Applicative Bisimulation and Quantum λ-Calculi

Ugo Dal Lago, Alessandro Rioli

Applicative Bisimulation and Quantum λ-Calculi*

Ugo Dal Lago and Alessandro Rioli

Università di Bologna & INRIA
{ugo.dallago,alessandro.rioli2}@unibo.it

Abstract. Applicative bisimulation is a coinductive technique to check program equivalence in higher-order functional languages. It is known to be sound — and sometimes complete — with respect to context equivalence. In this paper we show that applicative bisimulation also works when the underlying language of programs takes the form of a *linear* λ-calculus extended with features such as probabilistic binary choice, but also quantum data, the latter being a setting in which linearity plays a role. The main results are proofs of soundness for the obtained notions of bisimilarity.

1 Introduction

Program equivalence is one of the fundamental notions in the theory of programming languages. Studying the nature of program equivalence is not only interesting from a purely foundational point of view, but can also be the first step towards defining (semi)automatic techniques for program verification, or for validating compiler optimisations. As an example, conformance of a program to a specification often corresponds to the equivalence between the program and the specification, once the latter is written in the same formalism as the program.

If the language at hand is an higher-order functional language, equivalence is traditionally formalised as Morris' *context equivalence*: two programs are considered equivalent if and only if they have the same behavior in *every possible* context [15]. This makes it relatively easy to prove two programs to be *not* equivalent, since this merely amounts to finding *one* context which separates them. On the other hand, proving two terms to be equivalent requires one to examine their behaviour in *every possible* context.

Various ways to alleviate the burden of proving context equivalence have been proposed in the literature, from CIU theorems (in which the class of contexts is restricted without altering the underlying relation [14]) to adequate denotational semantics, to logical relations [17]. We are here interested in coinductive techniques akin to bisimulation. Indeed, they have been shown to be very powerful, to the point of not only being sound, but even complete as ways to prove terms to be context equivalent [16]. Among the various notions of bisimulation which are known to be amenable to higher-order programs, the simplest one is

* This work is partially supported by the ANR project 12IS02001 PACE.

certainly Abramsky's applicative bisimulation [1], in which terms are seen as interactive objects and the interaction with their environment consists in taking input arguments or outputting observable results.

Applicative bisimulation is indeed well-known to be fully-abstract w.r.t. context equivalence when instantiated on plain, untyped, *deterministic* λ-calculi [1]. When the calculus at hand also includes a choice operator, the situation is more complicated: while applicative bisimilarity is invariably a congruence, thus sound for context equivalence, completeness generally fails [16, 13], even if some unexpected positive results have recently been obtained by Crubillé and the first author [4] in a probabilistic setting. An orthogonal issue is the one of linearity: does applicative bisimulation work well when the underlying calculus has linear types? The question has been answered positively, but only for deterministic λ-calculi [3, 2]. Finally, soundness does not hold in general if the programming language at hand has references [11].

In this paper, we define and study applicative bisimulation when instantiated on linear λ-calculi, starting with a purely deterministic language, and progressively extending it with probabilistic choice and quantum data, a setting in which linearity is an essential ingredient [19, 20]. The newly added features in the language are shown to correspond to mild variations in the underlying transition system, which in presence of probabilistic choice becomes a labelled Markov chain. The main contributions of this paper are congruence results for applicative bisimilarity in probabilistic and quantum λ-calculi, with soundness with respect to context equivalence as an easy corollary. In all the considered calculi, Howe's technique [9, 16] plays a key role.

This is the first successful attempt to apply coinductive techniques to quantum, higher-order, calculi. The literature offers some ideas and results about bisimulation and simulation in the context of quantum process algebras [8, 7, 6]. Deep relations between quantum computation and coalgebras have recently been discovered [10]. None of the cited works, however, deals with higher-order functions.

This paper is structured as follows. In Section 2, a simple linear λ-calculus, called ℓST_λ will be introduced, together with its operational semantics. This is a purely deterministic calculus, on top of which our extensions will be defined. Section 3 presents the basics of applicative bisimulation, instantiated on ℓST_λ. A probabilistic variation on ℓST_λ, called ℓPST_λ, is the subject of Section 4, which also discusses the impact of probabilities to equivalences and bisimilarity. Section 5 is about a quantum variation on ℓST_λ, dubbed ℓQST_λ, together with a study of bimilarity for it. Section 6 concludes the paper with a discussion about full-abstraction. An extended version of this paper with more details is available [5].

2 Linear λ-Calculi: A Minimal Core

In this section, a simple linear λ-calculus called ℓST_λ will be introduced, together with the basics of its operational semantics. *Terms* and *values* are generated by

the following grammar:

$$e, f ::= v \;\Big|\; ee \;\Big|\; \texttt{if}\ e\ \texttt{then}\ e\ \texttt{else}\ e \;\Big|\; \texttt{let}\ e\ \texttt{be}\ \langle x, x\rangle\ \texttt{in}\ e \;\Big|\; \Omega;$$
$$v, w ::= x \;\Big|\; \texttt{tt} \;\Big|\; \texttt{ff} \;\Big|\; \lambda x.e \;\Big|\; \langle v, v\rangle.$$

Observe the presence not only of abstractions and applications, but also of value pairs, and of basic constructions for booleans. Pairs of arbitrary terms can be formed as follows, as syntactic sugar:

$$\langle e, f\rangle = (\lambda x.\lambda y.\langle x, y\rangle)ef.$$

Finally, terms include a constant Ω for divergence. b is a metavariable for truth values, i.e. b stands for either $\texttt{tt}$ or $\texttt{ff}$. We need a way to enforce linearity, i.e., the fact that functions use their arguments *exactly* once. This can take the form of a linear type system whose language of *types* is the following:

$$A, B ::= \texttt{bool} \;\Big|\; A \multimap A \;\Big|\; A \otimes A.$$

The set $\mathcal{Y}$ includes all types. *Typing judgments* are in the form $\Gamma \vdash e : A$, where Γ is a set of assignments of types to variables. Typing rules are standard, and can be found in Figure 1. The set $\mathcal{T}^{\ell ST_\lambda}_{\Gamma,A}$ contains all terms e such that

$$\frac{}{x : A \vdash x : A} \qquad \frac{}{\vdash b : \texttt{bool}} \qquad \frac{\Gamma \vdash e : A \multimap B \qquad \Delta \vdash f : A}{\Gamma, \Delta \vdash ef : B} \qquad \frac{}{\Gamma \vdash \Omega : A}$$

$$\frac{\Gamma \vdash v : A \qquad \Delta \vdash w : B}{\Gamma, \Delta \vdash \langle v, w\rangle : A \otimes B} \qquad \frac{\Gamma, x : X, y : Y \vdash e : A \qquad \Delta \vdash f : X \otimes Y}{\Gamma, \Delta \vdash \texttt{let}\ f\ \texttt{be}\ \langle x, y\rangle\ \texttt{in}\ e : A}$$

$$\frac{\Gamma, x : A \vdash e : B}{\Gamma \vdash \lambda x.e : A \multimap B} \qquad \frac{\Gamma \vdash e : \texttt{bool} \qquad \Delta \vdash f : A \qquad \Delta \vdash g : A}{\Gamma, \Delta \vdash \texttt{if}\ e\ \texttt{then}\ f\ \texttt{else}\ g : A}$$

Fig. 1. Typing Rules

$\Gamma \vdash e : A$. $\mathcal{T}^{\ell ST_\lambda}_{\emptyset,A}$ is usually written as $\mathcal{T}^{\ell ST_\lambda}_{A}$. Notations like $\mathcal{V}^{\ell ST_\lambda}_{\Gamma,A}$ or $\mathcal{V}^{\ell ST_\lambda}_{A}$ are the analogues for values of the corresponding notations for terms.

Endowing ℓST_λ with call-by-value small-step or big-step semantics poses no significant problem. In the first case, one defines a binary relation $\rightarrow$ between closed terms of any type by the usual rule for β-reduction, the natural rule for the conditional operator, and the following rule: $\texttt{let}\ \langle v, w\rangle\ \texttt{be}\ \langle x, y\rangle\ \texttt{in}\ e \rightarrow e\{v, w/x, y\}$. Similarly, one can define a big-step evaluation relation $\Downarrow$ between closed terms and values by a completely standard set of rules (see [5] for more details). The expression $e \Downarrow$, as usual, indicates the existence of a value v with

$e \Downarrow v$. Subject reduction holds in the following sense: if $\emptyset \vdash e : A$, $e \to f$, and $e \Downarrow v$, then both $\emptyset \vdash f : A$ and $\emptyset \vdash v : A$.

The expressive power of the just-introduced calculus is rather poor. Nonetheless, it can be proved to be complete for first-order computation over booleans, in the following sense: for every function $F : \{\mathtt{tt}, \mathtt{ff}\}^n \to \{\mathtt{tt}, \mathtt{ff}\}$, there is a term which *computes* F, i.e. a term e_F such that $e_F\langle b_1, \ldots, b_n\rangle \Downarrow F(b_1, \ldots, b_n)$ for every $b_1, \ldots, b_n \in \{\mathtt{tt}, \mathtt{ff}\}^n$. Indeed, even if copying and erasing bits is not in principle allowed, one could anyway encode, e.g., duplication as the following combinator of type $\mathtt{bool} \multimap \mathtt{bool} \otimes \mathtt{bool}$: $\lambda x.\mathtt{if}\ x\ \mathtt{then}\ \langle \mathtt{tt}, \mathtt{tt}\rangle\ \mathtt{else}\ \langle \mathtt{ff}, \mathtt{ff}\rangle$. Similarly, if $\Gamma \vdash e : A$ and x is a fresh variable, one can easily find a term $\mathtt{weak}\ x\ \mathtt{in}\ e$ such that $\Gamma, x : \mathtt{bool} \vdash \mathtt{weak}\ x\ \mathtt{in}\ e : A$ and $\mathtt{weak}\ b\ \mathtt{in}\ e$ behaves like e for every $b \in \{\mathtt{ff}, \mathtt{tt}\}$.

But how could one capture program equivalence in an higher-order setting like the one we are examining? The canonical answer goes back to Morris [15], who proposed *context* equivalence (also known as *observational* equivalence) as the right way to compare terms. Roughly, two terms are context equivalent iff they behave the same when observed in any possible *context*, i.e. when tested against any possible *observer*. Formally, a context is nothing more than a term with a single occurrence of a special marker called the *hole* and denoted as $[\cdot]$ (see [5]). Given a context C and a term e, $C[e]$ is the term obtained by filling the single occurrence of $[\cdot]$ in C with e. For contexts to make sense in a typed setting, one needs to extend typing rules to contexts, introducing a set of rules deriving judgments in the form $\Gamma \vdash C[\Delta \vdash A] : B$, which can be read informally as saying that whenever $\Delta \vdash e : A$, it holds that $\Gamma \vdash C[e] : B$.

We are now in a position to define the context preorder: given two terms e and f such that $\Gamma \vdash e, f : A$, we write $e \leq_{\Gamma,A} f$ iff for every context C such that $\emptyset \vdash C[\Gamma \vdash A] : B$, if $C[e] \Downarrow$ then $C[f] \Downarrow$. If $e \leq_{\Gamma,A} f$ and $f \leq_{\Gamma,A} e$, then e and f are said to be *context equivalent*, and we write $e \equiv_{\Gamma,A} f$. What we have just defined, infact, are two *typed relations* $\leq$ and $\equiv$, that is to say two families of relations indexed by contexts and types, i.e. $\leq$ is the family $\{\leq_{\Gamma,A}\}_{\Gamma,A}$, while $\equiv$ is $\{\equiv_{\Gamma,A}\}_{\Gamma,A}$. If in the scheme above the type B is restricted so as to be $\mathtt{bool}$, then the obtained relations are the *ground* context preorder and *ground* context equivalence, respectively. Context equivalence is, almost by construction, a congruence. Similarly, the context preorder is easily seen to be a precongruence.

3 Applicative Bisimilarity and its Properties

Context equivalence is universally accepted as the canonical notion of equivalence of higher-order programs, being robust, and only relying on the underlying operational semantics. Proving terms *not* context equivalent is relatively easy: ending up with a single context separating the two terms suffices. On the other hand, the universal quantification over all contexts makes proofs of equivalence hard.

A variety of techniques have been proposed to overcome this problem, among them logical relations, adequate denotational models and context lemmas. As first proposed by Abramsky [1], coinductive methodologies (and the bisimulation proof method in particular) can be fruitfully employed. Abramsky's *applicative* bisimulation is based on taking argument passing as the basic interaction mechanism: what the environment can do with a λ-term is either evaluating it or passing it an argument.

In this section, we will briefly delineate how to define applicative bisimilarity for the linear λ-calculus ℓST_λ. We will do that in an unnecessarily pedantic way, defining a labelled transition system, and then playing the usual bisimulation game on top of it. This has the advantage of making the extensions to probabilistic and quantum calculi much easier.

A *labelled transition system* (LTS in the following) is a triple $\mathscr{L} = (\mathcal{S}, \mathcal{L}, \mathcal{N})$, where $\mathcal{S}$ is a set of *states*, $\mathcal{L}$ is a set of *labels*, and $\mathcal{N}$ is a subset of $\mathcal{S} \times \mathcal{L} \times \mathcal{S}$. If for every $s \in \mathcal{S}$ and for every $\ell \in \mathcal{L}$ there is *at most* one state $t \in \mathcal{S}$ with $(s, \ell, t) \in \mathcal{N}$, then $\mathscr{L}$ is said to be *deterministic*. The theory of bisimulation for LTSs is very well-studied [18] and forms one of the cornerstones of concurrency theory.

An applicative bisimulation relation is nothing more than a bisimulation on an LTS $\mathscr{L}_{\ell ST_\lambda}$ defined *on top of* the λ-calculus ℓST_λ. More specifically, the LTS $\mathscr{L}_{\ell ST_\lambda}$ is defined as the triple

$$(\overline{\mathcal{T}^{\ell ST_\lambda}} \uplus \overline{\mathcal{V}^{\ell ST_\lambda}}, \overline{\mathcal{E}^{\ell ST_\lambda}} \uplus \overline{\mathcal{V}^{\ell ST_\lambda}} \cup \{\mathit{eval}, \mathtt{tt}, \mathtt{ff}\} \cup (\mathcal{Y} \uplus \mathcal{Y}), \mathcal{N}_{\ell ST_\lambda}),$$

where:

- $\overline{\mathcal{T}^{\ell ST_\lambda}}$ is the set $\cup_{A \in \mathcal{Y}} (\mathcal{T}_A^{\ell ST_\lambda} \times \{A\})$, similarly for $\overline{\mathcal{V}^{\ell ST_\lambda}}$. On the other hand, $\overline{\mathcal{E}^{\ell ST_\lambda}}$ is $\cup_{A,B,E \in \mathcal{Y}} (\mathcal{T}^{\ell ST_\lambda}_{x:A, y:B, E} \times \{(A, B, E)\})$. Observe how any pair (v, A) appears twice as a state, once as an element of $\overline{\mathcal{T}^{\ell ST_\lambda}}$ and again as an element of $\overline{\mathcal{V}^{\ell ST_\lambda}}$. Whenever necessary to avoid ambiguity, the second instance will be denoted as $(\widehat{v}, A)$. Similarly for the two copies of any type A one finds as labels.
- The label *eval* models evaluation of terms, while the labels $\mathtt{tt}, \mathtt{ff}$ are the way a boolean constant declares its own value.
- The relation $\mathcal{N}_{\ell ST_\lambda}$ contains all triples in the following forms:

$$((\widehat{\mathtt{tt}}, \mathtt{bool}), \mathtt{tt}, (\widehat{\mathtt{tt}}, \mathtt{bool})); \qquad ((\widehat{\mathtt{ff}}, \mathtt{bool}), \mathtt{ff}, (\widehat{\mathtt{ff}}, \mathtt{bool}));$$
$$((\widehat{\lambda x.e}, A \multimap B), (v, A), (e\{v/x\}, B));$$
$$((\widehat{\langle v, w \rangle}, A \otimes B), (e, (A, B, E)), (e\{v/x, w/y\}, E));$$
$$((e, A), A, (e, A)); \qquad ((\widehat{v}, A), \widehat{A}, (\widehat{v}, A)); \qquad ((e, A), \mathit{eval}, (\widehat{v}, A));$$

where, in the last item, we of course assume that $e \Downarrow v$.

Basically, values interact with their environment based on their types: abstractions take an input argument, pairs gives their two components to a term which can handle them, and booleans constants simply expose their value. The only way to interact with terms is by evaluating them. Both terms and values expose

their type. As one can easily verify, the labelled transition system $\mathscr{L}_{\ell ST_\lambda}$ is deterministic. Simulation and bisimulation relations for $\mathscr{L}_{\ell ST_\lambda}$ are defined as for any other LTS. Notice, however, that both are binary relations on *states*, i.e., on elements of $\overline{\mathcal{T}^{\ell ST_\lambda}} \uplus \overline{\mathcal{V}^{\ell ST_\lambda}}$. Let us observe that:

- Two pairs (e, A) and (f, B) can be put in relation only if $A = B$, because each state makes its type public through a label. For similar reasons, states in the form (v, A) and $(\widehat{w}, B)$ cannot be in relation, not even if $A = B$.
- If (v, A) and (w, A) are in relation, then also $(\widehat{v}, A)$ and $(\widehat{w}, A)$ are in relation. Conversely, if $(\widehat{v}, A)$ and $(\widehat{w}, A)$ are in a (bi)simulation relation R, then $R \cup \{((v, A), (w, A))\}$ is itself a (bi)simulation.

As a consequence, (bi)similarity can be seen as a relation on terms, indexed by types. Similarity is denoted as $\preceq$, and its restriction to (closed) terms of type A is indicated with $\preceq_A$. For bisimilarity, symbols are $\sim$ and $\sim_A$, respectively. (Bi)similarity can be generalised to a typed relation by the usual open extension.

Example 1. An example of two distinct programs which can be proved bisimilar are the following:

$$e = \lambda x.\lambda y.\lambda z.\texttt{and}\ (xy)\ (\texttt{or}\ z\ \texttt{tt}); \qquad f = \lambda x.\lambda y.\lambda z.x(\texttt{or}\ (\texttt{and}\ z\ \texttt{ff})\ y);$$

where `and` and `or` are combinators computing the eponymous boolean functions. Both e and f can be given the type $(\texttt{bool} \multimap \texttt{bool}) \multimap \texttt{bool} \multimap \texttt{bool} \multimap \texttt{bool}$ in the empty context. They can be proved bisimilar by just giving a relation $R_{e,f}$ which contains the pair (e, f) and which can be proved to be an applicative bisimulation. Another interesting example of terms which can be proved bisimilar are the term $e = \texttt{if}\ f\ \texttt{then}\ g\ \texttt{else}\ h$ and the term s obtained from e by λ-abstracting all variables which occur free in g (and, equivalently, in h), then applying the same variables to the obtained term. For more details, see [5].

Is bisimilarity sound for (i.e., included in) context equivalence? And how about the reverse inclusion? For a linear, deterministic λ-calculus like the one we are describing, both questions have already been given a positive answer [7]. In the next two sections, we will briefly sketch how the correspondence can be proved.

3.1 (Bi)similarity is a (Pre)congruence.

A natural way to prove that similarity is included in the context preorder, (and thus that bisimilarity is included in context equivalence) consists in first showing that similarity is a *precongruence*, that is to say a preorder relation which is compatible with all the operators of the language.

While proving that $\preceq$ is a preorder is relatively easy, the naive proof of compatibility (i.e. the obvious induction) fails, due to application. A nice way out is due to Howe [9], who proposed a powerful and reasonably robust proof based on so-called precongruence candidates. Intuitively, the structure of Howe's method is the following:

1. First of all, one defines an operator $(\cdot)^H$ on typed relations, in such a way that whenever a typed relation R is a preorder, R^H is a precongruence.
2. One then proves, again under the condition that R is an equivalence relation, that R is included into R^H, and that R^H is substitutive.
3. Finally, one proves that $\preceq^H$ is itself an applicative simulation. This is the so-called Key Lemma [16], definitely the most difficult of the three steps.

Points 2 and 3 together imply that $\preceq$ and $\preceq^H$ coincide. But by point 1, $\preceq^H$, thus also $\preceq$, are precongruences. Points 1 and 2 do not depend on the underlying operational semantics, but on only on the language's constructs.

In Figure 2, one can find the full set of rules defining $(\cdot)^H$ when the underlying terms are those of ℓST_λ.

$$\frac{\emptyset \vdash cRt : A}{\emptyset \vdash cR^H t : A} \qquad \frac{x : A \vdash xRt : A}{\emptyset \vdash xR^H t : A}$$

$$\frac{\Gamma, x : B \vdash eR^H h : A \qquad \Gamma \vdash (\lambda\, x.h)Rt : B \multimap A}{\Gamma \vdash (\lambda\, x.e)R^H t : B \multimap A}$$

$$\frac{\Gamma \vdash eR^H h : B \multimap A \qquad \Delta \vdash fR^H s : B \qquad \Gamma, \Delta \vdash (hs)Rt : A}{\Gamma, \Delta \vdash (ef)R^H t : A}$$

$$\frac{\Gamma \vdash eR^H h : \mathtt{bool} \quad \Delta \vdash fR^H s : A \quad \Delta \vdash gR^H r : A \quad \Gamma, \Delta \vdash (\mathtt{if}\ h\ \mathtt{then}\ s\ \mathtt{else}\ r)Rt : A}{\Gamma, \Delta \vdash (\mathtt{if}\ e\ \mathtt{then}\ f\ \mathtt{else}\ g)R^H t : A}$$

$$\frac{\Gamma \vdash eR^H h : X \otimes Y \quad \Delta, x : X, y : Y \vdash fR^H s : A \quad \Gamma, \Delta \vdash (\mathtt{let}\ h\ \mathtt{be}\ \langle x, y\rangle\ \mathtt{in}\ s)Rt : A}{\Gamma, \Delta \vdash (\mathtt{let}\ e\ \mathtt{be}\ \langle x, y\rangle\ \mathtt{in}\ f)R^H t : A}$$

$$\frac{\Gamma \vdash vR^H u : A \qquad \Delta \vdash wR^H z : B \qquad \Gamma, \Delta \vdash \langle u, z\rangle Re : A \otimes B}{\Gamma, \Delta \vdash \langle v, w\rangle R^H e : A \otimes B}$$

Fig. 2. The Howe's Rules for ℓST_λ.

Theorem 1. *In ℓST_λ, $\preceq$ is included in $\leq$, thus $\sim$ is included in $\equiv$.*

4 Injecting Probabilistic Choice

The expressive power of ℓST_λ is rather limited, due to the presence of linearity. Nevertheless, the calculus is complete for first-order computations over the finite domain of boolean values, as discussed previously. Rather than relaxing linearity, we now modify ℓST_λ by endowing it with a form or probabilistic choice, thus obtaining a new linear λ-calculus, called ℓPST_λ, which is complete for probabilistic circuits. We see ℓPST_λ as an intermediate step towards ℓQST_λ, a quantum λ-calculus we will analyze in the following section.

The language of terms of ℓPST_λ is the one of ℓST_λ where, however, there is one additional binary construct $\oplus$, to be interpreted as probabilistic choice: $e ::= e \oplus e$. The set $\mathcal{Y}$ of types is the same as the one of ℓST_λ. An evaluation operation is introduced as a relation $\Downarrow \subseteq \mathcal{T}_{\emptyset,A}^{\ell PST_\lambda} \times \mathscr{D}_A^{\ell PST_\lambda}$ between the sets of closed terms of type A belonging to ℓPST_λ and the one of subdistributions of values of type A in ℓPST_λ. The elements of $\mathscr{D}_A^{\ell PST_\lambda}$ are actually subdistributions whose support is some finite subset of the set of values $\mathcal{V}_A^{\ell PST_\lambda}$, i.e., for each such $\mathscr{E}$, we have $\mathscr{E} : \mathcal{V}_A^{\ell PST_\lambda} \mapsto \mathbb{R}_{[0,1]}$ and $\sum_{v \in \mathcal{V}_A^{\ell PST_\lambda}} \mathscr{E}(v) \leq 1$. Whenever this does not cause ambiguity, subdistributions will be referred to simply as distributions. In Figure 3 a selection of the rules for big-step semantics in ℓPST_λ is given. Expressions in the form $\{v_i^{p_i}\}_{i \in I}$ have the obvious meaning, namely the distribution with support $\{v_i\}_{i \in I}$ which attributes probability p_i to each v_i.

As for the terms $e \in \mathcal{T}_A^{\ell PST_\lambda}$, the following lemma holds:

Lemma 1. *If $\emptyset \vdash e : A$, then there is a unique distribution $\mathscr{E}$ such that $e \Downarrow \mathscr{E}$.*

Lemma 1 only holds because the λ-calculus we are working with is linear, and as a consequence strongly normalising. If $e \Downarrow \mathscr{E}$, then the unique $\mathscr{E}$ from Lemma 1 is called the *semantics* of e and is denoted simply as $[\![e]\!]$.

$$\frac{}{v \Downarrow \{v^1\}} \qquad \frac{}{\Omega \Downarrow \emptyset} \qquad \frac{e \Downarrow \mathscr{E} \qquad f \Downarrow \mathscr{F} \qquad s\{w/x\} \Downarrow \mathscr{G}_{\lambda\, x.s,w}}{ef \Downarrow \sum_{\lambda\, x.s \in \mathfrak{S}(\mathscr{E}), w \in \mathfrak{S}(\mathscr{F})} \mathscr{E}(\lambda\, x.s)\mathscr{F}(w)\mathscr{G}_{\lambda\, x.s,w}}$$

$$\frac{e \Downarrow \mathscr{E} \qquad f \Downarrow \mathscr{F} \qquad g \Downarrow \mathscr{G}}{\texttt{if } e \texttt{ then } f \texttt{ else } g \Downarrow \mathscr{E}(\texttt{tt})\mathscr{F} + \mathscr{E}(\texttt{ff})\mathscr{G}} \qquad \frac{e \Downarrow \mathscr{E} \qquad f \Downarrow \mathscr{F}}{e \oplus f \Downarrow \frac{1}{2}\mathscr{E} + \frac{1}{2}\mathscr{F}}$$

Fig. 3. Big-step Semantics of ℓPST_λ — Selection

Context equivalence and the context preorder are defined very similarly to ℓST_λ, the only difference being the underlying notion of observation, which in ℓST_λ takes the form of *convergence*, and in ℓPST_λ becomes the *probability* of convergence.

4.1 Applicative Bisimilarity

Would it be possible to define applicative bisimilarity for ℓPST_λ similarly to what we have done for ℓST_λ? The first obstacle towards this goal is the dynamics of ℓPST_λ, which is not deterministic but rather probabilistic, and thus cannot fit into an LTS. In the literature, however, various notions of probabilistic bisimulation have been introduced, and it turns out that the earliest and simplest one, due to Larsen and Skou [12], is sufficient for our purposes.

A *labelled Markov chain* (LMC in the following) is a triple $(\mathcal{S}, \mathcal{L}, \mathcal{P})$, where $\mathcal{S}$ and $\mathcal{L}$ are as in the definition of a LTS, while $\mathcal{P}$ is a *transition probability matrix*, i.e., a function from $\mathcal{S}\times\mathcal{L}\times\mathcal{S}$ to $\mathbb{R}_{[0,1]}$ such that for every s and for every ℓ, it holds that $\mathcal{P}(s,\ell,\mathcal{S}) \leq 1$ (where the expression $\mathcal{P}(s,\ell,X)$ stands for $\sum_{t\in X}\mathcal{P}(s,\ell,t)$ whenever $X \subseteq \mathcal{S}$). Given such a LMC $\mathscr{M}$, an equivalence relation R on $\mathcal{S}$ is said to be a *bisimulation* on $\mathscr{M}$ iff whenever $(s,t) \in R$, it holds that $\mathcal{P}(s,\ell,E) = \mathcal{P}(t,\ell,E)$ for every equivalence class E of $\mathcal{S}$ modulo R. A preorder R on $\mathcal{S}$ is said to be a *simulation* iff for every subset X of $\mathcal{S}$, it holds that $\mathcal{P}(s,\ell,X) \leq \mathcal{P}(t,\ell,R(X))$. With some efforts (see [5] for some more details) one can prove that there exist largest bisimulation and simulation, that we continue to call *similarity* and *bisimilarity*, respectively. Probabilistic (bi)simulation, despite the endeavor required to define it, preserves all fundamental properties of its deterministic sibling. As an example, a symmetric probabilistic simulation is a bisimulation. Moreover, bisimilarity is the intersection of similarity and co-similarity.

Labelled Markov chains are exactly the objects we need when generalising the construction $\mathscr{L}_{\ell ST_\lambda}$ to ℓPST_λ. The LMC $\mathscr{M}_{\ell PST_\lambda}$, indeed, is defined as the triple

$$(\overline{\mathcal{T}^{\ell PST_\lambda}} \uplus \overline{\mathcal{V}^{\ell PST_\lambda}}, \overline{\mathcal{E}^{\ell PST_\lambda}} \uplus \overline{\mathcal{V}^{\ell PST_\lambda}} \cup \{eval, \mathtt{tt}, \mathtt{ff}\} \cup (\mathcal{Y} \uplus \mathcal{Y}), \mathcal{P}_{\ell PST_\lambda})$$

where $\mathcal{P}_{\ell PST_\lambda}$ is the function assuming the following values:

$$\mathcal{P}_{\ell PST_\lambda}((\widehat{\mathtt{tt}}, \mathtt{bool}), \mathtt{tt}, (\widehat{\mathtt{tt}}, \mathtt{bool})) = 1; \qquad \mathcal{P}_{\ell PST_\lambda}((\widehat{\mathtt{ff}}, \mathtt{bool}), \mathtt{ff}, (\widehat{\mathtt{ff}}, \mathtt{bool})) = 1;$$
$$\mathcal{P}_{\ell PST_\lambda}((\widehat{\lambda x.e}, A \multimap B), (v, A), (e\{v/x\}, B)) = 1;$$
$$\mathcal{P}_{\ell PST_\lambda}((\widehat{\langle v, w\rangle}, A \otimes B), (e, (A, B, E)), (e\{v/x, w/y\}, E)) = 1;$$
$$\mathcal{P}_{\ell PST_\lambda}((e, A), A, (e, A)) = 1 \qquad \mathcal{P}_{\ell PST_\lambda}((\widehat{v}, A), \widehat{A}, (\widehat{v}, A)) = 1;$$
$$\mathcal{P}_{\ell PST_\lambda}((e, A), eval, (\widehat{v}, A)) = [\![e]\!](v);$$

and having value 0 in all the other cases. It is easy to realise that $\mathcal{P}_{\ell PST_\lambda}$ can indeed be seen as the natural generalisation of $\mathcal{N}_{\ell ST_\lambda}$: on states in the form $(\widehat{v}, A)$, the function either returns 0 or 1, while in correspondence to states like (e, A) and the label *eval*, it behaves in a genuinely probabilistic way.

As for ℓST_λ, simulation and bisimulation relations, and the largest such relations, namely similarity and bisimilarity, can be given by just instantiating the general scheme described above to the specific LMC modeling terms of ℓPST_λ and their dynamics. All these turn out to be relations on *closed* terms, but as for ℓST_λ, they can be turned into proper typed relations just by the usual open extension.

The question now is: are the just introduced coinductive methodologies sound with respect to context equivalence? And is it that the proof of precongruence for similiarity from Section 3.1 can be applied here? The answer is positive, but some effort is needed. More specifically, one can proceed as in [4], generalising Howe's method to a probabilistic setting, which makes the Key Lemma harder to prove. By the way, the set of Howe's rules are the same as in ℓST_λ, except for a new one, namely

$$\frac{\Gamma \vdash eR^H h : A \qquad \Delta \vdash fR^H s : A \qquad \Gamma, \Delta \vdash (h \oplus s)Rt : A}{\Gamma, \Delta \vdash (e \oplus f)R^H t : A}$$

Thus:

Theorem 2. *In ℓPST_λ, $\preceq$ is included in $\leq$, thus $\sim$ is included in $\equiv$.*

5 On Quantum Data

Linear λ-calculi with classical control and quantum data have been introduced and studied both from an operational and from a semantical point of view [20, 7]. Definitionally, they can be thought of as λ-calculi in which ordinary, classic, terms have access to a so-called quantum register, which models quantum data.

A quantum register $\mathscr{Q}$ on a finite set of quantum variables $\mathcal{Q}$ is mathematically described by an element of a finite-dimensional Hilbert space whose computational basis is the set $\mathcal{SB}(\mathcal{Q})$ of all maps from $\mathcal{Q}$ to $\{\mathtt{tt}, \mathtt{ff}\}$ (of which there are $2^{|\mathcal{Q}|}$). Any element of this basis takes the form $|r_1 \leftarrow b_1, r_2 \leftarrow b_2, \cdots, r_n \leftarrow b_n\rangle$, where $\mathcal{Q} = \{r_1, \ldots, r_n\}$ and $b_1, \ldots, b_n \in \{\mathtt{tt}, \mathtt{ff}\}$. Elements of this Hilbert space, called $\mathcal{H}(\mathcal{Q})$, are in the form

$$\mathscr{Q} = \sum_{\eta \in \mathcal{SB}(\mathcal{Q})} \alpha_\eta |\eta\rangle, \tag{1}$$

where the complex numbers $\alpha_\eta \in \mathbb{C}$ are the so-called *amplitudes*, and must satisfy the *normalisation condition* $\sum_{\eta \in \mathcal{SB}(\mathcal{Q})} |\alpha_\eta|^2 = 1$. If $\eta \in \mathcal{SB}(\mathcal{Q})$ and r is a variable not necessarily in $\mathcal{Q}$, then $\eta\{r \leftarrow b\}$ stands for the substitution which coincides with η except on r where it equals b.

The interaction of a quantum register with the outer environment can create or destroy quantum bits increasing or decreasing the dimension of $\mathscr{Q}$. This shaping of the quantum register is mathematically described making use of the following operators:

- The probability operator $\mathtt{PR}_b^r : \mathcal{H}(\mathcal{Q}) \to \mathbb{R}_{[0,1]}$ gives the probability to obtain $b \in \{\mathtt{tt}, \mathtt{ff}\}$ as a result of the measurement of $r \in \mathcal{Q}$ in the input register:

$$\mathtt{PR}_b^r(\mathscr{Q}) = \sum_{\eta(r)=b} |\alpha_\eta|^2.$$

- If $r \in \mathcal{Q}$, then the projection operator $\mathtt{MS}_b^r : \mathcal{H}(\mathcal{Q}) \to \mathcal{H}(\mathcal{Q} - \{r\})$ measures the variable r, stored in the input register, destroying the corresponding qubit. More precisely $\mathtt{MS}_{\mathtt{tt}}^r(\mathscr{Q})$ and $\mathtt{MS}_{\mathtt{ff}}^r(\mathscr{Q})$ give as a result the quantum register configuration corresponding to a measure of the variable r, when the result of the variable measurement is $\mathtt{tt}$ or $\mathtt{ff}$, respectively:

$$\mathtt{MS}_b^r(\mathscr{Q}) = [\mathtt{PR}_b^r(\mathscr{Q})]^{-\frac{1}{2}} \sum_{\eta \in \mathcal{SB}(\mathcal{Q} - \{r\})} \alpha_{\eta\{r \leftarrow b\}} |\eta\rangle,$$

  where $\mathscr{Q}$ is as in (1).
- If $r \notin \mathcal{Q}$, then the operator $\mathtt{NW}_b^r : \mathcal{H}(\mathcal{Q}) \to \mathcal{H}(\mathcal{Q} \cup \{r\})$ creates a new qubit, accessible through the fresh variable name r, and increases the dimension of the quantum register by one .

Qubits can not only be created and measured, but their value can also be *modified* by applying unitary operators to them. Given any such n-ary operator U, and any sequence of distinct variables $r_1, \ldots, r_n$ (where $r_i \in \mathcal{Q}$ for every $1 \leq i \leq n$), one can build a unitary operator $U_{r_1,\ldots,r_n}$ on $\mathcal{H}(\mathcal{Q})$.

5.1 The Language

We can obtain the quantum language ℓQST_λ as an extension of basic ℓST_λ. The grammar of ℓST_λ is enhanced by adding the following values:

$$e ::= U(v) \mid \texttt{meas}(v) \mid \texttt{new}(v); \qquad\qquad v ::= r;$$

where r ranges over an infinite set of quantum variables, and U ranges over a finite set of unitary transformations. The term $\texttt{new}(v)$ acting on boolean constant, returns (a quantum variable pointing to) a qubit of the same value, increasing this way the dimension of the quantum register. The term $\texttt{meas}(v)$ measures a value of type qubit, therefore it decreases the dimension of the quantum register.

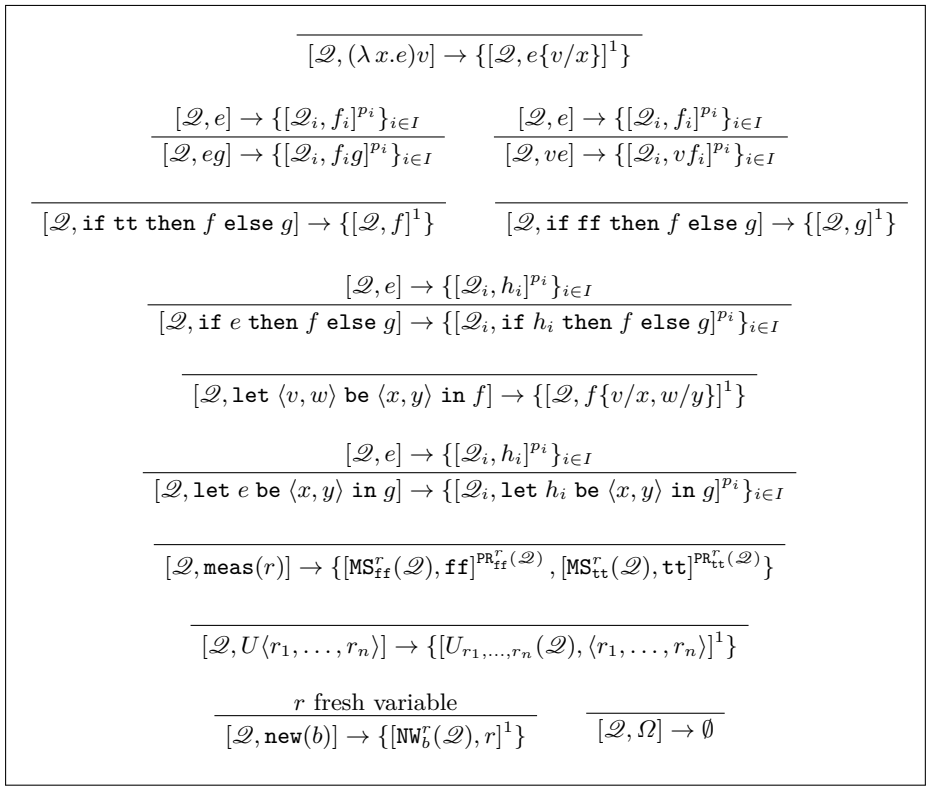
Typing terms in ℓQST_λ does not require any particular efforts. The class of types needs to be sligthly extended with a new base type for qubits, called `qbit`, while contexts now give types not only to classical variables, but also to quantum variables. The new typing rules are in Figure 4.

$$\frac{\Gamma \vdash v : \texttt{qbit}}{\Gamma \vdash \texttt{meas}(v) : \texttt{bool}} \qquad \frac{\Gamma \vdash v : \texttt{bool}}{\Gamma \vdash \texttt{new}(v) : \texttt{qbit}}$$

$$\frac{\Gamma \vdash v : \texttt{qbit}^{\otimes n}}{\Gamma \vdash U(v) : \texttt{qbit}^{\otimes n}} \qquad \frac{}{r : \texttt{qbit} \vdash r : \texttt{qbit}}$$

Fig. 4. Typing rules in ℓQST_λ.

The semantics of ℓQST_λ, on the other hand, cannot be specified merely as a relation between terms, since terms only make sense computationally if coupled with a quantum register, namely in a pair in the form $[\mathcal{Q}, e]$, which is called a *quantum closure*. Analogously to what has been made for ℓPST_λ, small step reduction operator $\rightarrow$ and the big step evaluation operator $\Downarrow$ are given as relations between the set of quantum closures and of quantum closures distributions. In figures 5 and 6 the small-step semantics and big-step semantics for ℓQST_λ are given. Quantum closures, however, are not what we want to compare, since what we want to be able to compare are *terms*. Context equivalence, in other words, continues to be a relation on terms, and can be specified similarly to the probablistic case, following, e.g. [20].



$$\frac{}{[\mathscr{Q}, (\lambda\, x.e)v] \to \{[\mathscr{Q}, e\{v/x\}]^1\}}$$

$$\frac{[\mathscr{Q}, e] \to \{[\mathscr{Q}_i, f_i]^{p_i}\}_{i\in I}}{[\mathscr{Q}, eg] \to \{[\mathscr{Q}_i, f_i g]^{p_i}\}_{i\in I}} \qquad \frac{[\mathscr{Q}, e] \to \{[\mathscr{Q}_i, f_i]^{p_i}\}_{i\in I}}{[\mathscr{Q}, ve] \to \{[\mathscr{Q}_i, v f_i]^{p_i}\}_{i\in I}}$$

$$\frac{}{[\mathscr{Q}, \texttt{if tt then } f \texttt{ else } g] \to \{[\mathscr{Q}, f]^1\}} \qquad \frac{}{[\mathscr{Q}, \texttt{if ff then } f \texttt{ else } g] \to \{[\mathscr{Q}, g]^1\}}$$

$$\frac{[\mathscr{Q}, e] \to \{[\mathscr{Q}_i, h_i]^{p_i}\}_{i\in I}}{[\mathscr{Q}, \texttt{if } e \texttt{ then } f \texttt{ else } g] \to \{[\mathscr{Q}_i, \texttt{if } h_i \texttt{ then } f \texttt{ else } g]^{p_i}\}_{i\in I}}$$

$$\frac{}{[\mathscr{Q}, \texttt{let } \langle v, w\rangle \texttt{ be } \langle x, y\rangle \texttt{ in } f] \to \{[\mathscr{Q}, f\{v/x, w/y\}]^1\}}$$

$$\frac{[\mathscr{Q}, e] \to \{[\mathscr{Q}_i, h_i]^{p_i}\}_{i\in I}}{[\mathscr{Q}, \texttt{let } e \texttt{ be } \langle x, y\rangle \texttt{ in } g] \to \{[\mathscr{Q}_i, \texttt{let } h_i \texttt{ be } \langle x, y\rangle \texttt{ in } g]^{p_i}\}_{i\in I}}$$

$$\frac{}{[\mathscr{Q}, \texttt{meas}(r)] \to \{[\mathtt{MS}^r_{\mathtt{ff}}(\mathscr{Q}), \mathtt{ff}]^{\mathtt{PR}^r_{\mathtt{ff}}(\mathscr{Q})}, [\mathtt{MS}^r_{\mathtt{tt}}(\mathscr{Q}), \mathtt{tt}]^{\mathtt{PR}^r_{\mathtt{tt}}(\mathscr{Q})}\}}$$

$$\frac{}{[\mathscr{Q}, U\langle r_1, \ldots, r_n\rangle] \to \{[U_{r_1,\ldots,r_n}(\mathscr{Q}), \langle r_1, \ldots, r_n\rangle]^1\}}$$

$$\frac{r \text{ fresh variable}}{[\mathscr{Q}, \texttt{new}(b)] \to \{[\mathtt{NW}^r_b(\mathscr{Q}), r]^1\}} \qquad \frac{}{[\mathscr{Q}, \Omega] \to \emptyset}$$

Fig. 5. Small-step Semantics of ℓQST_λ.

5.2 Applicative Bisimilarity in ℓQST_λ

Would it be possible to have a notion of bisimilarity for ℓQST_λ? What is the underlying "Markov Chain"? It turns out that LMCs as introduced in Section 4.1 are sufficient, but we need to be careful. In particular, states of the LMC are not terms, but quantum closures, of which there are in principle nondenumerably many. However, since we are only interested in quantum closures which can be obtained (in a finite number of evaluation steps) from closures having an empty quantum register, this is not a problem: we simply take states as *those* closures, which we dub *constructible*. $\mathscr{M}_{\ell QST_\lambda}$ can be built similarly to $\mathscr{M}_{\ell PST_\lambda}$, where (constructible) quantum closures take the place of terms. The non zero elements of the function $\mathcal{P}_{\ell QST_\lambda}$ are defined as follows:

$$\mathcal{P}_{\ell QST_\lambda}((\widehat{\mathtt{tt}}, \mathtt{bool}), \mathtt{bool}, (\widehat{\mathtt{tt}}, \mathtt{bool})) = 1;$$
$$\mathcal{P}_{\ell QST_\lambda}((\widehat{\mathtt{ff}}, \mathtt{bool}), \mathtt{bool}, (\widehat{\mathtt{ff}}, \mathtt{bool})) = 1;$$
$$\mathcal{P}_{\ell QST_\lambda}(([\mathscr{Q}, \widehat{r}], \mathtt{qbit}), ([\mathscr{W}, e], A), ([\mathscr{Q} \otimes \mathscr{W}, e\{r/x\}], A)) = 1;$$
$$\mathcal{P}_{\ell QST_\lambda}((\left[\mathscr{Q}, \widehat{\lambda x.e}\right], A \multimap B), ([\mathscr{W}, \hat{v}], A), ([\mathscr{Q} \otimes \mathscr{W}, e\{v/x\}], B)) = 1;$$

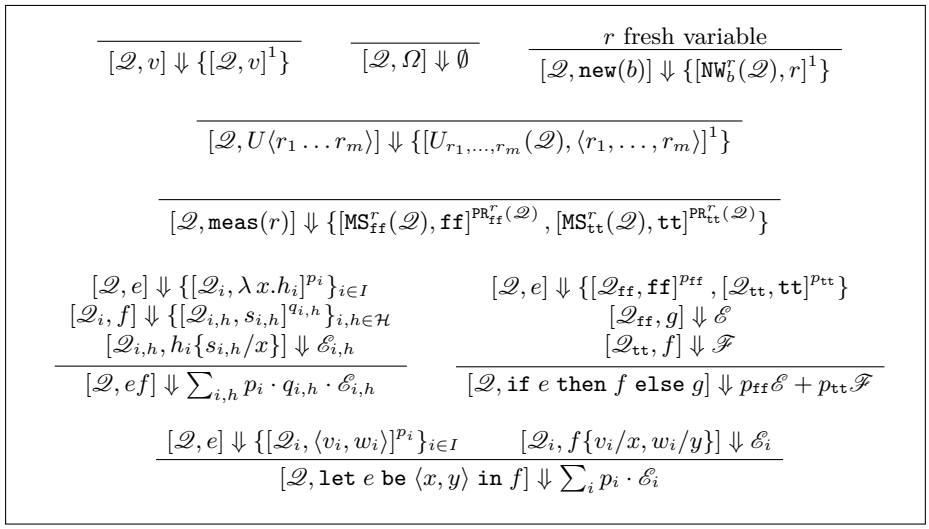


Fig. 6. Big-step Semantics of ℓQST_λ.

$$\mathcal{P}_{\ell QST_\lambda}((\left[\mathscr{Q}, \widehat{\langle v, w\rangle}\right], A \otimes B), ([\mathscr{W}, e], (A, B, E)), ([\mathscr{Q} \otimes \mathscr{W}, e\{v/x, w/y\}], E)) = 1;$$
$$\mathcal{P}_{\ell QST_\lambda}(([\mathscr{Q}, e], A), A, ([\mathscr{Q}, e], A)) = 1 \qquad \mathcal{P}_{\ell QST_\lambda}(([\mathscr{Q}, \widehat{e}], A), A, ([\mathscr{Q}, \widehat{e}], A)) = 1;$$
$$\mathcal{P}_{\ell QST_\lambda}(([\mathscr{Q}, e], A), \mathit{eval}, ([\mathscr{U}, v], A)) = [\![[\mathscr{Q}, e]]\!]([\mathscr{U}, v]).$$

Once we have a LMC, it is easy to apply the same definitional scheme we have seen for ℓPST_λ, and obtain a notion of applicative (bi)similarity. Howe's method, in turn, can be adapted to the calculus here, resulting in a proof of precongruence and ultimately in the following:

Theorem 3. *In ℓQST_λ, $\preceq$ is included in $\leq$, thus $\sim$ is included in $\equiv$.*

More details on the proof of this can be found in [5].

Example 2. An interesting pair of terms which can be proved bisimilar are the following two:

$$e = \lambda x.\texttt{if}\ (\texttt{meas}\ x)\ \texttt{then ff else tt}; \qquad f = \lambda x.\texttt{meas}(X\ x);$$

where X is the unitary operator which flips the value of a qubit. This is remarkable given, e.g. the "non-local" effects entanglement could cause.

6 On Full-Abstraction

In the deterministic calculus ℓST_λ, bisimilarity not only is *included* into context equivalence, but *coincides* with it (and, analogously, similarity coincides with the context preorder). This can be proved by observing that in $\mathscr{L}_{\ell ST_\lambda}$, bisimilarity

coincides with trace equivalence, and each linear test, i.e., each trace, can be implemented by a context. This result is not surprising, and has already been obtained in similar settings elsewhere [2].

But how about ℓPST_λ and ℓQST_λ? Actually, there is little hope to prove full-abstraction between context equivalence and bisimilarity in a linear setting if probabilistic choice is present. Indeed, as shown by van Breugel et al. [21], probabilistic bisimilarity can be characterised by a notion of test equivalence where tests can be *conjunctive*, i.e., they can be in the form $t = \langle s, p\rangle$, and t succeeds if both s and p succeeds. Implementing conjuctive tests, thus, requires *copying* the tested term, which is impossible in a linear setting. Indeed, it is easy to find a counterexample to full-abstraction already in ℓPST_λ. Consider the following two terms, both of which can be given type $\texttt{bool} \multimap \texttt{bool}$ in ℓPST_λ:

$$e = \lambda x.\texttt{weak}\ x\ \texttt{in}\ \texttt{tt} \oplus \texttt{ff}; \qquad f = (\lambda x.\texttt{weak}\ x\ \texttt{in}\ \texttt{tt}) \oplus (\lambda x.\texttt{weak}\ x\ \texttt{in}\ \texttt{ff}).$$

The two terms are not bisimilar, simply because `tt` and `ff` are not bisimilar, and thus also $\lambda x.\texttt{weak}\ x\ \texttt{in}\ \texttt{tt}$ and $\lambda x.\texttt{weak}\ x\ \texttt{in}\ \texttt{ff}$ cannot be bisimilar. However, e and f can be proved to be context equivalent: there is simply no way to discriminate between them by way of a linear context (see [5] for more details).

What one may hope to get is full-abstraction for extensions of the considered calculi in which duplication is reintroduced, although in a controlled way. This has been recently done in a probabilistic setting by Crubillé and the first author [4], and is the topic of current investigations by the authors for a non-strictly-linear extension of ℓQST_λ.

7 Conclusions

We show that Abramsky's applicative bisimulation can be adapted to linear λ-calculi endowed with probabilistic choice and quantum data. The main result is that in both cases, the obtained bisimilarity relation is a congruence, thus included in context equivalence.

For the sake of simplicity, we have deliberately kept the considered calculi as simple as possible. We believe, however, that many extensions would be harmless. This includes, as an example, generalising types to *recursive* types which, although infinitary in nature, can be dealt with very easily in a coinductive setting. Adding a form of controlled duplication requires more care, e.g. in presence of quantum data (which cannot be duplicated).

References

1. S. Abramsky. The lazy λ-calculus. In D. Turner, editor, *Research Topics in Functional Programming*, pages 65–117. Addison Wesley, 1990.
2. Gavin M. Bierman. Program equivalence in a linear functional language. *J. Funct. Program.*, 10(2):167–190, 2000.
3. Roy L. Crole. Completeness of bisimilarity for contextual equivalence in linear theories. *Electronic Journal of the IGPL*, 9(1), January 2001.

4. Raphaëlle Crubillé and Ugo Dal Lago. On probabilistic applicative bisimulation and call-by-value λ–calculi. In *ESOP*, volume 8410 of *LNCS*, pages 209–228, 2014.
5. Ugo Dal Lago and Alessandro Rioli. Applicative bisimulation and quantum λ-calculi (long version). Available at `http://eternal.cs.unibo.it/abqlc.pdf`, 2014.
6. Timothy A. S. Davidson, Simon J. Gay, Hynek Mlnarik, Rajagopal Nagarajan, and Nick Papanikolaou. Model checking for communicating quantum processes. *IJUC*, 8(1):73–98, 2012.
7. Yuxin Deng and Yuan Feng. Open bisimulation for quantum processes. *CoRR*, abs/1201.0416, 2012.
8. Simon J. Gay and Rajagopal Nagarajan. Communicating quantum processes. In *POPL*, pages 145–157, 2005.
9. Douglas J. Howe. Proving congruence of bisimulation in functional programming languages. *Inf. Comput.*, 124(2):103–112, 1996.
10. Bart Jacobs. Coalgebraic walks, in quantum and turing computation. In *FOSSACS*, volume 6604 of *LNCS*, pages 12–26, 2011.
11. Vasileios Koutavas, Paul Blain Levy, and Eijiro Sumii. From applicative to environmental bisimulation. *Electr. Notes Theor. Comput. Sci.*, 276:215–235, 2011.
12. Kim Guldstrand Larsen and Arne Skou. Bisimulation through probabilistic testing. *Inf. Comput.*, 94(1):1–28, 1991.
13. Søren B. Lassen and Corin Pitcher. Similarity and bisimilarity for countable non-determinism and higher-order functions. *Electr. Notes Theor. Comput. Sci.*, 10:246–266, 1997.
14. Robin Milner. Fully abstract models of typed λ-calculi. *Theor. Comput. Sci.*, 4:1–22, 1977.
15. J. Morris. *Lambda Calculus Models of Programming Languages*. PhD thesis, MIT, 1969.
16. Andrew M. Pitts. Operationally-based theories of program equivalence. In *Semantics and Logics of Computation*, pages 241–298. Cambridge University Press, 1997.
17. Gordon Plotkin. Lambda definability and logical relations. In *Memo SAI-RM-4, School of Artificial Intelligence*. Edinburgh, 1973.
18. Davide Sangiorgi. *Introduction to Bisimulation and Coinduction*. Cambridge Universtity Press, 2012.
19. Peter Selinger and Benoît Valiron. A lambda calculus for quantum computation with classical control. In *TLCA*, volume 3461 of *LNCS*, pages 354–368, 2005.
20. Peter Selinger and Benoît Valiron. On a fully abstract model for a quantum linear functional language. *Electron. Notes Theor. Comput. Sci.*, 210:123–137, 2008.
21. Franck van Breugel, Michael W. Mislove, Joël Ouaknine, and James Worrell. Domain theory, testing and simulation for labelled Markov processes. *Theor. Comput. Sci.*, 333(1-2):171–197, 2005.