $\mathrm{Ty}' = [\nu(A) \mapsto \mathrm{Ty}(c)]$, $\Upsilon' = [\nu(A) \mapsto S]$.
If $\nu(A) = \emptyset$ then $\Phi' = \emptyset$. Otherwise $(S, \Phi') \in \mathrm{QLoc}_\Phi(V)$.

(PQ) $\langle K[f\,V], c', \gamma, \phi, h, \mathrm{Ty}, \Psi, \Upsilon, Sn, \Phi\rangle \xrightarrow{\overline{f}(A^S, c^{S'})/(c,(K,c')),(f,A))} \langle \gamma\cdot\gamma', \phi \uplus \nu(A) \uplus \{c\}, h, \mathrm{Ty}'', \Psi, \Psi(f)\cup\nu(A), \Upsilon'', \Phi\cdot\Phi'\rangle$
when $A \in \mathbf{AVal}(V)$, $\gamma' = [\nu(A) \mapsto V]$, $\mathrm{Ty}(f) = (\mu g.(x : \tau_1^{Q_1}) \to \tau_2^{Q_2})^Q$, $\mathrm{Ty}' = [\nu(A) \mapsto \tau_1^{Q_1}, c \mapsto \tau_2^{Q_2\{f/g\}\{\nu(A)/x\}}]$,
$\Upsilon' = [\nu(A) \mapsto S, c \mapsto S']$
If $\nu(A) = \emptyset$ then $S' = Sn$ and $\Phi' = \emptyset$. Otherwise $(S, \Phi') \in \mathrm{QLoc}_\Phi(V)$, $S' = Sn \uplus \mathrm{dom}(\Phi')$.

(OA) $\langle \gamma, \phi, h, \mathrm{Ty}, \Psi, Fn, \Upsilon, \Phi\rangle \xrightarrow{c^S(A),(c,(K,c')),(f,A'))} \langle K[A], c', \gamma, \phi \uplus \nu(A), h, \mathrm{Ty}\cdot\mathrm{Ty}', \Psi', \Upsilon, S, \Phi\rangle$
when $S = \Upsilon(c)$, $\mathrm{Ty}(c) = \tau^Q$, $A \in \mathbf{AVal}_{ty}(\tau)$, $\mathrm{Ty}' = [\nu(A) \mapsto \mathrm{Ty}(c)]$, $\Psi' = [\nu(A) \mapsto F]$
If $\nu(A) = \emptyset$ then F not needed.
Otherwise $\mathrm{ArgQ}(\tau) \subseteq F \subseteq Fn$ and $\mathcal{R}^\Psi_{\mathrm{Ty}}(F; Fn') \subseteq Q\{\Psi(f)/f\}$, where $Fn' = \Psi(f) \cup \nu(A')$.

(OQ) $\langle \gamma, \phi, h, \mathrm{Ty}, \Psi, Fn, \Upsilon, \Phi\rangle \xrightarrow{f^S(A,c)} \langle V\,A, c, \gamma, \phi \uplus \{c\} \uplus \nu(A), h, \mathrm{Ty}\cdot\mathrm{Ty}', \Psi\cdot\Psi', \Upsilon, S, \Phi\rangle$
when $f \in Fn$, $S = \Upsilon(f)$, $\gamma(f) = V$, $\mathrm{Ty}(f) = (\mu g.(x : \tau_1^{Q_1}) \to \tau_2^{Q_2})^Q$, $A \in \mathbf{AVal}_{ty}(\tau_1)$, $Q_2 \sqsubseteq Fn \cup \{g, x\}$,
$\mathrm{Ty}' = [\nu(A) \mapsto \tau_1^{Q_1}, c \mapsto \tau_2^{Q_2\{f/g\}\{\nu(A)/x\}}]$, $\Psi' = [\nu(A) \mapsto F, c \mapsto Fn]$,
If $\nu(A) = \emptyset$ then F not needed. Otherwise $\mathrm{ArgQ}(\tau_1) \subseteq R \subseteq Fn$ and $(\bigcup_{g\in F} \mathcal{R}^\Psi_{\mathrm{Ty}}(g)) \cap \mathcal{R}^\Psi_{\mathrm{Ty}}(f) \subseteq \mathcal{R}^\Psi_{\mathrm{Ty}}(Q)$.

Fig. 3: The dual transition system $\mathcal{L}_O$

any terminating contextual interaction is based on a complete trace t generated by the program and a matching trace $t^\perp\,\overline{\circ}(())$ generated by the context.

Lemma 47 (Correctness)**.** *Suppose* $\Gamma \vdash M : \sigma$ *is r-free,* $\Sigma, \Gamma', h, K, \gamma$ *are as in CIU, and* $(\rho, c, \Upsilon, \Phi)$ *is a location qualified* (Γ', c)*-assignment compatible with* γ*. Then* $(K[M]\{\gamma\}, h) \Downarrow$ *iff there exists a complete* $t \in \mathbf{Tr}(\mathsf{C}^{\rho,c,\Upsilon}_{\Gamma\vdash M:\sigma})$ *such that* $t^\perp\,\overline{\circ}(()) \in \mathbf{Tr}(\mathsf{C}^{\rho,c,\Upsilon,\Phi}_{\Gamma',h,K,\gamma})$.

It follows that complete trace inclusion is sound for $\lesssim_{ciu}$.

Theorem 48. *For any r-free* $\Gamma \vdash M_1, M_2 : \sigma$, $\mathbf{Tr}(\Gamma \vdash M_1 : \sigma) \subseteq \mathbf{Tr}(\Gamma \vdash M_2 : \sigma)$ *implies* $\Gamma \vdash M_1 \lesssim_{ciu} M_2 : \sigma$.

Proof. Let $\Gamma \vdash \sigma \preceq \Sigma'; \Gamma' \vdash \sigma'$, $\Sigma' \subseteq \Sigma$, $\Sigma \vdash h$, $\mathrm{FV}(K) = \emptyset$, $\Sigma; \Gamma' \vdash K : (\Sigma'; \Gamma' : \sigma') \Rightarrow \mathsf{Unit}^\perp$, $\Sigma; \Gamma' \vdash \gamma$, and $(K[M_1]\{\gamma\}, h) \Downarrow$. We can find some $(\rho, c, \Upsilon, \Phi)$ a location qualified (Γ', c)-assignment compatible with γ. Finding ρ and c is clearly always possible, as we are simply supplying fresh names for functions in γ. So we need to specify Υ and Φ. We can initially consider taking abstract states to be in correspondence with locations in qualifiers in Γ'. This might lead some set $\Upsilon(f)$ to be empty. This can be resolved by introducing fresh abstract states, and mapping them to an empty set in Φ. By Lemma 47 (left-to-right), there exists t such that $t \in \mathbf{Tr}(\mathsf{C}^{\rho,c,\Upsilon}_{\Gamma\vdash M_1:\tau^Q})$ and $t^\perp\,\overline{\circ}(()) \in \mathbf{Tr}(\mathsf{C}^{\rho,c,\Upsilon,\Phi}_{\Gamma',h,K,\gamma})$. Furthermore, we have that t is complete. By $\mathbf{Tr}(\Gamma \vdash M_1 : \tau^Q) \subseteq \mathbf{Tr}(\Gamma \vdash M_2 : \tau^Q)$, we have that $t \in \mathbf{Tr}(\mathsf{C}^{\rho,c,\Upsilon}_{\Gamma\vdash M_2:\tau^Q})$. As $t \in \mathbf{Tr}(\mathsf{C}^{\rho,c,\Upsilon}_{\Gamma\vdash M_2:\tau^Q})$ and $t^\perp\,\overline{\circ}(()) \in \mathbf{Tr}(\mathsf{C}^{\rho,c,\Upsilon,\Phi}_{\Gamma',h,K,\gamma})$, by Lemma 47 (right-to-left) we can conclude $(K[M_2]\{\gamma\}, h) \Downarrow$. Thus, we have that $\Gamma \vdash M_1 \lesssim_{ciu} M_2 : \tau^Q$. □

VII. Rearrangements

In Lemmata 40 and 45 we identified a number of technical properties that characterise the traces produced by the program and the context respectively. We dub them relevant below.

Definition 49. An (N_O, N_P, Υ)-qualified X-trace is ***relevant*** if it is well-qualified, bracketed, X-visible as well as $\Psi\overline{X}$-visible and Ψ-compatible for some Ψ.

While complete trace inclusion gives us a sound model for $\lesssim_{ciu}$ (Theorem 48), it cannot be expected to be complete. This is because, under reachability restrictions, contexts may have restricted power to observe the order of in which certain actions are generated or even if they are generated at all. We formalise the intuitions next.

Note that in a qualified X-trace the head names of X-actions are annotated with sets of abstract states. We say that two such actions are ***conflicting*** if the annotations are *not* disjoint, i.e. they share an abstract state.

Definition 50. Let $X \in \{O, P\}$ and t, t' be relevant X-traces. We define $t \preceq_X t'$ to hold if t is obtained by removing some actions (possibly none) in t' and permuting them so that:

- for any conflicting X-actions $\mathbf{a}_1, \mathbf{a}_2$ in t' s.t. $\mathbf{a}_1$ occurs before $\mathbf{a}_2$, if $\mathbf{a}_2$ occurs in t then $\mathbf{a}_1$ occurs in t before $\mathbf{a}_2$;
- each X-action in t must be immediately followed by the same $\overline{X}$-action as in t'.

We will write $t \simeq_X t'$, when t is a permutation of t'. Given fixed t', the set of all t such that $t \preceq_X t'$ (resp. $t \simeq_X t'$) will be denoted by $\Pi_X(t')$ (resp. $\Pi_X(t')$).

Example 51. Recall $\mathtt{t}_1, \mathtt{t}_2$ from Example 20. Define $\mathtt{t}'_1 = \overline{h^l}(), c_1)\ c_1(f^{l,m})\ \overline{h^l}(), c_3)\ c_3(g^{l,n})\ \overline{g^{l,n}}(), c_5)\ c_5(())\ \overline{f^{l,m}}(), c_4)\ c_4(())\ \overline{c^l}(())$ and $\mathtt{t}'_2 = \overline{h^l}(), c_1)\ c_1(f^m)\ \overline{h^l}(), c_3)\ c_3(g^n)\ \overline{g^n}(), c_5)\ c_5(())\ \overline{f^m}(), c_4)\ c_4(())\ \overline{c^l}(())$. We have $\mathtt{t}_2 \simeq_P \mathtt{t}'_2$, but neither $\mathtt{t}_1 \preceq_P \mathtt{t}'_1$ nor $\mathtt{t}'_1 \preceq_P \mathtt{t}_1$.

Example 52. For $\mathtt{t}_1 = \overline{h^S}(), c_1)\ c_1(f^T)\ \overline{f^T}(1, c_2)\ c_2(())\ \overline{f^T}(2, c_3)\ c_3(())\ \overline{c_0^S}(())$ and $\mathtt{t}_2 = \overline{h^S}(), c_1)\ c_1(f^T)\ \overline{f^T}(1, c_2)$

$c_2(())\,\overline{c_0^S}(())$, with $S \cap T = \emptyset$ then $\mathtt{t}_2 \preceq_P \mathtt{t}_1$.

Remark 53. At this point we can explain the decision to include continuation names in traces, which may appear unnecessary as the absence of control-flow operators means they can be recovered from the bracketing condition (cf. [4, 5]). The presence of continuation names makes the actions of a trace unique, as they either uniquely introduce a continuation name (in questions), or use one (in answers). This makes the above Definition well defined, as we can uniquely match actions between traces. It also ensures reorderings cannot change the pointer structure of traces, so a question is always answered in the same way, in all reorderings of a trace.

The following result captures the limitations on what contexts can observe about a trace. The proof of this fact depends crucially on Lemma 46, which allows us to reason about the impact of changing the order on actions in a trace has on the values in the heap of a configuration.

Lemma 54 (Closure under $\mathrm{II_O}$)**.** *Let Γ', h, K, γ, ρ, Υ, Φ and c be such that $\mathsf{C}^{\rho,c,\Upsilon,\Phi}_{\Gamma',h,K,\gamma}$ is a context configuration. If $t \in \mathbf{Tr}(\mathsf{C}^{\rho,c,\Upsilon,\Phi}_{\Gamma',h,K,\gamma})$, then $\mathrm{II_O}(t) \subseteq \mathbf{Tr}(\mathsf{C}^{\rho,c,\Upsilon,\Phi}_{\Gamma',h,K,\gamma})$.*

This motivates the order on traces introduced below.

Definition 55. $\mathbf{Tr}(\Gamma \vdash M_1 : \tau^Q) \subseteq_{\mathrm{II}} \mathbf{Tr}(\Gamma \vdash M_2 : \tau^Q)$ is defined to hold if, for all $((\rho, c, \Upsilon), t) \in \mathbf{Tr}(\Gamma \vdash M_1 : \tau^Q)$, there exists $t' \in \mathrm{II_P}(t)$ such that $((\rho, c, \Upsilon), t') \in \mathbf{Tr}(\Gamma \vdash M_2 : \tau^Q)$.

Using Lemmata 47, 54, one can prove that $\subseteq_{\mathrm{II}}$ is sound for $\lesssim_{ciu}$. To obtain completeness, we show that the relevant traces correspond to some interaction with a context. This is demonstrated by the definability result below. Because of Lemma 54, definability can only hold up to $\mathrm{II_O}$.

Lemma 56 (Definability)**.** *Let $\Gamma \vdash \sigma$ be an r-free judgement, and (ρ, c, Υ) a qualified (Γ, c)-assignment. Suppose $t\,\overline{\circ}(())$ is a relevant $([\circ \mapsto \mathsf{Unit}^\perp], N_{\rho,c}, \Upsilon)$-qualified O-trace, i.e. $t^\perp$ is a complete $(N_{\rho,c}, \emptyset, \Upsilon)$-qualified P-trace. There exists a passive configuration $\mathbf{C}$ such that $t\,\overline{\circ}(()) \in \mathbf{Tr}(\mathbf{C})$, and, whenever $t'\,\overline{\circ}(()) \in \mathbf{Tr}(\mathbf{C})$, then $t'\,\overline{\circ}(()) \in \mathrm{II_O}(t\,\overline{\circ}(()))$ (up to renaming via permutations on* Names *and* AStates *preserving $N_{\rho,c} \uplus \{\circ\}$ and Υ). Moreover, there exists Γ', h, K, γ as in CIU, and Φ such that $\mathbf{C} = \mathsf{C}^{\rho,c,\Upsilon,\Phi}_{\Gamma',h,K,\gamma}$.*

Now, thanks to Lemmata 47, 54 and 56, we arrive at the first full abstraction result.

Theorem 57 (Full Abstraction for $\lesssim_{ciu}$)**.** *For any r-free judgements $\Gamma \vdash M_1, M_2 : \tau^Q$, we have $\Gamma \vdash M_1 \lesssim_{ciu} M_2 : \sigma$ iff $\mathbf{Tr}(\Gamma \vdash M_1 : \tau^Q) \subseteq_{\mathrm{II}} \mathbf{Tr}(\Gamma \vdash M_2 : \tau^Q)$.*

Finally, we proceed to investigate $\simeq_{ciu}$. By the result above, for $\simeq_{ciu}$-equivalent terms, we obtain $\mathbf{Tr}(\Gamma \vdash M_1 : \tau^Q) \subseteq_{\mathrm{II}} \mathbf{Tr}(\Gamma \vdash M_2 : \tau^Q) \subseteq_{\mathrm{II}} \mathbf{Tr}(\Gamma \vdash M_1 : \tau^Q)$. Hence, for any t_1 in $\mathbf{Tr}(\Gamma \vdash M_1 : \tau^Q)$, there exist t_2 from $\mathbf{Tr}(\Gamma \vdash M_2 : \tau^Q)$ and t_1' from $\mathbf{Tr}(\Gamma \vdash M_1 : \tau^Q)$ such that $t_1' \preceq_P t_2 \preceq_P t_1$. Note that t_1, t_1' are generated by the same term and, by $t_1' \preceq_P t_1$, the first action where they could differ must be a P-action. Consequently, M_1 must have generated the same trace up to that action. Thus, the active states that have been reached are nearly the same (up to Ψ). Consequently, they contain the same term, meaning that the next action is bound to be the same. Hence, $t_1 = t_1'$. Thus, t_2 must be a permutation of t_1. Consequently, $\simeq_{ciu}$ is characterised by permuted traces.

Definition 58. The ***permutation semantics*** of an r-free judgement $\Gamma \vdash M : \tau^Q$ is defined to be $\mathbf{Perm}(\Gamma \vdash M : \tau^Q) \triangleq \{((\rho, c, \Upsilon), t) | ((\rho, c, \Upsilon), t') \in \mathbf{Tr}(\Gamma \vdash M : \tau^Q),\ t \in \Pi_P(t')\}$.

Theorem 59 (Full Abstraction for $\simeq_{ciu}$)**.** *For any r-free judgements $\Gamma \vdash M_1, M_2 : \tau^Q$, $\Gamma \vdash M_1 \simeq_{ciu} M_2 : \sigma$ iff $\mathbf{Perm}(\Gamma \vdash M_1 : \sigma) = \mathbf{Perm}(\Gamma \vdash M_2 : \sigma)$.*

Using Lemma 43, we can now derive our final result.

Corollary 60 (Full Abstraction)**.** *For any r-free saturated judgements $\Gamma \vdash M_1, M_2 : \tau^Q$, $\Gamma \vdash M_1 \lesssim_{ctx} M_2 : \sigma$ iff $\mathbf{Tr}(\Gamma \vdash M_1 : \tau^Q) \subseteq_{\mathrm{II}} \mathbf{Tr}(\Gamma \vdash M_2 : \tau^Q)$, and $\Gamma \vdash M_1 \simeq_{ctx} M_2 : \sigma$ iff $\mathbf{Perm}(\Gamma \vdash M_1 : \sigma) = \mathbf{Perm}(\Gamma \vdash M_2 : \sigma)$.*

Example 61. Recall the (in)equivalences discussed in the Introduction. If $h : (\mu h.\mathsf{Unit}^\perp \to (\mathsf{Unit}^\perp \to \mathsf{Unit}^\perp)^{\{h\}})^\emptyset$, the Corollary implies inequivalence because of $\mathtt{t}_1 \not\simeq_P \mathtt{t}_1'$ (Example 51). When $h : (\mu h.\mathsf{Unit}^\perp \to (\mathsf{Unit}^\perp \to \mathsf{Unit}^\perp)^\emptyset)^\emptyset$, $\mathtt{t}_1, \mathtt{t}_1'$ do not arise and $\mathtt{t}_2 \simeq_P \mathtt{t}_2'$, so the terms are equivalent.

Example 62. Recall the approximation example from the Introduction. The terms generate traces of the form $\mathtt{t}_1, \mathtt{t}_2$ from Example 52, respectively. Because $\mathtt{t}_2 \preceq_P \mathtt{t}_1$, the set of traces of the left is related to that of the right by $\subseteq_{\mathrm{II}}$, so by Corollary 60, the approximation follows.

VIII. Conclusion

We presented the first full abstraction result for a language with reachability types. We show how restricting aliasing complicates the notion of contextual testing, leading to a refined notion of visibility, and the use of a novel reordering of traces to capture the inability of contexts to observe sequentiality.

Since their introduction [1], reachability types have been extended with polymorphism [2], and it would be interesting to investigate combining this with the polymorphic trace semantics of [18]. The language $\lambda_\diamond$ of [2] also allows for functions to be observably 'pure' (without local state), which poses a challenge for trace semantics, though recent work [19] has achieved full abstraction for a pure language using traces.

Lifting the r-free restriction would require carefully disclosing locations [4, 13], which would have to interact with abstract state in some way. Reasoning techniques could also be built upon the LTS [20]. Melliès has characterised innocence using closure of strategies under permutation of independent moves using Mazurkiewicz traces [21], and it would be interesting to understand the connection to our approach.

References

[1] Y. Bao, G. Wei, O. Bracevac, Y. Jiang, Q. He, and T. Rompf, "Reachability types: tracking aliasing and separation in higher-order functional programs," *Proc. ACM Program. Lang.*, vol. 5, no. OOPSLA, pp. 1–32, 2021.

[2] G. Wei, O. Bračevac, S. Jia, Y. Bao, and T. Rompf, "Polymorphic Reachability Types: Tracking Freshness, Aliasing, and Separation in Higher-Order Generic Programs," *Proc. ACM Program. Lang.*, vol. 8, no. POPL, pp. 393–424, 2024.

[3] O. Bračevac, G. Wei, S. Jia, S. Abeysinghe, Y. Jiang, Y. Bao, and T. Rompf, "Graph IRs for Impure Higher-Order Languages: Making Aggressive Optimizations Affordable with Precise Effect Dependencies," *Proc. ACM on Program. Lang.*, vol. 7, no. OOPSLA2, pp. 400–430, 2023.

[4] J. Laird, "A fully abstract trace semantics for general references," in *Proceedings of ICALP*, ser. Lecture Notes in Computer Science. Springer, 2007, vol. 4596, pp. 667–679.

[5] G. Jaber and A. S. Murawski, "Complete trace models of state and control," in *Proceedings of ESOP*, ser. Lecture Notes in Computer Science, vol. 12648. Springer, 2021, pp. 348–374.

[6] S. Abramsky and G. McCusker, "Game semantics," in *Logic and Computation*, H. Schwichtenberg and U. Berger, Eds. Springer-Verlag, 1998, proceedings of the NATO Advanced Study Institute, Marktoberdorf.

[7] A. S. Murawski and N. Tzevelekos, "An invitation to game semantics," *ACM SIGLOG News*, vol. 3, no. 2, pp. 56–67, 2016.

[8] D. Sangiorgi, "A theory of bisimulation for the pi-calculus," *Acta Inf.*, vol. 33, no. 1, pp. 69–97, 1996.

[9] S. B. Lassen and P. B. Levy, "Typed normal form bisimulation," in *Proceedings of CSL*, ser. Lecture Notes in Computer Science. Springer, 2007, vol. 4646, pp. 283–297.

[10] S. Abramsky and G. McCusker, "Call-by-value games," in *Proceedings of CSL*, ser. Lecture Notes in Computer Science, vol. 1414. Springer-Verlag, 1997, pp. 1–17.

[11] Y. Bao, G. Wei, O. Bracevac, and T. Rompf, "Modeling Reachability Types with Logical Relations," *CoRR*, vol. abs/2309.05885, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2309.05885

[12] [Online]. Available: https://github.com/TiarkRompf/reachability/tree/main/base/lambda_star_full

[13] A. S. Murawski and N. Tzevelekos, "Full abstraction for Reduced ML," *Annals of Pure and Applied Logic*, vol. 164, no. 11, pp. 1118–1143, Nov. 2013.

[14] J. Laird, "A game semantics of names and pointers," *Annals of Pure and Applied Logic*, vol. 151, no. 2–3, pp. 151–169, Feb. 2008.

[15] J. M. E. Hyland and C.-H. L. Ong, "On Full Abstraction for PCF: I. Models, observables and the full abstraction problem, II. Dialogue games and innocent strategies, III. A fully abstract and universal game model," *Information and Computation*, vol. 163(2), pp. 285–408, 2000.

[16] J. Laird, "Full abstraction for functional languages with control," in *Proceedings of LICS*, 1997, pp. 58–67.

[17] I. Mason and C. Talcott, "Equivalence in functional languages with effects," *Journal of Functional Programming*, vol. 1, no. 3, pp. 287–327, Jul. 1991.

[18] G. Jaber and N. Tzevelekos, "A trace semantics for System F parametric polymorphism," in *Proceedings of FOSSACS*, ser. Lecture Notes in Computer Science, vol. 10803. Springer, 2018, pp. 20–38.

[19] V. Koutavas, Y.-Y. Lin, and N. Tzevelekos, "Fully abstract normal form bisimulation for call-by-value PCF," in *Proceedings of LICS*. IEEE, 2023, pp. 1–13.

[20] G. Jaber and A. S. Murawski, "Compositional relational reasoning via operational game semantics," in *Proceedings of LICS*. IEEE, 2021, pp. 1–13.

[21] P.-A. Melliès, "Asynchronous games 2: The true concurrency of innocence," vol. 358, no. 2–3, pp. 200–228.