On Equivalences, Metrics, and Polynomial Time

Alberto Cappai, Ugo Dal Lago

On Equivalences, Metrics, and Polynomial Time*

Alberto Cappai and Ugo Dal Lago

Università di Bologna & INRIA
{ugo.dallago,alberto.cappai2}@unibo.it

Abstract. Interactive behaviors are ubiquitous in modern cryptography, but are also present in λ-calculi, in the form of higher-order constructions. Traditionally, however, typed λ-calculi simply do not fit well into cryptography, being both deterministic and too powerful as for the complexity of functions they can express. We study interaction in a λ-calculus for probabilistic polynomial time computable functions. In particular, we show how notions of context equivalence and context metric can both be characterized by way of traces when defined on linear contexts. We then give evidence on how this can be turned into a proof methodology for computational indistinguishability, a key notion in modern cryptography. We also hint at what happens if a more general notion of a context is used.

1 Introduction

Modern cryptography [14] is centered around the idea that security of cryptographic constructions needs to be defined precisely and, in particular, that crucial aspects are *how* an adversary interacts with the construction, and *when* he wins this game. The former is usually specified by way of an *experiment*, while the latter is often formulated stipulating that the probability of a favorable result for the adversary needs to be small, where being "small" usually means being *negligible* in a security parameter. This framework would however be vacuous if the adversary had access to an unlimited amount of resources, or if it were deterministic. As a consequence the adversary is usually assumed to work within probabilistic polynomial time (PPT in the following), this way giving rise to a robust definition. Summing up, there are three key concepts here, namely *interaction*, *probability* and *complexity*. Security as formulated above can often be spelled out semantically as the so-called *computational indistinguishability* between two distributions, the first one being the one produced by the construction and the second one modeling an idealized construction or a genuinely random object.

Typed λ-calculi as traditionally conceived, do not fit well into this picture. Higher-order types clearly allow a certain degree of interaction, but probability and complexity are usually absent: reduction is deterministic (or at least confluent), while the expressive power of λ-calculi tends to be very high. This picture has somehow changed in the last ten years: there have been some successful

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

attempts at giving probabilistic λ-calculi whose representable functions coincide with the ones which can be computed by PPT algorithms [15, 17, 5]. These calculi invariably took the form of restrictions on Gödel's T, endowed with a form of binary probabilistic choice. All this has been facilitated by implicit computational complexity, which offers the right idioms to start from [11, 12], themselves based on linearity and ramification. The emphasis in all these works were either the characterization of probabilistic complexity classes [5], or more often security [17, 16]: one could see λ-calculi as a way to specify cryptographic constructions and adversaries for them. The crucial idea here is that computational indistinguishability can be formulated as a form of context equivalence. The real challenge, however, is whether all this can be characterized by handier notions, which would alleviate the inherently difficult task of dealing with all contexts when proving two terms to be equivalent.

The literature offers many proposals going precisely in this direction: this includes logical relations, context lemmas, or coinductive techniques. In applicative bisimulation [1], as an example, terms are modeled as interactive objects. This way, one focuses on how the interpreted program interacts with its environment, rather than on its internal evolution. None of them have so far been applied to calculi capturing probabilistic polynomial time, and relatively few among them handle probabilistic behavior.

In this paper, we study notions of equivalence and distance in one of these λ-calculi, called RSLR [5]. More precisely:

- After having briefly introduced RSLR and studied its basic metatheoretical properties (Section 2), we define *linear context equivalence*. We then show how the role of contexts can be made to play by *traces*. Finally, a coinductive notion of equivalence in the style of Abramsky's bisimulation is introduced. We also hint at how all this can be extended to metrics. This can be found in Section 4.
- We then introduce a notion of *parametrized context equivalence* for RSLR terms, showing that it coincides with computational indistinguishability when the compared programs are of base type. We then turn our attention to the problem of characterizing the obtained notion of equivalence by way of linear tests, giving a positive answer to that by way of a notion of parametrized trace metric. A brief discussion about the role of linear contexts in cryptography is also given. All this is in Section 5.

An extended version of this paper with more details is available [2].

2 Characterizing Probabilistic Polynomial Time

In this section we introduce RSLR [5], a λ-calculus for probabilistic polynomial time computation, obtained by extending Hofmann's SLR [12] with an operator for binary probabilistic choice. Compared to other presentations of the same calculus, we consider a call-by-value reduction and elide nonlinear function spaces and pairs. This has the advantage of making the whole theory less baroque, without any fundamental loss in expressiveness (see Section 5.1 below).

First of all, *types* are defined as follows:

$$\mathsf{A} ::= \ \mathsf{Str} \ \Big| \ \blacksquare\mathsf{A} \to \mathsf{A} \ \Big| \ \square\,\mathsf{A} \to \mathsf{A}.$$

The expression Str serves to type strings, and is the only *base* type. $\blacksquare\mathsf{A} \to \mathsf{B}$ is the type of functions (from A to B) which can be evaluated in constant time, while for $\square\mathsf{A} \to \mathsf{B}$ the running time can be any polynomial. *Aspects* are the elements of $\{\square, \blacksquare\}$ and are indeed fundamental to ensure polytime soundness. We denote them with metavariables like a or b. We define a partial order $<:$ between aspects simply as the relation $\{(\square, \square), (\square, \blacksquare), (\blacksquare, \blacksquare)\}$, and subtyping by the rules below:

$$\frac{}{\mathsf{A} <: \mathsf{A}} \qquad \frac{\mathsf{A} <: \mathsf{B} \qquad \mathsf{B} <: \mathsf{C}}{\mathsf{A} <: \mathsf{C}} \qquad \frac{\mathsf{B} <: \mathsf{A} \qquad \mathsf{C} <: \mathsf{D} \qquad \mathsf{a} <: \mathsf{b}}{\mathsf{aA} \to \mathsf{C} <: \mathsf{bB} \to \mathsf{D}}$$

The syntactical categories of *terms* and *values* are defined by the following grammar:

$$\begin{aligned} t \ &::= x \ \Big| \ v \ \Big| \ \mathsf{0}(t) \ \Big| \ \mathsf{1}(t) \ \Big| \ \mathsf{tail}(t) \ \Big| \ tt \\ &\quad \Big| \ \mathsf{case}_{\mathsf{A}}(t,t,t,t) \ \Big| \ \mathsf{rec}_{\mathsf{A}}(t,t,t,t) \ \Big| \ \mathsf{rand}; \\ v \ &::= \underline{m} \ \Big| \ \lambda x : \mathsf{aA}.t; \end{aligned}$$

where m ranges over the set $\{0,1\}^*$ of finite, binary strings, while x ranges over a denumerable set of variables X. We write T, V for the sets of terms and values, respectively. The operators $\mathsf{0}$ and $\mathsf{1}$ are constructors for binary strings, while tail is a destructor. The only nonstandard constant is rand, which returns $\underline{0}$ or $\underline{1}$, each with probability $\frac{1}{2}$, thus modeling uniform binary choice. The terms $\mathsf{case}_{\mathsf{A}}(t, t_0, t_1, t_\epsilon)$ and $\mathsf{rec}_{\mathsf{A}}(t, t_0, t_1, t_\epsilon)$ are terms for case distinction and recursion, in which first argument specifies the term (of base type) which guides the process. The expression $\underline{\epsilon}$ stands for the empty string and we set $\mathsf{tail}(\underline{\epsilon}) \to \underline{\epsilon}$.

As usual, a *typing context* Γ is a finite set of assignments of an aspect and a type to a variable, where as usual any variable occurs *at most once*. Any such assignment is indicated with $x : \mathsf{aA}$. The expression Γ, Δ stands for the union of the two typing contexts Γ and Δ, which are assumed to be disjoint. The union Γ, Δ is indicated with $\Gamma; \Delta$ whenever we want to insist on Γ to only involve the base type Str. *Typing judgments* are in the form $\Gamma \vdash t : \mathsf{A}$. Typing rules are in Figure 1. The expression $\mathrm{T}^{\mathsf{A}}_{\Gamma}$ (respectively, $\mathrm{V}^{\mathsf{A}}_{\Gamma}$) stands for the set of terms (respectively, values) of type A under the typing context Γ. Please observe how the type system we have just introduced enforces variables of higher-order type to occur free at most once and outside the scope of a recursion. Moreover, the type of terms which serve as step-functions in a recursion are assumed to be $\square$-free, and this is precisely what allow this calculus to characterize polytime functions.

The operational semantics of RSLR is of course probabilistic: any closed term t evaluates not to a single value but to a *value distribution*, i.e, a function $\mathcal{D} : \mathrm{V} \to \mathbb{R}$ such that $\sum_{v \in \mathrm{V}} \mathcal{D}(v) = 1$. Judgments expressing this fact are in the

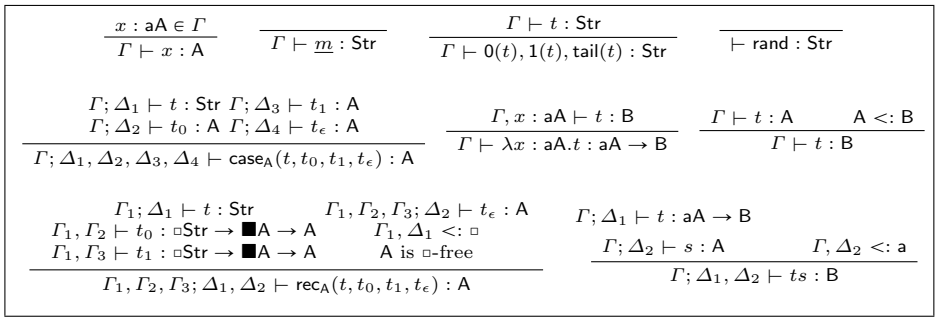

Fig. 1. RSLR's Typing Rules

form $t \Downarrow \mathcal{D}$, and are derived through a formal system (see [2] for more details). In the rest of this paper, we use some standard notation on distributions: the expression $\{v_1^{\alpha_1}, \ldots, v_n^{\alpha_n}\}$ stands for the distribution assigning probability α_i to v_i (for every $1 \leqslant i \leqslant n$), while the support of a distribution $\mathcal{D}$ is indicated with $\mathsf{S}(\mathcal{D})$. Given a set X, $\mathbb{P}_X$ is the set of all distributions over X. Noticeably:

Lemma 1. *For every term $t \in \mathrm{T}_{\varnothing}^{\mathsf{A}}$ there is a unique distribution $\mathcal{D}$ such that $t \Downarrow \mathcal{D}$, which we denote as $[\![t]\!]$. Moreover, If $v \in \mathsf{S}(\mathcal{D})$, then $v \in \mathrm{V}_{\varnothing}^{\mathsf{A}}$.*

This tells us both that $\Downarrow$ can be seen as a *function*, and that subject reduction holds.

A *probabilistic function* on $\{0,1\}^*$ is a function F from $\{0,1\}^*$ to $\mathbb{P}_{\{0,1\}^*}$. A term $t \in \mathrm{T}_{\varnothing}^{\mathsf{aStr} \to \mathsf{Str}}$ is said to *compute* F iff for every string $\underline{m} \in \{0,1\}^*$ it holds that $t\underline{m} \Downarrow \mathcal{D}$ where $\mathcal{D}(\underline{n}) = F(\underline{m})(\underline{n})$ for every $\underline{n} \in \{0,1\}^*$. What makes RSLR very interesting, however, is that it precisely captures those probabilistic functions which can be computed in polynomial time (see, e.g., [6] for a definition):

Theorem 1 (Polytime Completeness). *The set of probabilistic functions which can be computed by* RSLR *terms coincides with the polytime computable ones.*

This result is well-known [17, 5], and can be proved in various ways, e.g. combinatorially or categorically.

3 Equivalences

Intuitively, we can say that two programs are equivalent if no one can distinguish them by observing their external, visible, behavior. A formalization of this intuition usually takes the form of context equivalence. A *context* is a term in which the *hole* $[\cdot]$ occurs at most once. Formally, contexts are defined by the following grammar:

$$C ::= t \mid [\cdot] \mid \lambda x.C \mid Ct \mid tC \mid \mid \mathsf{0}(C) \mid \mathsf{1}(C) \mid \mathsf{tail}(C)$$
$$\mid \mathsf{case}_{\mathsf{A}}(C,t,t,t) \mid \mathsf{case}_{\mathsf{A}}(t,C,C,C) \mid \mathsf{rec}_{\mathsf{A}}(C,t,t,t).$$

What the above definition already tells us is that our emphasis in this paper will be on *linear* contexts, which are contexts whose holes lie outside the scope of any recursion operator. Given a term t we define $C[t]$ as the term obtained by substituting the occurrences of $[\cdot]$ in C (if any) with t. We only consider non-binding contexts here, i.e. contexts are meant to be filled with closed terms (this can be justified formally [2]). In other words, the type system from Section 2 can be turned into one for contexts whose judgments take the form $\Gamma \vdash C[\vdash \mathsf{A}] : \mathsf{B}$, which means that for every closed term t of type A, it holds that $\Gamma \vdash C[t] : \mathsf{B}$. See [2] for more details. Now that the notion of a context has been properly defined, one can finally give the central notion of equivalence in this paper.

Definition 1 (Context Equivalence). *Given two terms t, s such that $\vdash t, s : A$, we say that t and s are* context equivalent *iff for every context C such that $\vdash C[\vdash A] : \mathsf{Str}$ we have that $[\![C[t]]\!](\underline{\epsilon}) = [\![C[s]]\!](\underline{\epsilon})$.*

The way we defined it means that context equivalence is a family of relations $\{\equiv_{\mathsf{A}}\}_{\mathsf{A} \in \mathsf{A}}$ indexed by types, which we denote as $\equiv$. Context equivalence is easily proved to be a congruence, i.e., a compatible equivalence relation.

3.1 Trace Equivalence

In this section we introduce a notion of *trace equivalence* for RSLR, and we show that it characterizes context equivalence.

We define a *trace* as a sequence of actions $l_1 \cdot l_2 \cdot \ldots \cdot l_n$ such that $l_i \in \{\mathsf{pass}(v), \mathsf{view}(\underline{m}) \mid v \in \mathrm{V}, \underline{m} \in \mathrm{V}^{\mathsf{Str}}\}$. Traces are indicated with metavariables like T, S. The *compatibility* of a trace T with a type A is defined inductively on the structure of A. If $\mathsf{A} = \mathsf{Str}$ then the only trace compatible with A is $\mathsf{T} = \mathsf{view}(\underline{m})$, with $\underline{m} \in \mathrm{V}^{\mathsf{Str}}$, otherwise, if $\mathsf{A} = \mathsf{aB} \to \mathsf{C}$ then traces compatible with A are in the form $\mathsf{T} = \mathsf{pass}(v) \cdot \mathsf{S}$ with $v \in \mathrm{V}^{\mathsf{B}}$ and S is itself compatible with C. With a slight abuse of notation, we often assume traces to be compatible with the underlying type.

Due to the probabilistic nature of our calculus, it is convenient to work with *term distributions*, i.e., distributions whose support is the set of closed terms of a certain type A, instead of plain terms. We denote term distributions with metavariables like $\mathcal{T}$ or $\mathcal{S}$. The effect traces have to distributions can be formalized by giving some binary relations:

- First of all, we need a binary relation on term distributions, called $\Rrightarrow$. Intuitively, $\mathcal{T} \Rrightarrow \mathcal{S}$ iff $\mathcal{T}$ evolves to $\mathcal{S}$ by performing internal moves, only. Furthermore, we use $\to$ to indicate a single internal move.
- We also need a binary relation $\Rightarrow^{\cdot}$ between term distributions, which is however labeled by a trace, and which models internal *and external* reduction.
- Finally, we need a labeled relation $\mapsto^{\cdot}$ between distributions and *real numbers*, which captures the probability that distributions accept traces.

The three relations are defined inductively by the rules in Figure 2. The following gives basic, easy, results about the relations we have introduced:

$$\frac{}{\mathcal{T} \Rightarrow^{\epsilon} \mathcal{T}} \qquad \frac{\mathcal{T} \Rightarrow^{\mathsf{S}} \{(\lambda x.t_i)^{p_i}\}}{\mathcal{T} \Rightarrow^{\mathsf{S}\cdot\mathsf{pass}(v)} \{(t_i\{v/x\})^{p_i}\}} \qquad \frac{\mathcal{T} \Rightarrow^{\mathsf{S}} \mathcal{S} \qquad \mathcal{S} \Rrightarrow \mathcal{U}}{\mathcal{T} \Rightarrow^{\mathsf{S}} \mathcal{U}}$$

$$\frac{\mathcal{T} \Rightarrow^{\mathsf{S}} \{(\underline{m_i})^{p_i}\}}{\mathcal{T} \mapsto^{\mathsf{S}\cdot\mathsf{view}(\underline{m})} \sum_{\underline{m_i}=\underline{m}} p_i} \qquad \frac{t \to \{(t_i)^{p_i}\}}{\mathcal{T} + \{(t)^{p}\} \Rrightarrow \mathcal{T} + \{(t_i)^{p\cdot p_i}\}}$$

Fig. 2. Term Distribution Small-Step Rules

Lemma 2. *Let $\mathcal{T}$ be a term distribution for the type A. Then, there is a unique value distribution $\mathcal{D}$ such that $\mathcal{T} \Rrightarrow^{*} \mathcal{D}$. As a consequence, for every trace T compatible for A there is a unique real number p such that $\mathcal{T} \mapsto^{\mathsf{T}} p$. This real number is denoted as* $\Pr(\mathcal{T}, \mathsf{T})$.

We are now ready to define what we mean by trace equivalence:

Definition 2. *Given two term distributions $\mathcal{T}, \mathcal{S}$ we say that they are* trace equivalent *(and we write $\mathcal{T} \simeq^{\mathsf{T}} \mathcal{S}$) if, for all traces T it holds that* $\Pr(\mathcal{T}, \mathsf{T}) = \Pr(\mathcal{S}, \mathsf{T})$. *In particular, then, two terms t, s are trace equivalent when $\{t^1\} \simeq^{\mathsf{T}} \{s^1\}$ and we write $t \simeq^{\mathsf{T}} s$ in that case.*

It is easy to prove that trace equivalence is an equivalence relation. The next step, then, is to prove that trace equivalence is compatible, thus paving the way to a proof of soundness w.r.t. context equivalence. Unfortunately, the direct proof of compatibility (i.e., an induction on the structure of contexts) simply does not work: the way the operational semantics is specified makes it impossible to track how a term behaves *in a context*. Following [7], we proceed by considering a refined semantics, defined not on terms but on pairs whose first component is a context and whose second component is a term distribution. Formally, a *context pair* has the form $(C, \mathcal{T})$, where C is a context and $\mathcal{T}$ is a term distribution. A *(context) pair distribution* is a distribution over context pairs. Such a pair distribution $\mathcal{P} = \{(C_i, \mathcal{T}_i)^{p_i}\}$ is said to be *normal* if for all i ad for all t in the support of $\mathcal{T}_i$ we have that $C_i[t]$ is a value. Similarly to terms, the internal and external evolution of traces can be defined by way of relations $\to$, $\Rrightarrow$ $\Rightarrow^{\cdot}$ and $\mapsto^{\cdot}$ (see [2] for more details).

The following tells us that working with context pairs is the same as working with terms as far as traces are concerned:

Lemma 3. *Suppose given a context C, a term distribution $\mathcal{T}$, and a trace S. Then if $(C, \mathcal{T}) \Rightarrow^{\mathsf{S}} \{(C_i, \mathcal{T}_i)^{p_i}\}$ then $C[\mathcal{T}] \Rightarrow^{\mathsf{S}} \{(C_i[\mathcal{T}_i])^{p_i}\}$. Moreover, if $(C, \mathcal{T}) \mapsto^{\mathsf{S}} p$, then* $\Pr(C[\mathcal{T}], \mathsf{S}) = p$.

But how could we exploit context pairs for our purposes? The key idea can be informally explained as follows: there is a notion of "relatedness" for pair distributions which not only is stricter than trace equivalence, but can be proved to be preserved along reduction, even when interaction with the environment is taken into account.

Definition 3 (Trace Relatedness). *Let $\mathcal{P}, \mathcal{Q}$ be two pair distributions. We say that they are* trace-related*, and we write $\mathcal{P} \triangledown \mathcal{Q}$ if there exist families $\{C_i\}_{i \in I}$, $\{\mathcal{T}_i\}_{i \in I}$, $\{\mathcal{S}_i\}_{i \in I}$, and $\{p_i\}_{i \in I}$ such that $\mathcal{P} = \{(C_i, \mathcal{T}_i)^{p_i}\}$, $\mathcal{Q} = \{(C_i, \mathcal{S}_i)^{p_i}\}$ and for every $i \in I$, it holds that $\mathcal{T}_i \simeq^{\mathsf{T}} \mathcal{S}_i$.*

Lemma 4 (Bisimulation, Externally). *Given two pair distributions $\mathcal{P}, \mathcal{Q}$ with $\mathcal{P} \triangledown \mathcal{Q}$, then for all traces S we have:*

1. *If $\mathcal{P} \Rightarrow^{\mathsf{S}} \mathcal{M}$, with $\mathcal{M}$ normal distribution, then $\mathcal{Q} \Rightarrow^{\mathsf{S}} \mathcal{N}$, where $\mathcal{M} \triangledown \mathcal{N}$ and $\mathcal{N}$ is a normal distribution too.*
2. *If $\mathcal{P} \mapsto^{\mathsf{S}} p$ then $\mathcal{Q} \mapsto^{\mathsf{S}} p$.*

We are now in a position to prove the main result of this section:

Theorem 2 (Full Abstraction). *Context equivalence and trace equivalence coincide.*

3.2 Some Words on Applicative Bisimulation

As we already discussed, the quantification over all contexts makes the task of proving two terms to be context equivalent burdensome, even if we restrict to linear contexts. And we cannot say that trace equivalence really overcomes this problem: there is a universal quantification anyway, even if contexts are replaced by objects (i.e. traces) having a simpler structure. It is thus natural to look for other techniques. The interactive view provided by traces suggests the possibility to go for coinductive techniques akin to Abramsky's applicative bisimulation, which has already been shown to be adaptable to probabilistic λ-calculi [4, 3].

One could define (see [2] for more details) a notion of applicative bisimulation for RSLR by giving a labeled Markov chain whose states are closed terms and whose labels are either eval, which models evaluation, $\mathsf{pass}(v)$, which models parameter passing, and $\mathsf{view}(\underline{m})$, which models testing for equality with the string $\underline{m}$. With some effort, one can prove that a greatest applicative bisimulation exists, and that it consists of the union (at any type) of all bisimulation relations. This is denoted as $\sim$ and said to be (applicative) *bisimilarity*. One can then generalize $\sim$ to a relation $\sim_\circ$ on open terms by the usual open extension (see [2] for more details).

One way to show that bisimilarity is included in context equivalence consists in proving that $\sim_\circ$ is a congruence; to reach this goal one can go through the *Howe's method* [13], which works well but requires some care (see [2] for more details). Is there any hope to get full abstraction? The answer is negative: applicative bisimilarity is too strong to match context equivalence. A counterexample to that can be built easily following the analogous one from [4].

This, however, is not the end of the story on coinductive methodologies for context equivalence in RSLR. A different route, suggested by trace equivalence, consists in taking the naturally definable (deterministic) labeled transition system of term *distributions* and ordinary bisimilarity over it. What one obtains this way is a precise characterization of context equivalence. There is a price to pay however, since one is forced to reason on distributions rather than terms. For more details, see [2].

4 From Equivalences to Metrics

The notion of observation on top of which context equivalence is defined is the probability of evaluating to the empty string, and is thus quantitative in nature. This suggests the possibility of generalizing context equivalence into a notion of *distance* between terms:

Definition 4 (Context Distance). *For every type A, we define $\delta^{\mathsf{C}}_{A} : \mathrm{T}^{A}_{\varnothing} \times \mathrm{T}^{A}_{\varnothing} \to \mathbb{R}_{[0,1]}$ as $\delta^{\mathsf{C}}_{A}(t,s) = \sup_{\vdash C[\vdash A]:\mathsf{Str}} |[\![C[t]]\!](\underline{\epsilon}) - [\![C[s]]\!](\underline{\epsilon})|$.*

For every type A, the function $\delta^{\mathsf{C}}_{\mathsf{A}}$ is a *pseudometric*[1] on the space of closed terms. Obviously, $\delta^{\mathsf{C}}_{\mathsf{A}}(t,s) = 0$ iff t and s are context equivalent. As such, then, the context distance can be seen as a natural generalization of context equivalence, where a real number between 0 and 1 is assigned to each pair of terms and is meant to be a measure of how different the two terms are in terms of their behavior. δ^{C} refers to the family $\{\delta^{\mathsf{C}}_{\mathsf{A}}\}_{\mathsf{A}\in\mathrm{A}}$.

One may wonder whether δ^{C}, as we have defined it, can somehow be characterized by a trace-based notion of metric, similarly to what have been done in Section 3 for equivalences. First of all, let us *define* such a distance. Actually, the very notion of a trace needs to be slightly modified: in the action $\mathsf{view}(\cdot)$, instead of observing a *single* string $\underline{m}$, we need to be able to observe the action on a *finite string set* M. The probability of accepting a trace in a term will be modified accordingly: $\Pr(t, \mathsf{view}(\mathrm{M})) = [\![t]\!](\mathrm{M})$.

Definition 5 (Trace Distance). *For every type A, we define $\delta^{\mathsf{T}}_{A} : \mathrm{T}^{A}_{\varnothing} \times \mathrm{T}^{A}_{\varnothing} \to \mathbb{R}_{[0,1]}$ as $\delta^{\mathsf{T}}_{A}(t,s) = \sup_{\mathsf{T}} |\Pr(t,\mathsf{T}) - \Pr(s,\mathsf{T})|$.*

It is easy to realize that if $t \simeq^{\mathsf{T}} s$ then $\delta^{\mathsf{T}}_{\mathsf{A}}(t,s) = 0$. Moreover, $\delta^{\mathsf{T}}_{\mathsf{A}}$ is itself a pseudometric. As usual, δ^{T} denotes the family $\{\delta^{\mathsf{T}}_{\mathsf{A}}\}_{\mathsf{A}\in\mathrm{A}}$.

But how should we proceed if we want to prove the two just introduced notions of distance to coincide? Could we proceed more or less like in Section 3.1? The answer is positive, but of course something can be found which plays the role of compatibility, since the latter is a property of *equivalences* and not of *metrics*. The way out is relatively simple: what corresponds to compatibility in metrics is non-expansiveness (see, e.g., [8]). A notion of distance δ is said to be *non-expansive* iff for every pair of terms t, s and for every context C, it holds that $\delta(C[t], C[s]) \leqslant \delta(t,s)$.

The proof of non-expansiveness for δ^{T} reflects the proof of compatibility for trace equivalence. What needs to be adapted, of course, is the notion of trace-relatedness, which should be made parametric on a real number ε, standing for the distance between the two context pair distributions at hand. Once we have non-expansiveness (see [2] for more details), full abstraction is within reach. As a corollary of non-expansiveness, one gets that:

Theorem 3 (Full Abstraction). *Context distance and trace distance coincide.*

[1] Following the literature on the subject, this stands for any function $\delta : A \times A \to \mathbb{R}$ such that $\delta(x,y) = \delta(y,x)$, $\delta(x,x) = 0$ and $\delta(x,y) + \delta(y,z) \geqslant \delta(x,z)$

One may wonder whether a coinductive notion of distance, sort of a metric analogue to applicative bisimilarity, can be defined. The answer is positive [8]. It however suffers from the same problems applicative bisimilarity has: in particular, it is not fully abstract.

5 Computational Indistinguishability

In this section we show how our notions of equivalence and distance relate to computational indistinguishability (CI in the following), a key notion in modern cryptography.

Definition 6. *Two distribution ensembles* $\{D_n\}_{n\in\mathbb{N}}$ *and* $\{E_n\}_{n\in\mathbb{N}}$ *(where both* D_n *and* E_n *are distributions on binary strings) are said to be* computationally indistinguishable *iff for every PPT algorithm* $\mathcal{A}$ *the following quantity is a negligible*[2] *function of* $n \in \mathbb{N}$*:* $|\Pr_{x\leftarrow D_n}(\mathcal{A}(x, 1^n) = \epsilon) - \Pr_{x\leftarrow E_n}(\mathcal{A}(x, 1^n) = \epsilon)|$.

It is a well-known fact in cryptography that in the definition above, $\mathcal{A}$ can be assumed to sample from x just *once* without altering the definition itself, provided the two involved ensembles are efficiently computable ([9], Theorem 3.2.6, page 108). This is in contrast to the case of arbitrary ensembles [10].

The careful reader should have already spotted the similarity between CI and the notion of context distance as given in Section 4. There are some key differences, though:

1. While context distance is an *absolute* notion of distance, CI depends on a parameter n, the so-called *security parameter*.
2. In computational indistinguishability, one can compare distributions over *strings*, while the context distance can evaluate how far terms of *arbitrary* types are.

The discrepancy Point 1 puts in evidence, however, can be easily overcome by turning the context distance into something slightly more parametric.

Definition 7 (Parametric Context Equivalence). *Given two terms* t, s *such that* $\vdash t, s : \mathsf{aStr} \to \mathsf{A}$*, we say that* t *and* s *are* parametrically context equivalent *iff for every context* C *such that* $\vdash C[\vdash \mathsf{A}] : \mathsf{Str}$ *we have that* $|[\![C[t1^{\mathbf{n}}]]\!](\underline{\epsilon}) - [\![C[s1^{\mathbf{n}}]]\!](\underline{\epsilon})|$ *is negligible in* $\mathbf{n}$*.*

This way, we have obtained a characterization of CI:

Theorem 4. *Let* t, s *be two terms of type* $\mathsf{aStr} \to \mathsf{Str}$*. Then* t, s *are parametric context equivalent iff the distribution ensembles* $\{[\![t1^{\mathbf{n}}]\!]\}_{\mathbf{n}\in\mathbb{N}}$ *and* $\{[\![s1^{\mathbf{n}}]\!]\}_{\mathbf{n}\in\mathbb{N}}$ *are computationally indistinguishable.*

Please observe that Theorem 4 only deals with terms of type $\mathsf{aStr} \to \mathsf{Str}$. The significance of parametric context equivalence when instantiated to terms of type $\mathsf{aStr} \to \mathsf{A}$, where A is a higher-order type, will be discussed in Section 5.1

[2] A negligible function is a function which tends to 0 faster than any inverse polynomial (see [9] for more details).

below. How could traces capture the peculiar way parametric context equivalence treats the security parameter? First of all, observe that, in Definition 7, the security parameter is passed to the term being tested *without* any intervention from the context. The most important difference, however, is that contexts are objects which test *families of terms* rather than terms. As a consequence, the action $\mathsf{view}(\cdot)$ does not take strings or finite sets of strings as arguments (as in equivalences or metrics), but rather *distinguishers*, namely closed RSLR terms of type $\mathsf{aStr} \to \mathsf{Str}$ that we denote with the metavariable $\mathtt{D}$. The probability that a term t of type Str satisfies one such action $\mathsf{view}(\mathtt{D})$ is $\sum_m [\![t]\!](\underline{m}) \cdot [\![\mathtt{D}\underline{m}]\!](\underline{\epsilon})$.

A trace T is said to be *parametrically compatible* for a type $\mathsf{aStr} \to \mathsf{A}$ if it is compatible for A. This is the starting point for the following definition:

Definition 8. *Two terms $t, s : \mathsf{aStr} \to \mathsf{A}$ are* parametrically trace equivalent*, and we write $t \simeq_n^{\mathsf{T}} s$, iff for every trace T which is parametrically compatible with A, there is a negligible function $negl : \mathbb{N} \to \mathbb{R}_{[0,1]}$ such that $|\Pr(t, \mathsf{pass}(1^{\mathbf{n}}) \cdot \mathsf{T}) - \Pr(s, \mathsf{pass}(1^{\mathbf{n}}) \cdot \mathsf{T})| \leqslant negl(\mathbf{n})$.*

The fact that parametric trace equivalence and parametric context equivalence are strongly related is quite intuitive: they are obtained by altering in a very similar way two notions which are already known to coincide (by Theorem 3). Indeed:

Theorem 5. *Parametric trace equivalence and parametric context equivalence coincide.*

The right-to-left inclusion is trivial, indeed every trace can be easily emulated by a context. The other one, as usual is more difficult, and requires a careful analysis of the behavior of terms depending on parameter, when put in a context. Overall, however, the structure of the proof is similar to the one we presented in Section 3.1 (see [2]).

5.1 Higher-Order Computational Indistinguishability?

Theorem 4 and Theorem 5 together tell us that two terms t, s of type $\mathsf{aStr} \to \mathsf{Str}$ are parametrically trace equivalent iff the distributions they denote are computationally indistinguishable. But what happens if the type of the two terms t, s is in the form $\mathsf{aStr} \to \mathsf{A}$ where A is an *higher-order* type? What do we obtain? Actually, the literature on cryptography does not offer a precise definition of "higher-order" computational indistinguishability, so a formal comparison with parametric context equivalence is not possible, yet.

Apparently, linear contexts do not capture equivalences as traditionally employed in cryptography, already when A is the first-order type $\mathsf{aStr} \to \mathsf{Str}$. A central concept in cryptography, indeed, is *pseudorandomness*, which can be spelled out for strings, giving rise to the concept of a pseudorandom generator, but also for functions, giving rise to pseudorandom functions [14]. Formally, a function $F : \{0,1\}^* \to \{0,1\}^* \to \{0,1\}^*$ is said to be a pseudorandom function iff $F(s)$ is a function which is indistinguishable from a random function from $\{0,1\}^n$

to $\{0,1\}^n$ whenever s is drawn at random from n-bit strings. Indistinguishability, again, is defined in terms of PPT algorithms having *oracle* access to $F(s)$. Now, having access to an oracle for a function is of course different than having *linear* access to it. Indeed, building a *linear* pseudorandom function is very easy: $G(s)$ is defined to be the function which returns s independently on the value of its input. G is of course not pseudorandom in the classical sense, since testing the function multiple times a distinguisher immediately sees the difference with a truly random function. On the other hand, the RSLR term t_G implementing the function G above is such that $\lambda x.t_G s$ is trace *equivalent* to a term r where:

- s is a term which produces in output $|x|$ bits drawn at random;
- r is the term $\lambda x.q$ of type $\mathsf{aStr} \to \mathsf{bStr} \to \mathsf{Str}$ such that q returns a random function from $|x|$-bitstrings to $|x|$-bitstrings. Strictly speaking, r cannot be an RSLR term, but it can anyway be used as an idealized construction.

But this is not the end of the story. Sometime, enforcing linear access to primitives is necessary. Consider, as an example, the two terms

$$t = \lambda n.(\lambda k.\lambda x.\lambda y.ENC(x,k))\,GEN(n);$$
$$s = \lambda n.(\lambda k.\lambda x.\lambda y.ENC(y,k))\,GEN(n);$$

where ENC is meant to be an encryption function and GEN is a function generating a random key. t and s should be considered equivalent whenever ENC is a secure cryptoscheme. But if ENC is secure against passive attacks (but not against active attacks), the two terms can possibly be distinguished with high probability if copying is available. The two terms can indeed be proved to be parametrically context equivalent if ENC is the cryptoscheme induced by a pseudorandom generator (see [2] for a proof and for more details).

Summing up, parametrized context equivalence coincides with CI when instantiated on base types, has some interest also on higher-order types, but is different from the kind of equivalences cryptographers use when dealing with higher-order objects (e.g. when defining pseudorandom functions). This discrepancy is mainly due to the linearity of the contexts we consider here. It seems however very hard to overcome it by just considering arbitrary nonlinear contexts instead of linear ones. Indeed, it would be hard to encode any *arbitrary* PPT distinguisher accessing an oracle by an RSLR context: those adversaries are only required to be PPT for oracles implementing certain kinds of functions (e.g. n-bits to n-bits, as in the case of pseudorandomness), while filling a RSLR context with any PPT algorithm is guaranteed to result in a PPT algorithm. This is anyway a very interesting problem, which is outside the scope of this paper, and that we are currently investigating in the context of a different, more expressive, probabilistic λ-calculus.

6 Conclusions

We believe that the main contribution of this work is the new light it sheds on the relations between computational indistinguishability, linear contexts and traces.

In particular, this approach, which is implicitly used in the literature on the subject [17, 16], is shown to have some limitations, but also to suggest a notion of higher-order indistinguishability which could possibly be an object of study in itself. This is indeed the main direction for future work we foresee.

References

1. Samson Abramsky. The lazy lambda calculus. In D. Turner, editor, *Research Topics in Functional Programming*, pages 65–117. Addison Wesley, 1990.
2. Alberto Cappai and Ugo Dal Lago. On equivalences, metrics, and polynomial time (long version). `http://arxiv.org/abs/1506.03710`, 2015.
3. 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.
4. Ugo Dal Lago, Davide Sangiorgi, and Michele Alberti. On coinductive equivalences for higher-order probabilistic functional programs. In *POPL*, 2014.
5. Ugo Dal Lago and Paolo Parisen Toldin. A higher-order characterization of probabilistic polynomial time. In *FOPARA*, volume 7177 of *LNCS*, pages 1–18. Springer, 2011.
6. Ugo Dal Lago, Sara Zuppiroli, and Maurizio Gabbrielli. Probabilistic recursion theory and implicit computational complexity. *Sci. Ann. Comp. Sci.*, 24(2):177–216, 2014.
7. Yuxin Deng and Yu Zhang. Program equivalence in linear contexts. *CoRR*, abs/1106.2872, 2011.
8. Josee Desharnais, Vineet Gupta, Radha Jagadeesan, and Prakash Panangaden. Metrics for labeled markov systems. In *CONCUR*, volume 1664 of *LNCS*, pages 258–273, 1999.
9. Oded Goldreich. *The Foundations of Cryptography - Volume 1, Basic Techniques*. Cambridge University Press, 2001.
10. Oded Goldreich and Madhu Sudan. Computational indistinguishability: A sample hierarchy. In *CCC*, pages 24–33, 1998.
11. Martin Hofmann. A mixed modal/linear lambda calculus with applications to bellantoni-cook safe recursion. In *CSL*, volume 1414 of *LNCS*, pages 275–294, 1997.
12. Martin Hofmann. Safe recursion with higher types and bck-algebra. *Ann. Pure Appl. Logic*, 104(1-3):113–166, 2000.
13. Douglas J. Howe. Proving congruence of bisimulation in functional programming languages. *Inf. Comput.*, 124(2):103–112, 1996.
14. Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman and Hall/CRC Press, 2007.
15. John C. Mitchell, Mark Mitchell, and Andre Scedrov. A linguistic characterization of bounded oracle computation and probabilistic polynomial time. In *FOCS*, pages 725–733, 1998.
16. David Nowak and Yu Zhang. A calculus for game-based security proofs. In *PROVSEC*, volume 6402 of *LNCS*, pages 35–52, 2010.
17. Yu Zhang. The computational SLR: a logic for reasoning about computational indistinguishability. *Mathematical Structures in Computer Science*, 20(5):951–975, 2010.