A Trace Semantics for System F Parametric Polymorphism

Guilhem Jaber[1] and Nikos Tzevelekos[2]

[1] ENS de Lyon, Université de Lyon, LIP
[2] Queen Mary University of London

Abstract. We present a trace model for Strachey parametric polymorphism. The model is built using operational nominal game semantics and captures parametricity by using names. It is used here to prove an operational version of a conjecture of Abadi, Cardelli, Curien and Plotkin which states that Strachey equivalence implies Reynolds equivalence in System F.

1 Introduction

Parametricity was first introduced by Strachey [22] as a way to characterise the behaviour of polymorphic programs as being uniform with respect to the type of the arguments provided. He opposed this notion to ad-hoc polymorphism, where a function can produce arbitrarily different outputs when provided inputs of different types (for example an integer and a boolean). To formalise this notion of parametricity, Reynolds introduced relational parametricity [21]. It is defined using an equivalence on programs, that we call Reynolds equivalence and is a generalisation of logical relations to System F. This equivalence uses arbitrary relations over pairs of types to relate polymorphic programs. So a parametric program that takes related arguments as input will produce related results. Reynolds parametricity has been developed into a fundamental theory for studying polymorphic programs [23,1,20].

Following results of Mitchell on PER-models of polymorphism [18], Abadi, Cardelli, Curien and Plotkin [1,20] introduced another, more intentional notion of equivalence, called Strachey equivalence. Two terms of System F are Strachey equivalent whenever, by removing all their type annotations, we obtain two $\beta\eta$-equivalent untyped terms. The authors conjectured that Strachey equivalence implies Reynolds equivalence (the converse being easily shown to be false).

In this paper we examine a notion of Reynolds equivalence based on operational logical relations, and prove that, for this notion, the conjecture holds. To do so, we introduce a trace model for System F based on operational nominal game semantics [14,12]. Terms in our model are denoted as sets of traces, generated by a labelled transition system, which represent interactions with arbitrary term contexts. In order to abstract away type information from inputs to polymorphic functions, our semantics uses *names* to model such inputs. The idea is the following: since names have no internal structure, the function has no choice but to act "the same way" on such inputs, i.e. be parametric. Our trace model yields a third notion of equivalence: trace equivalence (i.e. equality of sets of traces). Then, the result is proven by showing that trace equivalence is included in (operational) Reynolds equivalence, while it includes Strachey equivalence.

$$\frac{\Delta; \Gamma, x : \theta \vdash M : \theta'}{\Delta; \Gamma \vdash \lambda x^{\theta}.M : \theta \to \theta'} \qquad \frac{\Delta; \Gamma \vdash M : \theta \to \theta' \qquad \Delta; \Gamma \vdash N : \theta}{\Delta; \Gamma \vdash MN : \theta'} \qquad \begin{array}{|r} (\lambda x.M)N =_{\beta\eta} M\{N/x\} \\ (\Lambda X.M)\theta =_{\beta\eta} M\{\theta/X\} \\ \lambda x.Mx =_{\beta\eta} M \\ \Lambda X.MX =_{\beta\eta} M \end{array}$$

$$\frac{(x : \theta) \in \Gamma}{\Delta; \Gamma \vdash x : \theta} \qquad \frac{\Delta, X; \Gamma \vdash M : \theta}{\Delta; \Gamma \vdash \Lambda X.M : \forall X.\theta} \qquad \frac{\Delta; \Gamma \vdash M : \forall X.\theta}{\Delta; \Gamma \vdash M\theta' : \theta\{\theta'/X\}}$$

Fig. 1. Typing rules and $\beta\eta$-equality axioms.

The traces in our model are formed of *moves*, which represent interactions between the modelled term (the *Player*) and its context (the *Opponent*): either of Player or Opponent can interrogate the terms provided by the other one, or respond to a previous such interrogation. These moves are called *questions* and *answers* respectively. Names enter the scene when calling terms which are of polymorphic type, in which case the calling party would replace the actual argument type θ with a *type name* α, and record locally the correspondence between α and θ. Another use of names in our model is for representing terms that are passed around as arguments to questions. These are called *computation names*, and are typed according to the term they each represent.

2 Definition of System F and Parametricity

We start off by giving the definitions of System F and of the parametric equivalence relations we shall examine on it. The grammar for System F is standard and given by:

$$\begin{array}{rl} \mathsf{Type} \ni & \theta, \theta' ::= X \mid \theta \to \theta' \mid \forall X.\theta \\ \mathsf{Term} \ni & M, N ::= \lambda x^{\theta}.M \mid \Lambda X.M \mid MN \mid M\theta \end{array}$$

We write x, etc. for *(term) variables*, sourced from a countable set Var; and X, etc. for *type variables*, taken from TVar. We define substitutions of open variables of either kind in the usual capture-avoiding way. For instance, the term obtained by consecutively applying substitutions $\eta : \mathsf{Var} \rightharpoonup \mathsf{Term}$ and $\delta : \mathsf{TVar} \rightharpoonup \mathsf{Type}$ on M is written $M\{\eta\}\{\delta\}$.

Terms are typed in environments $\Delta; \Gamma$, where Δ is a finite set of type variables, and Γ is a set $\{x_1 : \theta_1, \ldots, x_m : \theta_m\}$ of variable-type pairs. The typing rules are given in Figure 1. The operational semantics we examine is $\beta\eta$-equality, defined as the least syntactic congruence $=_{\beta\eta}$ that includes the axioms given on the RHS part of Figure 1.

We shall use the following common polymorphic encodings:

- $\mathbf{Bool} = \forall X.\ X \to X \to X$, $\mathbf{true} = \Lambda X.\lambda x^X.\lambda y^X.x$ and $\mathbf{false} = \Lambda X.\lambda x^X.\lambda y^X.y$,
- $\mathbf{Unit} = \forall X.\ X \to X$ and $\mathbf{id} = \Lambda X.\lambda x^X.x$.

Reynolds Equivalence We next introduce logical relations for System F. First, we let Rel be the set of all typed relations between closed terms that are compatible with $=_{\beta\eta}$:

$$\begin{aligned} \mathrm{Rel} = \{(\theta_1, \theta_2, R) \mid R \subseteq \mathsf{Term} \times \mathsf{Term} \wedge \forall (M_1, M_2) \in R.\ \cdot;\cdot \vdash M_i : \theta_i \\ \wedge\ \forall M_1' =_{\beta\eta} M_1. \forall M_2' =_{\beta\eta} M_2.\ (M_1', M_2') \in R\} \end{aligned}$$

Logical relations $\mathcal{R}[\![\theta]\!]_\delta$ are defined below, indexed by environments $\delta : \mathsf{TVar} \rightharpoonup \mathrm{Rel}$:

$$\begin{aligned} \mathcal{R}[\![X]\!]_\delta &= R \text{ when } \delta(X) = (\_, \_, R) \\ \mathcal{R}[\![\forall X.\theta]\!]_\delta &= \{(M_1, M_2) \mid \forall (\theta_1, \theta_2, R) \in \mathrm{Rel}.\ (M_1\theta_1, M_2\theta_2) \in \mathcal{R}[\![\theta]\!]_{\delta \cdot [X \mapsto (\theta_1, \theta_2, R)]}\} \\ \mathcal{R}[\![\theta_1 \to \theta_2]\!]_\delta &= \{(M_1, M_2) \mid \forall (N_1, N_2) \in \mathcal{R}[\![\theta_1]\!]_\delta.\ (M_1N_1, M_2N_2) \in \mathcal{R}[\![\theta_2]\!]_\delta\} \end{aligned}$$

We can now define the first notion of parametric equivalence for System F.

Definition 1. Given terms $\Delta; \Gamma \vdash M_1, M_2 : \theta$, we say that they are ***Reynolds equivalent***, and write $\Delta; \Gamma \vdash M_1 \simeq_{log} M_2 : \theta$, if:

$$\forall \delta \in \mathcal{R}[\![\Delta]\!]. \forall (\eta_1, \eta_2) \in \mathcal{R}[\![\Gamma]\!]_\delta.\ (M_1\{\eta_1\}\{\delta_1\}, M_2\{\eta_2\}\{\delta_2\}) \in \mathcal{R}[\![\theta]\!]_\delta$$

where $\mathcal{R}[\![\Delta]\!] = \mathrm{dom}(\Delta) \to \mathrm{Rel}$, $\delta_1 = \{(X, \theta_1) \mid \delta(X) = (\theta_1, \_, \_)\}$ (similar for δ_2) and $\mathcal{R}[\![\Gamma]\!]_\delta = \{(\eta_1, \eta_2) \in (\mathrm{dom}(\Gamma) \rightharpoonup \mathsf{Term})^2 \mid \forall (x, \theta') \in \Gamma.\ (\eta_1(x), \eta_2(x)) \in \mathcal{R}[\![\theta']\!]_\delta\}$.

The following result is standard [21].

Theorem 2 (Fundamental Property). *If $\Delta; \Gamma \vdash M : \tau$ then $\Delta; \Gamma \vdash M \simeq_{log} M : \theta$.*

Remark 3. Note that our definition of Reynolds equivalence does not coincide with either of the definitions given in [1,20]: therein, parametricity is defined using relational logics (and accompanying proof systems), whereas here we use quantification over concrete relations over closed terms.

Strachey Equivalence Another notion of parametric equivalence is defined by means of erasing types from terms. We define the *type erasure* $\mathbf{erase}(M)$ of a term M by:

$$\begin{array}{ll} \mathbf{erase}(\Lambda X.M) = \mathbf{erase}(M) & \mathbf{erase}(MN) = \mathbf{erase}(M)\mathbf{erase}(N) \\ \mathbf{erase}(\lambda x^\theta.M) = \lambda x.\mathbf{erase}(M) & \mathbf{erase}(M\theta) = \mathbf{erase}(M) \end{array}$$

and $\mathbf{erase}(x) = x$. Thus, $\mathbf{erase}(M)$ is an untyped λ-term. Below we overload $=_{\beta\eta}$ to also mean $\beta\eta$-equality in the untyped λ-calculus.

Definition 4. Given terms $\Delta; \Gamma \vdash M_1, M_2 : \theta$, we say that they are ***Strachey equivalent*** if $\mathbf{erase}(M_1) =_{\beta\eta} \mathbf{erase}(M_2)$.

It was conjectured in [1,20] that Reynolds equivalence includes Strachey equivalence. We prove this holds for the version of Reynolds equivalence given in Definition 1.

Theorem 5. *Any two Strachey equivalent terms are also Reynolds equivalent.*

It is interesting to think why a direct approach would not work in order to prove this conjecture. Given Strachey equivalent terms M_1, M_2 of type $\mathbf{Bool}$, suppose we want to prove them Reynolds equivalent. We therefore take $(\theta_1, \theta_2, R) \in \mathrm{Rel}$, $(N_{1,1}, N_{2,1}) \in R$, and $(N_{1,2}, N_{2,2}) \in R$, and aim to prove that $(M_1\theta_1 N_{1,1} N_{1,2}, M_2\theta_2 N_{2,1} N_{2,2}) \in R$. Ideally, we would like to prove that there exists $j \in \{1, 2\}$ s.t. for all $i \in \{1, 2\}$, $M_i\theta_i N_{i,1} N_{i,2} =_{\beta\eta} N_{i,j}$, but that seems overly optimistic. A first trick is to use Theorem 2, to get that M_2 is related with itself. Thus, we get that $(M_2\theta_1 N_{1,1} N_{1,2}, M_2\theta_2 N_{2,1} N_{2,2}) \in R$, and it would suffice to prove $M_1\theta_1 N_{1,1} N_{1,2} =_{\beta\eta} M_2\theta_1 N_{1,1} N_{1,2}$ to conclude. However, our hypothesis is simply that $\mathbf{erase}(M_1) =_{\beta\eta} \mathbf{erase}(M_2)$.

A possible solution to the above could be to β-*reduce* both $M_i\theta_1 N_{1,1} N_{1,2}$, hoping that the distinction between the two terms will vanish. Our trace semantics provides a way to model the interaction between such a term M_i and a context $\bullet\, \theta_j N_{j,1} N_{j,2}$, and to deduce properties about the normal form reached by their application via head reduction.

3 A nominal trace semantics for System F

In this section we introduce a trace semantics for open terms which will be our main vehicle of study for System F. The terms in our semantics will be allowed to contain special constants representing any term that could fill in their open variables (these be term or type variables). The use of names can be seen as a nominal approach to parametricity: parametric types and values are represented in our semantics by names, without internal structure. Thus, e.g. a parametric function is going to behave "the same way" for any input, since the latter will be nothing but a name.

Our approach follows the line of work on nominal techniques [7,19] and nominal operational game semantics [14,12]. We let the set of ***names*** be:

$$\mathsf{N} = \mathsf{TN} \uplus \mathsf{CN}$$

We therefore use two kinds of names: type names $\alpha, \beta \in \mathsf{TN}$; and computation names $c, d \in \mathsf{CN}$. We will range over arbitrary names by a and variants. We extend the syntax of terms and types by including computation and type names as constants, and call the resulting syntax ***namey terms and types***:

$$M, N ::= c \mid x \mid \lambda x^{\theta}.M \mid \Lambda X.M \mid MN \mid M\theta \qquad \theta, \theta' ::= \alpha \mid X \mid \theta \to \theta' \mid \Lambda X.\theta$$

A namey term or type is *closed* if it contains no free (type/term) variables – but it may contain names. On the other hand, a ***value*** is a closed term in head normal form that contains no names. We range over values with v and variants.

We will use the notation $\hat{M}, \hat{N}$, and variants, to refer jointly to namey terms and namey types. Namey terms are typed with additional typing hypotheses for the added constants. These typings are made explicit in the trace model. By abuse of terminology, we will drop the adjective "namey" and refer to the above simply as "terms" and "types". Formally speaking, namey terms and types form *nominal sets* (cf. Definition 8).

Note 6 (what do c's and α's represent?). A computation name c represents a term that can replace the open variables of a term M. That is, in order to examine the semantics of $\lambda x^{\theta}.M$, we will look instead at $M\{c/x\}$ where c a computation name of appropriate type. Type names α have a similar purpose, for types.

Our trace semantics is built on top of head reduction, which is reminded next. Moreover, we shall be using types in *extended form*, which determines the number and types of arguments needed in order to fully apply a term of a given type.

Definition 7. The (standard) head reduction rules are given in Figure 2. Head normal forms are given by the syntax on the LHS below,

$$M_{\mathsf{hnf}} ::= E[x] \mid E[c] \mid \lambda x^{\theta}.M_{\mathsf{hnf}} \mid \Lambda X.M_{\mathsf{hnf}} \qquad E ::= \bullet \mid EM \mid E\theta$$

where E ranges over ***evaluation contexts*** (defined on the RHS). Evaluation contexts are typed with types of the form $\theta \rightsquigarrow \theta'$. We write $E : \theta \rightsquigarrow \theta'$ if we can derive $\bullet : \theta \vdash E : \theta'$.

An ***extended type form*** is a sequence $(\tau_1, ..., \tau_n, \xi)$ with $\xi \in \mathsf{TVar} \cup \mathsf{TN}$ and, for each i, $\tau_i \in \mathsf{Type} \cup \{\forall X \mid X \in \mathsf{TVar}\}$. Formally, the extended form of a type θ, written $\mathrm{ext}(\theta)$, is defined by:

$$\mathrm{ext}(\forall X.\theta) = (\forall X) :: \mathrm{ext}(\theta) \qquad \mathrm{ext}(\theta \to \theta') = \theta :: \mathrm{ext}(\theta') \qquad \mathrm{ext}(\xi) = (\xi)$$

$$(\lambda x.M)N \to M\{N/x\} \qquad \frac{M \to M'}{\lambda x.M \to \lambda x.M'} \qquad \frac{M \to M'}{\Lambda X.M \to \Lambda X.M'} \qquad \frac{M \to M' \quad (*)}{E[M] \to E[M']}$$
$$(\Lambda X.M)\theta \to M\{\theta/X\}$$

Fig. 2. Head reduction rules. Condition $(*)$ stipulates that M be not a Λ/λ-abstraction.

where we write $h :: t$ for the sequence with head h and tail t (cf. list notation). Elements of the form $\forall X$ in these sequences are binders that bind to their right.

We let $\to^*$ be the reflexive-transitive closure of $\to$. It is a standard result that $\to^*$ preserves typing and (strongly) normalises to head normal forms.

We finally introduce some infrastructure for working with objects with names.

Definition 8. We call a permutation $\pi : \mathsf{N} \to \mathsf{N}$ *finite* if the set $\{a \mid \pi(a) \neq a\}$ is finite, and *component-preserving* if, for all $a \in \mathsf{N}$, $a \in \mathsf{TN}$ iff $\pi(a) \in \mathsf{TN}$.

A ***nominal set*** [7] is a pair $(Z, *)$ of a set Z along with an action $(*)$ from the set of finite component-preserving computations of N on the set Z. For each $z \in Z$, the set of names featuring in z form its ***support***, written $\nu(z)$, which we stipulate to be finite.

In the sequel, when constructing objects with names (such as moves or traces) we shall implicitly assume that these form nominal sets, where the permutation action is defined by taking $\pi * z$ to be the result of applying π to each name in z.

3.1 Trace semantics preview

Before formally presenting the trace model, we look at some examples informally, postponing the full details for the next section. Head-reduction brings terms into head normal form. The trace semantics allows us to further 'reduce' terms of the form $E[c\hat{M}_1 \cdots \hat{M}_n]$, where c is some computation name. For such a term, following the game semantics approach [3,11], our model will issue a *move* interrogating the computation c on arguments $\hat{M}_i$, and putting E on top of an *evaluation stack*, denoted $\mathcal{E}$. The move is effectively a call to c, and $\mathcal{E}$ functions as a call stack which registers the calls that have been made and are still pending. This will effectively lead to a labelled transition system in which labels are moves issued by two parties: a *Player (P)*, representing the modelled term, and an *Opponent (O)* representing its enclosing term context.

Traces are sequences of ***moves***, which in turn are tuples of names belonging to one of these four classes, taking $c \in \mathsf{CN}$ and $a_i \in \mathsf{N}$ for each i:

- Player questions $\overline{c}(a_1, ..., a_n)$ (also *P-questions*),
- Opponent questions $c(a_1, ..., a_n)$ (also *O-questions*),
- PO-answers $\overline{\mathsf{OK}}\mathsf{OK}$, and OP-answers $\mathsf{OK}\overline{\mathsf{OK}}$.

Given a question move as above, we let its *core name* be c. We distinguish a computation name $c_{\text{in}} \in \mathsf{CN}$, and call questions with core name c_{in} ***initial***. We define a ***trace*** T to be a finite sequence of moves. Traces will be restricted to *legal* ones in Definition 12

In the following examples we give traces produced by simple System F terms. Traces are formally produced by an LTS over configurations whose main component is an

evaluation stack. An ***evaluation stack*** is a stack whose elements are typed evaluation contexts, apart from the top element which can also be a typed term:

$$\mathcal{E} ::= \mathcal{E}' \mid (M, \theta) :: \mathcal{E}' \qquad \mathcal{E}' ::= \Diamond \mid (E, \theta \rightsquigarrow \theta') :: \mathcal{E}'$$

We denote the empty stack with $\Diamond$. In the next two examples, for simplicity, configurations shall only contain evaluation stacks.

Example 9. Recall that $\mathbf{id} = \Lambda X.\lambda x^X.\, x : \mathbf{Unit}$ and $\mathbf{Unit} = \forall X.X \to X$. The extended type of $\mathbf{Unit}$, $\mathrm{ext}(\mathbf{Unit}) = (\forall X, X, X)$, indicates that $\mathbf{id}$ requires two arguments in order to be evaluated: one type and one term of that given type. Thus, the traces produced by $\mathbf{id}$ will start with an interrogating/calling move $c_{\mathrm{in}}(\alpha, c)$ of O:

- c_{in} is the computation name assigned (by convention) to the term being evaluated (in this case, $\mathbf{id}$);
- α, c are names abstracting the actual type and term arguments which $\mathbf{id}$ is called on. It is assumed that c is of type α.

Starting from the initial move $c_{\mathrm{in}}(\alpha, c)$, a trace of $\mathbf{id}$ can be produced as follows:

$$\langle \Diamond \rangle \xrightarrow{c_{\mathrm{in}}(\alpha,c)} \langle (\mathbf{id}\, \alpha\, c, \alpha) \rangle \to \langle (c, \alpha) \rangle \xrightarrow{\bar{c}()} \langle (\bullet, \alpha \rightsquigarrow \alpha) \rangle \xrightarrow{\mathsf{OK}\overline{\mathsf{OK}}} \langle \Diamond \rangle$$

Thus, O starts the interaction by interrogating $\mathbf{id}$ with α, c. This results in $\mathbf{id}\, \alpha\, c$, which gets head reduced to c. At this point, c is a head normal form of type α, and P can answer the initial question $c_{\mathrm{in}}(\alpha, c)$. This is done in two steps. First, P further reduces c by playing a move $\bar{c}()$ (here c takes 0 arguments as $\mathrm{ext}(\alpha) = (\alpha)$), and pushes the current evaluation context $(\bullet, \alpha \rightsquigarrow \alpha)$ on the stack. O then responds by triggering a pair of answers $\mathsf{OK}\overline{\mathsf{OK}}$, which answer both questions played so far. The resulting trace is: $c_{\mathrm{in}}(\alpha, c) \cdot \bar{c}() \cdot \mathsf{OK}\overline{\mathsf{OK}}$.

Note 10 (what are $\mathsf{OK}\overline{\mathsf{OK}}$ and $\overline{\mathsf{OK}}\mathsf{OK}$?). As System F base types are type variables, there is no real need for answer moves: a type X has no return values. For example, in the game models of Hughes [9] and Laird [15], answer moves were effectively suppressed (either explicitly, or by allowing moves $c(\cdots)$ to function as answers). Here, to give the semantics an operational flavour, we introduce instead explicit 'dummy' answers OK.

Example 11. Consider now $M = \lambda f^{\mathbf{Unit}}.\, f : \mathbf{Unit} \to \mathbf{Unit}$. We have that $\mathrm{ext}(\mathbf{Unit} \to \mathbf{Unit}) = (\mathbf{Unit}, \forall X, X, X)$, and therefore M requires three arguments for its evaluation: one term of type $\mathbf{Unit}$, one type, and one term if that latter type. We can therefore start a trace of M with an initial move $c_{\mathrm{in}}(c_1, \alpha_1, c)$ and continue as follows.

$$\langle \Diamond \rangle \xrightarrow{c_{\mathrm{in}}(c_1,\alpha_1,c_2)} \langle (M c_1 \alpha_1\, c_2, \alpha_1) \rangle \to \langle (c_1\, \alpha_1\, c_2, \alpha_1) \rangle \xrightarrow{\bar{c}_1(\alpha_2,c_3)} \langle (\bullet, \alpha_2 \rightsquigarrow \alpha_1) \rangle$$

Thus, the initial move leads to $M c_1 \alpha_1 c_2$, which in turn reaches the hnf $c_1 \alpha_1 c_2$, with $c_1 : \mathbf{Unit}$, and at that point P needs to invoke c_1 with arguments α_1 and c_2. These are abstracted away by fresh names α_2 and c_3 respectively, which are passed as arguments to c_1. c_3 in particular has type α_2. The result of this invocation will be of type α_2, which is the hole type in $(\bullet : \alpha_2 \rightsquigarrow \alpha_1)$. O can only produce a term of α_2 by simply returning c_3. Similarly to before, this is done in two steps: by O playing $c_3()$, which brings c_2 (the

term represented by c_3) at the top of the stack, which in turn triggers a pair of answers $\overline{\mathsf{OK}}\mathsf{OK}$ and brings c_2 inside the context $(\bullet : \alpha_2 \rightsquigarrow \alpha_1)$.

$$\langle (\bullet, \alpha_2 \rightsquigarrow \alpha_1) \rangle \xrightarrow{c_3()} \langle (c_2, \alpha_2) :: (\bullet, \alpha_2 \rightsquigarrow \alpha_1) \rangle \xrightarrow{\overline{\mathsf{OK}}\mathsf{OK}} \langle (c_2, \alpha_1) \rangle \xrightarrow{\bar{c}_2()} \langle (\bullet, \alpha_1 \rightsquigarrow \alpha_1) \rangle \xrightarrow{\mathsf{OK}\overline{\mathsf{OK}}} \langle \Diamond \rangle$$

The latter step leaves us with (c_2, α_1), which reaches $\Diamond$ as in the previous example.

3.2 Definition of the LTS

We now proceed with the formal definition of the trace semantics. We start off with a series of definitions setting the conditions for a trace to be legal.

The names appearing in a trace are owned by whoever introduces them. A move m ***introduces*** a name a in a trace T if m is a question $q(\vec{a})$ with $a_i = a$ for some i. For each $A \in \{O, P\}$, we let the set of names of T that *are owned by* A be:

$$A(T) = \{a \in \mathsf{N} \mid \exists m.\ m \text{ is an } A\text{-question in } T \wedge m \text{ introduces } a\}.$$

We will be referring to the names appearing in $A(T)$ as A-*names*.

Each move in a trace needs to be justified, i.e. depend on an earlier move (unless the move is initial). Justification is defined in different ways for questions and answers. Given a trace T and two moves m, m' in T, we say that m' ***justifies*** m when m' is before m in T and:

- m is a question with core name c and m' introduces c, or
- m is an answer which answers m' (and m' is a question).

Answering of questions is defined as follows. Each answer (occurrence) m answers the pair of question moves (m_1, m_2) containing the last two question moves in T which are before m and have not been answered yet.

We can now define legality conditions for traces. Below, for $A \in \{O, P\}$, we say that a move is A-*starting* if it is an A-question or an $AA^\perp$-answer (where $O^\perp = P$ and $P^\perp = O$). Similarly, a move is A-*ending* if it is either an A-question or an $A^\perp A$-answer.

Definition 12. A trace T is said to be ***legal*** when, for each $A \in \{O, P\}$:

1. A-ending moves can only be followed by $A^\perp$-starting moves;
2. all moves in T are justified, apart from the first move which must be initial;
3. apart from c_{in}, every name of T is introduced exactly once in it;
4. for each A-question with core name $c \neq c_{\text{in}}$, we have $c \in A^\perp(T)$;
5. if an $AA^\perp$-answer answers (m, m') then these are A- and $A^\perp$-questions respectively.

The conditions above can be given names (suggesting their purpose) as follows: 1. *alternation*, 2. *justification*, 3. *well-introduction*, 4. *well-calling*, 5. *well-answering*.

Each trace T has a complement, which we denote $T^\perp$ and is obtained from T by switching O/P in all of its moves (i.e. each $c(\vec{a})$ becomes $\bar{c}(\vec{a})$, $\mathsf{OK}\overline{\mathsf{OK}}$ becomes $\overline{\mathsf{OK}}\mathsf{OK}$, etc). T is legal iff $T^\perp$ is.

Traces are produced by use of a labelled transition system. The LTS comprises moves as labels, and of *configurations* as nodes. Each configuration contains an evaluation stack of terms and environments that need to be evaluated, as well as mappings containing type/term information on names that have appeared so far. We introduced evaluation stacks in the previous section. Here we shall restrict the allowed shapes thereof as follows.

We let ***passive*** and ***active*** evaluation stacks be defined by the following two grammars respectively, and take evaluation stacks to be $\mathcal{E} ::= \mathcal{E}_{\mathsf{pass}} \mid \mathcal{E}_{\mathsf{actv}}$,

$$\mathcal{E}_{\mathsf{pass}} ::= \Diamond \mid [(E, \alpha \rightsquigarrow \theta)] \mid (E, \alpha \rightsquigarrow \alpha') :: \mathcal{E}_{\mathsf{pass}}\,, \quad \mathcal{E}_{\mathsf{actv}} ::= [(M, \theta)] \mid (M, \alpha) :: \mathcal{E}_{\mathsf{pass}}\,,$$

where θ ranges over closed types with $\nu(\theta) = \varnothing$, and $\Diamond$ is the empty stack.

The other two components of configurations will be maps γ and ϕ of the shape:

$$\gamma \in (\mathsf{CN} \rightharpoonup (\mathsf{Term} \times \mathsf{Type})) \otimes (\mathsf{TN} \rightharpoonup (\mathsf{Type} \times \{\mathcal{U}\})), \quad \phi \in (\mathsf{CN} \rightharpoonup \mathsf{Type}) \otimes (\mathsf{TN} \rightharpoonup \{\mathcal{U}\}),$$

with $F \otimes G = \{f \cup g \mid f \in F \wedge g \in G\}$. $\mathcal{U}$ is a special "universe" symbol that represents the type of types – it is only used for convenience. Then, in words:

- γ assigns term-type pairs to computation names, and type-$\mathcal{U}$ pairs to type names,
- ϕ assigns types to computation names, and $\mathcal{U}$ to type names.

The role of a map γ is to abstract away terms to computational names, and types to type names. On the other hand, a map ϕ simply types names. In the LTS, when P wants to interrogate an O-computation name c with some arguments, they will abstract away the actual arguments to names, record the abstraction in γ, and call c on these names. On the other hand, when O interrogates a P-computation name c with some move $c(\vec{a})$, we will record in ϕ the types of the (new!) O-names $\vec{a}$.

The abstraction of arguments to names is instrumented by a dedicated operation AVal. This operation assigns to each sequence $((\hat{M}_1, \tau_1), ..., (\hat{M}_n, \tau_n), \xi)$, where $(\tau_1, ..., \tau_n, \xi)$ is an extended type (i.e. the type of the computation name we want to call) and each $\hat{M}_i$ is a closed term or type (the i-th argument), a set of triples of the form $(\vec{a}, \gamma, \beta)$ where:

- $\vec{a}$ is a sequence $(a_1, ..., a_n)$ of names (abstracting each of the arguments $\hat{M}_i$),
- γ is a map as above, with domain $\{a_1, ..., a_n\}$,
- β is the result type one gets after applying each a_i for each τ_i.

The operator is formally defined next. In the same definition we introduce the semantics of types, $[\![\theta]\!]$, as sets of triples of the form $(\vec{a}, \phi, \beta)$, which represent all possible input-output name tuples $(\vec{a}, \beta)$ that are allowed for θ, including their typing ϕ.

Definition 13. Given a closed type θ (which may contain type names), we let its semantics be $[\![\theta]\!] = [\![\mathrm{ext}(\theta)]\!]$, where the latter is defined inductively by:

$$\begin{aligned}
[\![(\alpha)]\!] &= \{(\varepsilon, \varepsilon, \alpha)\} \\
[\![\theta :: L]\!] &= \{((c, \vec{a}), \phi \cdot [c \mapsto \theta], \alpha) \mid c \in \mathsf{CN}, (\vec{a}, \phi, \alpha) \in [\![L]\!]\} \\
[\![\forall X :: L]\!] &= \{((\beta, \vec{a}), \phi \cdot [\beta \mapsto \mathcal{U}], \alpha) \mid \beta \in \mathsf{TN}, (\vec{a}, \phi, \alpha) \in [\![L\{\alpha/X\}]\!]\}
\end{aligned}$$

On the other hand, to each sequence $((\hat{M}_1, \tau_1), ..., (\hat{M}_n, \tau_n), \xi)$ we assign a set of *abstract values* $\mathsf{AVal}(((\hat{M}_1, \tau_1), ..., (\hat{M}_n, \tau_n), \xi))$ inductively by:

$$\begin{aligned}
\mathsf{AVal}((\alpha)) &= \{(\varepsilon, \varepsilon, \alpha)\} \\
\mathsf{AVal}((M, \theta) :: L) &= \{((c, \vec{a}), \gamma \cdot [c \mapsto (M, \theta)], \alpha) \mid c \in \mathsf{CN}, (\vec{a}, \gamma, \alpha) \in \mathsf{AVal}(L)\} \\
\mathsf{AVal}((\theta, \forall X) :: L) &= \{((\beta, \vec{a}), \gamma \cdot [\beta \mapsto (\theta, \mathcal{U})], \alpha) \mid \beta \in \mathsf{TN}, (\vec{a}, \gamma, \alpha) \in \mathsf{AVal}(L\{\beta/X\})\}
\end{aligned}$$

(INT) $\langle (M,\theta) :: \mathcal{E},\gamma,\phi\rangle \to \langle (M',\theta) :: \mathcal{E},\gamma,\phi\rangle$ when $M \to^* M'$ with M' a head normal form.

(OQ_0) $\langle \Diamond,\gamma,\phi\rangle \xrightarrow{c(a_1,\ldots,a_n)} \langle [(Ma_1\cdots a_n,\alpha)],\gamma,\phi\cdot\phi'\rangle$

with $\gamma(c) = (M,\theta)$, $((a_1,\ldots,a_n),\phi',\alpha) \in [\![\theta]\!]$ and $\alpha \in \mathrm{dom}(\phi\cdot\phi')$.

(OQ) $\langle (E,\alpha \rightsquigarrow \theta') :: \mathcal{E},\gamma,\phi\rangle \xrightarrow{c(a_1,\ldots,a_n)} \langle (Ma_1\cdots a_n,\alpha') :: (E,\alpha \rightsquigarrow \theta') :: \mathcal{E},\gamma,\phi\cdot\phi'\rangle$

with $\alpha \in \mathrm{dom}(\gamma)$, $\gamma(c) = (M,\theta)$, $((a_1,\ldots,a_n),\phi',\alpha') \in [\![\theta]\!]$ and $\alpha' \in \mathrm{dom}(\phi\cdot\phi') \cup \{\alpha\}$.

(PQ_0) $\langle [(E[c\hat{M}_1\cdots\hat{M}_n],\theta)],\gamma,\phi\rangle \xrightarrow{\bar{c}(a_1,\ldots,a_n)} \langle [(E,\alpha \rightsquigarrow \theta)],\gamma\cdot\gamma',\phi\rangle$
when θ is a closed with empty support, $\mathrm{ext}(\phi(c)) = (\tau_1,\ldots,\tau_n,\xi)$
and $((a_1,\ldots,a_n),\gamma',\alpha) \in \mathsf{AVal}((\hat{M}_1,\tau_1),\ldots,(\hat{M}_n,\tau_n),\xi)$.

(PQ) $\langle (E[c\hat{M}_1\cdots\hat{M}_n],\alpha') :: \mathcal{E},\gamma,\phi\rangle \xrightarrow{\bar{c}(a_1,\ldots,a_n)} \langle (E,\alpha \rightsquigarrow \alpha') :: \mathcal{E},\gamma\cdot\gamma',\phi\rangle$ when $\alpha' \in \mathrm{dom}(\phi)$,
$\mathrm{ext}(\phi(c)) = (\tau_1,\ldots,\tau_n,\xi)$ and $((a_1,\ldots,a_n),\gamma',\alpha) \in \mathsf{AVal}((\hat{M}_1,\tau_1),\ldots,(\hat{M}_n,\tau_n),\xi)$.

(OA) $\langle (\bullet,\alpha \rightsquigarrow \alpha) :: \mathcal{E},\gamma,\phi\rangle \xrightarrow{\mathsf{OK}\overline{\mathsf{OK}}} \langle \mathcal{E},\gamma,\phi\rangle$ when $\alpha \in \mathrm{dom}(\phi)$.

(PA) $\langle (M,\alpha) :: (E,\alpha \rightsquigarrow \theta) :: \mathcal{E},\gamma,\phi\rangle \xrightarrow{\overline{\mathsf{OK}}\mathsf{OK}} \langle (E[M],\theta) :: \mathcal{E},\gamma,\phi\rangle$ when $\alpha \in \mathrm{dom}(\gamma)$ and M a hnf.

Fig. 3. Reduction rules for the LTS.

Both ϕ and γ are finite partial functions whose domains are sets of names. For such maps, the extension notation we used e.g. in $\phi\cdot[c \mapsto z]$ (for appropriate z) means *fresh* extension: $\phi\cdot[c \mapsto z] = \phi \cup \{(c,z)\}$ and given that $c \notin \mathrm{dom}(\phi)$. This notation is extended to whole maps: e.g. $\phi\cdot\phi' = \phi \cup \phi'$ and given that $\mathrm{dom}(\phi) \cap \mathrm{dom}(\phi') = \varnothing$. Moreover, for each map γ we write $\mathsf{fst}(\gamma)$ for its first projection: $\mathsf{fst}(\gamma) = \{(a,\hat{M}) \mid \gamma(a) = (\hat{M},\_)\}$. Similarly, second projection is given by: $\mathsf{snd}(\gamma) = \{(a,Z) \mid \gamma(a) = (\_,Z)\}$.

Definition 14. A ***configuration*** is a triple $\langle \mathcal{E},\gamma,\phi\rangle$ where $\mathcal{E}$ is an evaluation stack and γ and ϕ are as above. The reduction rules of the LTS are given in Figure 3. We write $\mathsf{Tr}(C)$ for the set of traces generated by a configuration C.

Given a typed term $\Delta;\Gamma \vdash M : \theta$, with $\Delta = \{X_1,\ldots,X_n\}$, $\Gamma = \{x_1 : \theta_1,\ldots,x_m : \theta_m\}$, we set $\langle \Delta;\Gamma \vdash M : \theta\rangle = \langle \Diamond,[c_{\mathrm{in}} \mapsto (\widetilde{M},\widetilde{\theta})],\varepsilon\rangle$ and

$$[\![\Delta;\Gamma \vdash M : \theta]\!] = \{T \in \mathsf{Tr}(\langle \Delta;\Gamma \vdash M : \theta\rangle) \mid T \text{ has at most one initial move }\}$$

where $\widetilde{\theta} = \forall X_1.\ldots\forall X_n.\theta_1 \to \cdots \to \theta_m \to \theta$ and $\widetilde{M} = \Lambda X_1.\ldots\Lambda X_n.\lambda x_1^{\theta_1}.\ldots\lambda x_m^{\theta_m}.M$.

A configuration is active (resp. passive) if its evaluation stack is so. An active configuration stands for a term being computed and it may only produce P-moves. A passive configuration, on the other hand, stands for a scenario where O is next to play. Moreover, the map ϕ in a configuration contains information on the O-names that have been played, i.e. $\mathrm{dom}(\phi)$ contains O-names, while $\mathrm{dom}(\gamma)$ contains P-names.

To better grasp Figure 3 let us consider an initial configuration $\langle \Diamond,[c_{\mathrm{in}} \mapsto (M,\theta)],\varepsilon\rangle$ and look at its traces, for some closed term M (so no need for $\widetilde{M},\widetilde{\theta}$) with empty support.

– At the beginning, the only rule that can be applied is (OQ_0), whereby O interrogates the term M by issuing a move $c_{\mathrm{in}}(\vec{a})$. The names $\vec{a}$ are selected from $[\![\theta]\!]$ and represent arguments that O fully applies the term M on. Since θ has empty support, its extended form is of the shape $(\tau_1,...,\tau_n,X)$ with X bound by one of the τ_i's. Consequently, when the names $a_1,...,a_n$ are applied for $\tau_1,...,\tau_n$, the variable X will be replaced by

some type name α. The rule makes this explicit, by requiring that $(\vec{a}, \phi', \alpha) \in [\![\theta]\!]$. Thus, writing ϕ_0 instead of ϕ' and setting $\gamma_0 = [c_{\text{in}} \mapsto (M, \theta)]$, the transition brings us to a configuration $\langle [(M\vec{a}, \alpha)], \gamma_0, \phi_0 \rangle$, where $\text{dom}(\phi_0) = \{a_1, ..., a_n\}$.

– At this point, the term $M\vec{a}$ can be reduced using head reduction and brought to head normal form. Applying the (INT) rule we reach some $\langle [(E[c\hat{M}_1 \cdots \hat{M}_k], \alpha)], \gamma_0, \phi_0 \rangle$.

– We next interrogate the computation name c. The latter must have come from the $a_1, ..., a_n$ that were applied to M, hence is an O-name. To interrogate it, P plays a question $\bar{c}(\vec{a}')$, using the (PQ) rule and assuming $(\vec{a}', \gamma', \alpha') \in \mathsf{AVal}(((\hat{M}_1, \tau_1'), ..., (\hat{M}_k, \tau_k'), \xi))$, $\phi_0(c) = \theta'$, $\text{ext}(\theta') = (\tau_1', ..., \tau_k', \xi)$. This leads to $\langle [(E, \alpha' \rightsquigarrow \alpha)], \gamma_1, \phi_0 \rangle$ ($\gamma_1 = \gamma_0 \cdot \gamma'$).

– We are now at a passive configuration, where E has been stored on the stack and O is required to produce a response of type α'. By definition of AVal, either $\alpha' = \alpha$ or α' is in $a_1', ..., a_k'$ and hence belongs to P. In the latter case, O can only produce such a response by calling back P, using rule (OQ), playing an O-question and adding a new term on the evaluation stack. In the former case, O would directly respond with a hnf of type α, say N. But, since $E : \alpha \rightsquigarrow \alpha$ and therefore $E = \bullet$, P would simply reply back playing N again. To avoid this copycat of hnf's, we simply play an OP-answer and remove the top of the evaluation stack – this is what the (OA) rule achieves.

Example 15. In Figure 4 we include example traces for terms $M_1, M_2 : \mathbf{Unit} \to \mathbf{Unit}$ (taken from [1], Instance 3.25) and for the Church numerals $M_k : \mathbf{Nat}$. The former pair is an instance of Theorem 21 – Strachey equivalence implies trace equivalence.

In our scenario above we started from a passive configuration with empty stack and a singleton γ. A different way to produce a trace is to start from an active configuration with a stack containing only a term $E[c_{\text{in}}\hat{M}_1 \cdots \hat{M}_n]$, in which case the rule (PQ_0) would commence the trace. More generally, we call a configuration C with stack $\mathcal{E}$:

– a ***term configuration***, if $\mathcal{E} = \Diamond$ or the bottom element of $\mathcal{E}$ has type α or $\alpha \rightsquigarrow \alpha'$;
– a ***context configuration***, if the bottom of $\mathcal{E}$ has type θ or $\alpha \rightsquigarrow \theta$, and θ is a closed with empty support.

Each reduction sequence in the LTS can only contain either term or context configurations. In our discussion above and in Example 15 we examine the semantics of terms, and therefore use term configurations. In later sections, when we shall start looking at the semantics of contexts, we will be using context configurations as well.

While we have not defined leaves for our LTS, there is a natural notion of a trace being "completed". In particular, we call a trace T ***complete*** if all its questions have been answered. We write $\mathsf{CTr}(C)$ for the set of complete traces generated from C. Term and context configurations can both produce complete traces. Given a term configuration C and a complete trace T, we write $C \Downarrow_T$ if $C \xrightarrow{T} C'$ and C' has an empty evaluation stack. On the other hand, given a context configuration C, a complete trace T and a value v, we write $C \Downarrow_{T,v}$ if $C \xrightarrow{T} C'$ and C' has an evaluation stack with a single element (v, θ).

Lemma 16. *Given a term configuration C and $T \in \mathsf{Tr}(C)$, then T is complete iff $C \Downarrow_T$.*

We conclude this section by looking at some restrictions characterising actual configurations. We first extend fst to evaluation stacks by: $\mathsf{fst}(\Diamond) = \Diamond$ and $\mathsf{fst}((Z, \_) :: \mathcal{E}) = Z :: \mathsf{fst}(\mathcal{E})$.

$M_1 = \lambda f^{\mathbf{Unit}}.\, f\,\mathbf{Unit}\, f : \mathbf{Unit}\to\mathbf{Unit}$, $M_2 = \lambda f^{\mathbf{Unit}}.\Lambda X.\, f(X\to X)(fX) : \mathbf{Unit}\to\mathbf{Unit}$, and $\mathrm{ext}(\mathbf{Unit}\to\mathbf{Unit}) = (\mathbf{Unit}, \forall X, X, X)$. Traces for M_1 (left) and M_2 (right):

$\langle \Diamond, \gamma_0, \varepsilon, \varepsilon\rangle \qquad (\gamma_0 = [c_{\mathrm{in}} \mapsto (M_1, \theta)])$
$\xrightarrow{c_{\mathrm{in}}(c_1,\alpha_1,c_2)} \langle (M_1\, c_1\, \alpha_1\, c_2, \alpha_1), \gamma_0, \phi_0\rangle$
$\to \langle (c_1\,\mathbf{Unit}\, c_1\, \alpha_1\, c_2, \alpha_1), \gamma_0, \phi_0\rangle$
$\xrightarrow{\bar{c}_1(\alpha_2,c_3)} \langle (\bullet\, \alpha_1\, c_2, \alpha_2 \rightsquigarrow \alpha_1), \gamma_1, \phi_0\rangle$
$\xrightarrow{c_3()} \langle (c_1, \alpha_2) :: (\bullet\, \alpha_1\, c_2, \alpha_2 \rightsquigarrow \alpha_1), \gamma_1, \phi_0\rangle$
$\xrightarrow{\overline{\mathsf{OK}}\mathsf{OK}} \langle (c_1\, \alpha_1\, c_2, \alpha_1), \gamma_1, \phi_0\rangle$
$\xrightarrow{\bar{c}_1(\alpha_2',c_3')} \langle (\bullet, \alpha_2' \rightsquigarrow \alpha_1), \gamma_2, \phi_0\rangle$
$\xrightarrow{c_3'()} \langle (c_2, \alpha_2') :: (\bullet, \alpha_2' \rightsquigarrow \alpha_1), \gamma_2, \phi_0\rangle$
$\xrightarrow{\overline{\mathsf{OK}}\mathsf{OK}} \langle (c_2, \alpha_1), \gamma_2, \phi_0\rangle$
$\xrightarrow{\bar{c}_2()} \langle (\bullet, \alpha_1 \rightsquigarrow \alpha_1), \gamma_1, \phi_0\rangle \xrightarrow{\mathsf{OK}\overline{\mathsf{OK}}} \langle \Diamond, \gamma_1, \phi_0\rangle$

$\langle \Diamond, \gamma_0', \varepsilon, \varepsilon\rangle \qquad (\gamma_0' = [c_{\mathrm{in}} \mapsto (M_2, \theta)])$
$\xrightarrow{c_{\mathrm{in}}(c_1,\alpha_1,c_2)} \langle (M_2\, c_1\, \alpha_1\, c_2, \alpha_1), \gamma_0', \phi_0\rangle$
$\to \langle (c_1(\alpha_1\to\alpha_1)(c_1\alpha_1)c_2, \alpha_1), \gamma_0', \phi_0\rangle$
$\xrightarrow{\bar{c}_1(\alpha_2,c_3)} \langle (\bullet\, c_2, \alpha_2 \rightsquigarrow \alpha_1), \gamma_1', \phi_0\rangle$
$\xrightarrow{c_3()} \langle (c_1\alpha_1, \alpha_2) :: (\bullet\, c_2, \alpha_2 \rightsquigarrow \alpha_1), \gamma_1', \phi_0\rangle$
$\xrightarrow{\overline{\mathsf{OK}}\mathsf{OK}} \langle (c_1\, \alpha_1\, c_2, \alpha_1), \gamma_1', \phi_0\rangle \to \ldots$

where:

$\phi_0 = \{c_1 \mapsto \mathbf{Unit}, \alpha_1 \mapsto \mathcal{U}, c_2 \mapsto \alpha_1\}$
$\gamma_1 = \gamma_0 \cdot [\alpha_2 \mapsto (\mathbf{Unit}, \mathcal{U}), c_3 \mapsto (c_1, \mathbf{Unit})]$
$\gamma_2 = \gamma_1 \cdot [\alpha_2' \mapsto (\alpha_1, \mathcal{U}), c_3' \mapsto (c_2, \alpha_2')]$
$\gamma_1' = \gamma_0' \cdot [\alpha_2 \mapsto (\alpha_1\to\alpha_1, \mathcal{U}), c_3 \mapsto (c_1\alpha_1, \alpha_1\to\alpha_1)]$

$M_k = \Lambda X.\lambda f^{X\to X}.\, \lambda x^X.\, N_{f,x,k} \qquad N_{M_f,M_x,k} = \underbrace{M_f(M_f(\ldots(M_f}_{k}\ M_x))\ldots)$
$\mathrm{ext}(\mathbf{Nat}) = (\forall X, X\to X, X, X)$

Set $\gamma_0 = [c_{\mathrm{in}} \mapsto (M_k, \mathbf{Nat})]$. Reduction for M_k:

$\langle \Diamond, \gamma_0, \varepsilon, \varepsilon\rangle \xrightarrow{c_{\mathrm{in}}(\alpha_1,c_f,c_x)} \langle (M_k\, \alpha_1\, c_f\, c_x, \alpha_1), \gamma_0, \phi_0\rangle \to \langle (c_f(N_{c_f,c_x,k-1}), \alpha_1), \gamma_0, \phi_0\rangle$
$\xrightarrow{\bar{c}_f(c_1)} \langle (\bullet, \alpha_1 \rightsquigarrow \alpha_1), \gamma_1, \phi_0\rangle \xrightarrow{\mathsf{OK}\overline{\mathsf{OK}}} \langle \Diamond, \gamma_1, \phi_0\rangle \xrightarrow{c_1()} \langle (c_f(N_{c_f,c_x,k-2}), \alpha_1), \gamma_1, \phi_0\rangle$
$\xrightarrow{\bar{c}_f(c_2)} \langle (\bullet, \alpha_1 \rightsquigarrow \alpha_1), \gamma_2, \phi_0\rangle \xrightarrow{\mathsf{OK}\overline{\mathsf{OK}}} \langle \Diamond, \gamma_2, \phi_0\rangle \cdots \xrightarrow{c_{k-1}()} \langle (c_x, \alpha_1), \gamma_{k-1}, \phi_0\rangle$
$\xrightarrow{\bar{c}_x()} \langle (\bullet, \alpha_1 \rightsquigarrow \alpha_1), \gamma_{k-1}, \phi_0\rangle \xrightarrow{\mathsf{OK}\overline{\mathsf{OK}}} \langle \Diamond, \gamma_{k-1}, \phi_0\rangle$

where $\phi_0 = \{\alpha_1 \mapsto \mathcal{U}, c_f \mapsto (\alpha_1 \to \alpha_1), c_x \mapsto \alpha_1\}$ and $\gamma_i = \gamma_{i-1} \cdot [c_i \mapsto (N_{c_f,c_x,k-i}, \alpha_1)]$.

Fig. 4. Top: traces for two terms of type $\mathbf{Unit}\to\mathbf{Unit}$. Bottom: traces for Church numeral M_k.

Definition 17. A configuration $\langle \mathcal{E}, \gamma, \phi\rangle$ is said to be ***legal*** when:

- $\mathrm{dom}(\gamma) \cap \mathrm{dom}(\phi) = \varnothing$ and $\nu(\mathsf{fst}(\mathcal{E})) \cup \nu(\mathrm{cod}(\mathsf{fst}(\gamma))) \subseteq \mathrm{dom}(\phi)$;
- for all $c \in \mathrm{dom}(\gamma) \cap \mathsf{CN}$, given $\gamma(c) = (M, \theta)$, we have $\Delta_\phi; \Gamma_{\phi,\gamma} \vdash M : \theta\{\gamma_v\}$;
- if the top of $\mathcal{E}$ is (M, θ), then $\Delta_\phi; \Gamma_{\phi,\gamma} \vdash M : \widetilde{\theta}$ with either $\theta = \alpha \in \mathrm{dom}(\gamma)$ and $\gamma(\alpha) = (\widetilde{\theta}, \mathcal{U})$, or $\theta = \alpha \in \mathrm{dom}(\phi)$ and $\widetilde{\theta} = \theta$, or $\theta = \widetilde{\theta}$ is a closed type with empty support and $\mathcal{E} = [(M, \theta)]$;
- If $\mathcal{E} = (M, \alpha_1) :: (E, \alpha_2 \rightsquigarrow \theta) :: \mathcal{E}'$, either $\alpha_1 = \alpha_2$ or $\alpha_1 \in \mathrm{dom}(\phi)$;
- for all $(E, \alpha \rightsquigarrow \theta)$ in $\mathcal{E}$ with $\alpha \in \mathrm{dom}(\gamma)$, $\Delta_\phi; \Gamma_{\phi,\gamma}, \vdash E : \gamma_v(\alpha) \rightsquigarrow \theta$, and either $\theta = \alpha \in \mathrm{dom}(\phi)$ or θ is a closed type with empty support, and $(E, \alpha \rightsquigarrow \theta)$ is at the bottom of $\mathcal{E}$;

– for all $(E, \alpha \rightsquigarrow \theta)$ in $\mathcal{E}$ with $\alpha \in \mathrm{dom}(\phi)$, we have $\theta = \alpha$ and $E = \bullet$;
where $\Delta_\phi = \mathrm{dom}(\phi) \cap \mathsf{TN}$ and $\Gamma_{\phi,\gamma} = \{(x, \theta\{\mathsf{fst}(\gamma)\}) \mid (x, \theta) \in \phi\}$.

Lemma 18. *If C is a legal configuration and $C \xrightarrow{m} C'$ then C' is a legal configuration.*

4 Parametricity in the Trace Model, and proof of Theorem 5

We next examine the relationship between trace equivalence and the notions of Reynolds and Strachey equivalence. We prove that Strachey equivalence is included in trace equivalence (Theorem 21), which in turn is included in Reynolds equivalence (Theorem 28).

4.1 From Strachey to trace equivalence

Definition 19. Let $C_i = \langle \mathcal{E}_i, \gamma_i, \phi_i \rangle$, for $i = 1, 2$, be two configurations. We say that C_1 and C_2 are ***Strachey-equivalent*** when $\mathcal{E}_1$ and $\mathcal{E}_2$ have the same size, $\mathrm{dom}(\gamma_1) = \mathrm{dom}(\gamma_2)$, $\phi_1 = \phi_2$ and:

– for all $c \in \mathrm{dom}(\gamma_1)$, if $\gamma_i(c) = (M_i, \theta_i)$ then $\theta_1 = \theta_2$ and $\mathbf{erase}(M_1) =_{\beta\eta} \mathbf{erase}(M_2)$;
– if (Z_i, α_i) is the j-th element of $\mathcal{E}_i$, then $\alpha_1 = \alpha_2$ and $\mathbf{erase}(Z_1) =_{\beta\eta} \mathbf{erase}(Z_2)$;

where $E_1 =_{\beta\eta} E_2$ just if $E_1[x] =_{\beta\eta} E_2[x]$ for some/all fresh x.

The first inclusion can then be proven as follows.

Lemma 20. *Given two Strachey-equivalent legal configurations C_1, C_2, if $C_1 \xrightarrow{m} C_1'$ for some m, C_1' then there is $C_2 \xrightarrow{m} C_2'$ such that C_1' and C_2' are Strachey-equivalent.*

Theorem 21. *For all Strachey-equivalent $\Delta, \Gamma \vdash M_1, M_2 : \theta$, we have $[\![M_1]\!] = [\![M_2]\!]$.*

Proof. Taking $T \in [\![\Delta; \Gamma \vdash M_1 : \theta]\!]$, we prove that $T \in [\![\Delta; \Gamma \vdash M_2 : \theta]\!]$ by induction on the length of T, using the previous lemma. □

The inclusion above is strict. This is shown, for example, by the following terms $M_{\mathbf{true}}, M_{\mathbf{false}} : \mathbf{Unit} \to \mathbf{Unit}$, which are trace equivalent but not Strachey-equivalent:

$$M_{\mathbf{b}} = \lambda f^{\mathbf{Unit}}.\Lambda X.\lambda x^X.\mathbf{snd}(f(\mathbf{Bool} \times X)\langle \mathbf{b}, x\rangle) \quad (\mathbf{b} = \mathbf{true}, \mathbf{false})$$

Here we use the impredicative encoding of product types [8]: $\theta_1 \times \theta_2 = \forall X.(\theta_1 \to \theta_2 \to X) \to X$, $\langle M, N\rangle = \Lambda X.\lambda f^{\theta_1 \to \theta_2 \to X}.fMN$ and $\mathbf{snd} = \lambda x^{\theta_1 \times \theta_2}.x\theta_2(\lambda y^{\theta_1}.\lambda z^{\theta_2}.z)$. Setting $\gamma_0 = [c_{\mathrm{in}} \mapsto (M_{\mathbf{b}}, \mathbf{Unit} \to \mathbf{Unit})]$ and $C_{\mathbf{b}} = \langle \cdot; \cdot \vdash M_{\mathbf{b}} : \mathbf{Unit} \to \mathbf{Unit}\rangle$, we have:

$$\begin{aligned} C_{\mathbf{b}} &\xrightarrow{c_{\mathrm{in}}(c_f,\alpha,c)} \langle (\mathbf{snd}(c_f(\mathbf{Bool} \times \alpha)\langle \mathbf{b}, c\rangle), \alpha), \gamma_0, \phi_0\rangle \quad (\phi_0 = [c_f \mapsto \mathbf{Unit}, \alpha \mapsto \mathcal{U}, c \mapsto \alpha]) \\ &\xrightarrow{\bar{c}_f(\beta,c')} \langle (\mathbf{snd}\bullet, \beta \rightsquigarrow \alpha), \gamma_1, \phi_0\rangle \qquad (\gamma_1 = \gamma_0 \cdot [\beta \mapsto (\mathbf{Bool} \times \alpha, \mathcal{U}), c' \mapsto (\langle \mathbf{b}, c\rangle, \beta)]) \\ &\xrightarrow{c'()} \langle (\langle \mathbf{b}, c\rangle, \beta) :: (\mathbf{snd}\bullet, \beta \rightsquigarrow \alpha), \gamma_1, \phi_0\rangle \xrightarrow{\overline{\mathsf{OK}}\mathsf{OK}} \langle (\mathbf{snd}\langle \mathbf{b}, c\rangle, \alpha), \gamma_1, \phi_0\rangle \\ &\longrightarrow \langle (c, \alpha), \gamma_1, \phi_0\rangle \xrightarrow{\bar{c}()} \langle (\bullet, \alpha \rightsquigarrow \alpha), \gamma_1, \phi_0\rangle \xrightarrow{\mathsf{OK}\overline{\mathsf{OK}}} \langle \diamond, \gamma_1, \phi_0\rangle \end{aligned}$$

and this is the only complete trace in $[\![M_{\mathbf{b}}]\!]$. Indeed, O cannot interrogate another name, as c_{in} can only be played once, and c' cannot be played with the (OQ_0) rule.

The other inclusion (trace included in Reynolds) is more challenging and requires us to introduce machinery for relating the semantics of terms and semantics of contexts to that of terms and contexts composed.

(P-INT) $\langle (M,\alpha) :: \mathcal{E}_P, \mathcal{E}_O, \gamma_P, \gamma_O\rangle \longrightarrow \langle (M',\alpha) :: \mathcal{E}_P, \mathcal{E}_O, \gamma_P, \gamma_O\rangle$ when $M \to^* M'$ (hnf).

(O-INT) $\langle \mathcal{E}_P, (M,\alpha) :: \mathcal{E}_O, \gamma_P, \gamma_O\rangle \longrightarrow \langle (M',\alpha) :: \mathcal{E}_P, \mathcal{E}_O, \gamma_P, \gamma_O\rangle$ when $M \to^* M'$ (hnf).

(PA) $\langle (M,\alpha) :: (E, \alpha \rightsquigarrow \alpha') :: \mathcal{E}_P, (\bullet, \alpha \rightsquigarrow \alpha) :: \mathcal{E}_O, \gamma_P, \gamma_O\rangle \xrightarrow{\overline{\mathsf{OK}}\mathsf{OK}} \langle (E[M],\alpha') :: \mathcal{E}_P, \mathcal{E}_O, \gamma_P, \gamma_O\rangle$
with M a hnf and $\alpha \in \mathrm{dom}(\gamma_P)$.

(OA) $\langle (\bullet, \alpha \rightsquigarrow \alpha) :: \mathcal{E}_P, (M,\alpha) :: (E, \alpha \rightsquigarrow \theta) :: \mathcal{E}_O, \gamma_P, \gamma_O\rangle \xrightarrow{\mathsf{OK}\overline{\mathsf{OK}}} \langle \mathcal{E}_P, (E[M],\theta) :: \mathcal{E}_O, \gamma_P, \gamma_O\rangle$
with M a hnf and $\alpha \in \mathrm{dom}(\gamma_O)$.

(PQ) $\langle (E[c\hat{M}_1 \cdots \hat{M}_n], \alpha') :: \mathcal{E}_P, \mathcal{E}_O, \gamma_P, \gamma_O\rangle$
$\xrightarrow{\bar{c}(\vec{a})} \langle (E, \alpha \rightsquigarrow \alpha') :: \mathcal{E}_P, (M\vec{a}, \alpha) :: \mathcal{E}_O, \gamma_P \cdot \gamma', \gamma_O\rangle$ when $\alpha' \in \mathrm{dom}(\gamma_O)$, $\gamma_O(c) = (M,\theta)$, $\mathrm{ext}(\theta) = (\tau_1, \ldots, \tau_n, \xi)$ and $((a_1, \ldots, a_n), \gamma', \alpha) \in \mathsf{AVal}((\hat{M}_1, \tau_1), \ldots, (\hat{M}_2, \tau_n), \xi)$.

(OQ) $\langle \mathcal{E}_P, (E[c\hat{M}_1 \cdots \hat{M}_n], \theta) :: \mathcal{E}_O, \gamma_P, \gamma_O\rangle \xrightarrow{c(\vec{a})} \langle (M\vec{a}, \alpha) :: \mathcal{E}_P, (E, \alpha \rightsquigarrow \theta) :: \mathcal{E}_O, \gamma_P, \gamma_O \cdot \gamma'\rangle$
when $\theta = \alpha' \in \mathrm{dom}(\gamma_P)$ or θ a closed type with empty support, with $\gamma_P(c) = (M,\theta)$, $\mathrm{ext}(\theta) = (\tau_1, \cdots, \tau_n, \xi)$ and $((a_1, \ldots, a_n), \gamma', \alpha) \in \mathsf{AVal}((\hat{M}_1, \tau_1), \cdots, (\hat{M}_2, \tau_n), \xi)$.

Fig. 5. Composite LTS.

4.2 Composite LTS

We let a ***composite configuration*** be a tuple $\langle \mathcal{E}_P, \mathcal{E}_O, \gamma_P, \gamma_O\rangle$, where γ_P and γ_O are maps γ as above, $\mathcal{E}_P$ is a term evaluation stack, and $\mathcal{E}_O$ is a context evaluation stack. These configurations represent the interaction between a term and a context. The term-part in the interaction is played by $\mathcal{E}_P$ and γ_P, while the context-part by $\mathcal{E}_O$ and γ_O. As with ordinary configurations, we define an LTS for composite ones in Figure 5. Given a composite configuration C, a trace T and a value v (hnf with empty support) we write $C \Downarrow_{T,v}$ when $C \xrightarrow{T} \langle \Diamond, [(v,\theta)], \gamma_P, \gamma_O\rangle$.

Composite configurations allow us to compose a term and a context semantically: we essentially play the traces of one against the other. Another way to obtain a composite semantics is to work syntactically, i.e. by composing configurations and then executing the resulting term. This is defined next.

Definition 22. Given two evaluation stacks $(\mathcal{E}_P, \mathcal{E}_O)$, we build their ***merge*** (which may not always be defined) $\mathcal{E}_P || \mathcal{E}_O$ inductively by $\Diamond || [(M,\theta)] = M$ and:

$$((M,\alpha) :: \mathcal{E}_P) || ((E, \alpha \rightsquigarrow \theta) :: \mathcal{E}_O) = \mathcal{E}_P || ((E[M],\theta) :: \mathcal{E}_O)$$
$$((E, \alpha \rightsquigarrow \theta) :: \mathcal{E}_P) || ((M,\alpha) :: \mathcal{E}_O) = ((E[M],\theta) :: \mathcal{E}_P) || \mathcal{E}_O$$

When it is defined, we say that $\mathcal{E}_P, \mathcal{E}_O$ are ***compatible***. Then, a composite configuration $C = \langle \mathcal{E}_P, \mathcal{E}_O, \gamma_P, \gamma_O\rangle$ is ***legal*** when $(\mathcal{E}_P, \mathcal{E}_O)$ are compatible and when both $\langle \mathcal{E}_P, \gamma_P, \mathsf{snd}(\gamma_O)\rangle$ and $\langle \mathcal{E}_O, \gamma_O, \mathsf{snd}(\gamma_P)\rangle$ are legal.

We now relate the reduction of a composite configuration with the head reduction of the merge of its two evaluation stacks. First, taking the two environments γ_P, γ_O of a legal composite configuration, we compute their *closure* $(\gamma_P \cdot \gamma_O)^*$ as follows. Setting $\gamma^0 = \mathsf{fst}(\gamma_P \cdot \gamma_O)$, and $\gamma^i = \{(a, \hat{M}\{\gamma\}) \mid (a,\hat{M}) \in \gamma^{i-1}\}$ $(i > 0)$, there is an integer n such that $\nu(\mathrm{cod}(\gamma^n)) = \varnothing$. We write $(\gamma_P \cdot \gamma_O)^*$ for the environment defined as γ^n, for the least n satisfying this latter condition.

Theorem 23. *Given a legal composite configuration* $C = \langle \mathcal{E}_P, \mathcal{E}_O, \gamma_P, \gamma_O \rangle$*, then* $C \Downarrow_{T,v}$ *iff* $(\mathcal{E}_P \| \mathcal{E}_O)\{(\gamma_P \cdot \gamma_O)^*\} \to^* v$.

Finally, we relate the LTS's for composite configurations and ordinary configurations (Theorem 26). Combined with Theorem 23, this gives us a correlation between the traces of two compatible configurations and the head reduction we obtain once we merge their evaluation stacks.

Definition 24. Given legal configuartions $C_P = \langle \mathcal{E}_P, \gamma_P, \phi_P \rangle$ and $C_O = \langle \mathcal{E}_O, \gamma_O, \phi_O \rangle$, we say that they are *compatible* when $\mathcal{E}_P, \mathcal{E}_O$ are compatible, $\mathsf{snd}(\gamma_P) = \phi_O$ and $\mathsf{snd}(\gamma_O) = \phi_P$. For each pair (C_P, C_O) of compatible configurations, we define their merge $C_P \twoheadrightarrow C_O$ as the composite configuration $\langle \mathcal{E}_P, \mathcal{E}_O, \gamma_P, \gamma_O \rangle$.

Lemma 25. *Taking* (C_P, C_O) *a pair of compatible configurations,* $C_P \twoheadrightarrow C_O \Downarrow_{T,v}$ *iff* $C_P \Downarrow_T$ *and* $C_O \Downarrow_{T^\perp,v}$.

Theorem 26. *Given* $C_{P,1}, C_{P,2}, C_O$ *such that* $C_{P,1}, C_O$ *and* $C_{P,2}, C_O$ *are pairwise compatible and* $\mathsf{Tr}(C_{P,1}) = \mathsf{Tr}(C_{P,2})$*, if* $C_{P,1} \twoheadrightarrow C_O \Downarrow_{T,v}$*, then* $C_{P,2} \twoheadrightarrow C_O \Downarrow_{T,v}$.

Proof. From Lemma 25 we get $C_{P,1} \Downarrow_T$ and $C_O \Downarrow_{T^\perp,v}$. Thus, $T \in \mathsf{Tr}(C_{P,1})$ and hence $T \in \mathsf{Tr}(C_{P,2})$. Lemma 16 then yields $C_{P,2} \Downarrow_T$ and, from Lemma 25, $C_{P,2} \twoheadrightarrow C_O \Downarrow_{T,v}$. □

4.3 Proof of Theorem 5

Theorem 5 follows from Theorems 21 and 28. Theorem 28, which is proved below, shows that any trace equivalent terms are also Reynolds equivalent. This is achieved as follows. In the previous section we saw how to relate reductions of terms-in-context to the semantics of terms and contexts. Given terms M_1, M_2 which are trace equivalent, and fully applying them to related arguments, we obtain head reductions to values. These reductions can be decomposed into LTS reductions producing corresponding traces, for the terms and their argument terms (which form contexts). But, since the terms are trace equivalent, M_2 can simulate the behaviour of M_1 in the context of M_1, and that allows us to show that the two composites reduce to the same value.

We start by extending logical relations to extended types with empty support. We define $\mathcal{R}[\![\mathrm{ext}(\theta)]\!]_\delta$ by:

$$\begin{aligned}
\mathcal{R}[\![(X)]\!]_\delta &= \{R \mid \delta(X) = (_, _, R)\} \\
\mathcal{R}[\![\theta :: L]\!]_\delta &= \{(M_1, N_1) :: L' \mid (M_1, N_1) \in \mathcal{R}[\![\theta]\!]_\delta \wedge L' \in \mathcal{R}[\![L]\!]_\delta\} \\
\mathcal{R}[\![\forall X :: L]\!]_\delta &= \{(\theta_1, \theta_2) :: L' \mid (\theta_1, \theta_2, R) \in \mathrm{Rel} \wedge L' \in \mathcal{R}[\![L]\!]_{\delta \cdot [X \mapsto (\theta_1, \theta_2, R)]}\}
\end{aligned}$$

Lemma 27. $(M_1, M_2) \in \mathcal{R}[\![\theta]\!]_\delta$ *iff for all* $((\hat{N}_1^1, \hat{N}_2^1), \ldots, (\hat{N}_1^n, \hat{N}_2^n), R) \in \mathcal{R}[\![\mathrm{ext}(\theta)]\!]_\delta$, $(M_1 \hat{N}_1^1 \cdots \hat{N}_1^n, M_2 \hat{N}_2^1 \cdots \hat{N}_2^n) \in R$.

Theorem 28. *For all trace equivalent* $\Delta; \Gamma \vdash M_1, M_2 : \theta$*, we have that* $M_1 \simeq_{log} M_2$.

Proof. Taking $\delta \in \mathcal{R}[\![\Delta]\!]$ and $(\eta_1, \eta_2) \in \mathcal{R}[\![\Gamma]\!]_\delta$, we show $(M_1\{\eta_1\}\{\delta_1\}, M_2\{\eta_2\}\{\delta_2\}) \in \mathcal{R}[\![\theta]\!]_\delta$. Using Lemma 27, we take $((\hat{N}_1^1, \hat{N}_2^1), \ldots, (\hat{N}_1^n, \hat{N}_2^n), R) \in \mathcal{R}[\![\mathrm{ext}(\theta)]\!]_\delta$, and prove that $(M_1\{\eta_1\}\{\delta_1\}\hat{N}_1^1 \cdots \hat{N}_1^n, M_2\{\eta_2\}\{\delta_2\}\hat{N}_2^1 \cdots \hat{N}_2^n) \in R$.

For each $i \in \{1,2\}$, there exists a value v_i s.t. $M_i\{\eta_i\}\{\delta_i\}\hat{N}_i^1 \cdots \hat{N}_i^n \to^* v_i$. Using the closure of R w.r.t. $=_{\beta\eta}$, it suffices to show that $(v_1, v_2) \in R$. Suppose $\Delta = X_1, \ldots, X_k$ and $\Gamma = x_1 : \theta_1, \ldots, x_m : \theta_m$. We write C_{P_i} for the configuration $\langle \Delta; \Gamma \vdash M_i : \theta \rangle$, and $C_{O,i}$ for the configuration $\langle c_{\text{in}}\delta_i(X_1)\cdots\delta_i(X_k)\eta_i(x_1)\cdots\eta_i(x_m)\hat{N}_i^1\cdots\hat{N}_i^n, \varepsilon, [c_{\text{in}} \mapsto \widetilde{\theta}]\rangle$, where $\widetilde{\theta} = \forall X_1 \ldots \forall X_n . \theta_1 \to \cdots \to \theta_m \to \theta$.

From Theorem 23, for each $i \in \{1,2\}$ there is a trace T_i such that $C_{P,i} \twoheadrightarrow C_{O,i} \Downarrow_{T_i, v_i}$. M_1, M_2 being trace equivalent, we have that $\mathsf{Tr}(C_{P,1}) = \mathsf{Tr}(C_{P,2})$. So from Theorem 26, we get that $C_{P,2} \twoheadrightarrow C_{O,1} \Downarrow_{T_1, v_1}$, and from Theorem 23 that $M_2\{\eta_1\}\{\delta_1\}\hat{N}_1^1\cdots\hat{N}_1^n \to^* v_1$. Finally, from Theorem 2, we get that $(M_2\{\eta_1\}\{\delta_1\}\hat{N}_1^1\cdots\hat{N}_1^n, M_2\{\eta_2\}\{\delta_2\}\hat{N}_2^1\cdots\hat{N}_2^n) \in R$. Thus, using the closure of R w.r.t. $=_{\beta\eta}$, we have that $(v_1, v_2) \in R$. □

5 Related and Future Work

The literature on parametric polymorphism is vast; here we look at the works closest to ours, which come from the game semantics area. The first game model for System F was introduced by Hughes [10,9]. The model is intentional, in the sense that it is fully complete for $\beta\eta$-equivalence. Starting from that model, de Lataillade [6,5] characterised parametricity categorically via the notion of dinaturality [4]. In [2], Abramsky and Jagadeesan developed a model for System F to characterise genericity, as introduced by Longo, Milstead and Soloviev [17]. A type θ is said to be *generic* when two terms M_1, M_2 of type $\forall X.\theta'$ are equivalent just if $M_1\theta$ and $M_2\theta$ are equivalent. Their model contains several generic types. More recently, Laird [15] has introduced a game model for System F augmented with mutable variables. His model is closer to ours than the previous ones, and in particular his notion of copycat links can be seen as connected to the use of names for parametricity.

In all of the above models the denotation of terms is built compositionally by induction on the structure of the term. In a different line of work, closer in spirit to our model, Lassen and Levy [16] have introduced normal form bisimulations for a language with parametric polymorphism. These bisimulations are defined on LTSs whose definition has similarities with ours. However, the model is for a CPS-style language which has not only polymorphic but also recursive types. Finally, our own model for a higher-order polymorphic language with general references [13] can be seen as a direct precursor to this work, albeit in a very different setting (call-by-value, with references).

Further on, we would like to study the existence of generic types in our model, as well as its dinaturality properties. We would moreover like to examine coarser notions of trace equivalence that bring us closer to Reynolds polymorphism. Finally, we would like to see if the trace model can be used to prove the original conjecture of [1,20]. While this seems plausible in principle, proving equivalences using definable logical relations requires additional tools, such as restrictions on the LTS, to avoid circular reasoning.

References

1. M. Abadi, L. Cardelli, and P.-L. Curien. Formal parametric polymorphism. *Theor. Comput. Sci.*, 121(1&2):9–58, 1993.
2. S. Abramsky and R. Jagadeesan. A game semantics for generic polymorphism. *Annals of Pure and Applied Logic*, 133(1):3 – 37, 2005. Festschrift on the occasion of Helmut Schwichtenberg's 60th birthday.

3. S. Abramsky, R. Jagadeesan, and P. Malacaria. Full abstraction for PCF. *Inf. Comput.*, 163(2):409–470, 2000.
4. E. S. Bainbridge, P. J. Freyd, A. Scedrov, and P. J. Scott. Functorial polymorphism. *Theor. Comput. Sci.*, 70(1):35–64, Jan. 1990.
5. J. de Lataillade. *Quantification du second ordre en sémentique des jeux: application aux isomorphismes de types*. PhD thesis, Paris 7, 2007.
6. J. de Lataillade. Second-order type isomorphisms through game semantics. *Annals of Pure and Applied Logic*, 151(2-3):115–150, 2008.
7. M. Gabbay and A. M. Pitts. A new approach to abstract syntax with variable binding. *Formal Asp. Comput.*, 13(3-5):341–363, 2002.
8. J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and types*, volume 7. Cambridge University Press Cambridge, 1989.
9. D. Hughes. *Hypergame semantics: full completeness for System F*. PhD thesis, D. Phil. thesis, Oxford University, 2000.
10. D. J. D. Hughes. Games and definability for System F. In *Proceedings of the 12th Annual IEEE Symposium on Logic in Computer Science*, LICS '97, pages 76–, Washington, DC, USA, 1997. IEEE Computer Society.
11. J. M. E. Hyland and C. L. Ong. On full abstraction for PCF: i, ii, and III. *Inf. Comput.*, 163(2):285–408, 2000.
12. G. Jaber. Operational nominal game semantics. In *Foundations of Software Science and Computation Structures - 18th International Conference, FoSSaCS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*, pages 264–278, 2015.
13. G. Jaber and N. Tzevelekos. Trace semantics for polymorphic references. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 585–594, 2016.
14. J. Laird. A fully abstract trace semantics for general references. In *Automata, Languages and Programming, 34th International Colloquium, ICALP 2007, Wroclaw, Poland, July 9-13, 2007, Proceedings*, pages 667–679, 2007.
15. J. Laird. Game semantics for a polymorphic programming language. *J. ACM*, 60(4):29:1–29:27, Sept. 2013.
16. S. B. Lassen and P. B. Levy. Typed normal form bisimulation for parametric polymorphism. In *Proceedings of the 2008 23rd Annual IEEE Symposium on Logic in Computer Science*, LICS '08, pages 341–352, Washington, DC, USA, 2008. IEEE Computer Society.
17. G. Longo, K. Milsted, and S. Soloviev. The genericity theorem and parametricity in the polymorphic lambda-calculus. *Theor. Comput. Sci.*, 121(1-2):323–349, Dec. 1993.
18. J. C. Mitchell. On the equivalence of data representations. In V. Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation*, pages 305–329. Academic Press Professional, Inc., San Diego, CA, USA, 1991.
19. A. M. Pitts. *Nominal Sets: Names and Symmetry in Computer Science*. Cambridge University Press, New York, NY, USA, 2013.
20. G. D. Plotkin and M. Abadi. A logic for parametric polymorphism. In *International Conference on Typed Lambda Calculi and Applications, TLCA'93, Proceedings*, pages 361–375, 1993.
21. J. C. Reynolds. Types, abstraction and parametric polymorphism. In *IFIP Congress*, pages 513–523, 1983.
22. C. Strachey. Fundamental concepts in programming languages. *Higher-order and symbolic computation*, 13(1):11–49, 2000.
23. P. Wadler. Theorems for free! In *Proceedings of the Fourth International Conference on Functional Programming Languages and Computer Architecture*, FPCA '89, pages 347–359, New York, NY, USA, 1989. ACM.