Alma Mater Studiorum - Università di Bologna
DOTTORATO DI RICERCA IN:
Informatica
Ciclo: XXVIII

Settore Concorsuale di Afferenza: 01/B1
Settore Scientifico Disciplinare: INF01

On Equivalences, Metrics, and Computational Indistinguishability

Presentata da: Alberto Cappai

Coordinatore Dottorato:
Paolo Ciaccia

Relatore:
Ugo Dal Lago

Esame Finale Anno 2016

Abstract

The continuous technological progress and the constant growing of information flow we observe every day brought us an urgent need to find a way to defend our data from malicious intruders; cryptography is the field of computer science that deals with security and studies techniques to protect communications from third parties, but in the recent years there has been a crisis in proving the security of cryptographic protocols, due to the exponential increase in the complexity of modeling proofs.

In this scenario we study interactions in a typed λ-calculus properly defined to fit well into the key aspects of a cryptographic proof: interaction, complexity and probability. This calculus, RSLR, is an extension of Hofmann's SLR for probabilistic polynomial time computations and it is perfect to model cryptographic primitives and adversaries. In particular, we characterize notions of context equivalence and context metrics, when defined on linear contexts, by way of traces, making proofs easier. Furthermore we show how to use this techniqe to obtain a proof methodology for computational indistinguishability, a key notion in modern cryptography; finally we give some motivating examples of concrete cryptographic schemes.

Contents

List of Figures

Chapter 1

Introduction

Nowadays we live in a world in which we observe a continuous increase of information flowing around us; indeed today we can find a PC in every house, and there often happens that all members of the family own their personal laptops; furthermore the use of smartphones allows us to remain connected to the web everywhere and every time we want. Technological progress allows us to do lots of activities simply by connecting to the web; by using our PC or smartphone we can get information, talk with friends, connect ourself to other servers, make financial transactions, do home banking and so on.

Unfortunately, such a great abundance of possibilities has negative consequences, indeed the more we send information through the network, the more we show vulnerabilities to potential malicious agents. Much of the information that runs through the web are quite considerable, such as identities, passwords, financial transactions, positions and so on; a contingent theft of information might have very bad consequences, so the question that rises is: "How can we protect our personal data from intruders?". There will always be the possibility to avoid to do these considerable activities, but that's not what we are looking for; our purpose is to obtain secure methods that we can trust, in order to do all our activities safely.

1.1 Cryptography: the instrument to make communications secure

Cryptography is the field of computer science which deals with security and studies techniques to guarantee secure communications in presence of third parties and then, it is the instrument we use to achieve our purpose. The main goal of cryptography is to allow a secure communication between two or more participants over an insecure channel; the security of a communication is defined depending on what we want to ensure about the information we want to protect: *secrecy, integrity, authentication, non-repudiation, privacy* and so on. This goal is achieved by using the so called *cryptographic algorithms* and *protocols*.

Cryptographic algorithms and cryptographic protocols are sequences of instructions for one or more participants; these instructions have to be performed sequentially, a new step starts if and only if the previous one is concluded and obviously the instructions of a protocol should not be ambiguous. For example, a protocol that guarantees the secrecy of a confidential information is usually defined by giving a triple $(\mathsf{GEN}, \mathsf{ENC}, \mathsf{DEC})$ where $\mathsf{GEN}, \mathsf{ENC}, \mathsf{DEC}$ are cryptographic algorithms such that:

- GEN is used to *generate* the keys that will be used by the other two algorithms to protect information.

- ENC is used to *encrypt* the confidential information by using the keys generated by GEN so that they become unaccessible for a potential intruder which is not in possession of the keys. These kind of algorithms take in input a key (the encryption key) and a message and return a cyphertext.

- DEC is the algorithm used to *decrypt* the cyphertext generated by using the encryption algorithm. These algorithms take in input a cyphertext and a key

(the decryption key) and return a plaintext.

The algorithms of a protocol, such as $\mathsf{GEN}, \mathsf{ENC}, \mathsf{DEC}$, are called *cryptographic primitives*; if the encryption key is the same as the decryption key we call the algorithm *symmetric*, otherwise *asymmetric*.

The continuous need for security guarantees we were talking about previously led to a significant increase in the complexity of cryptographic protocols design; but how can we say that a protocol satisfies the security property we are interested in? How can we say that a protocol is *secure*? A common way used long ago was to propose a new cryptographic scheme to the community and wait for some vulnerabilities to be found. If someone found a potential attack to the scheme, the same had to be corrected and reanalyzed; if none was able to find errors in the scheme it could be developed and certified as secure.

Nowadays such a way of certification can't be considered trustworthy; indeed, we have lots of examples of cryptographic protocols which have been broken after several years (For instance the Chor-Rivest cryptosystem was broken after 10 years). So, what we really need is a way to analyze the cryptanalysis itself in addition to the protocol; we need the security proof to be certifiable, we need for each protocol a mathematical proof that all the possible adversaries are not able to break it.

The analysis of a cryptographic protocol can be done by two different points of view: *formal* and *computational*. The formal point of view, also called the Dolev-Yao model [20], assumes the perfect security of the primitives which are used in the protocol, so every primitive is seen as a blackbox which works only if the agent owns the required information. Messages and keys are seen as atomic elements that can be combined during the protocol, so the purpose of a formal analysis is to study the protocol and find all the possible bugs that could advantage a malicious intruder. The computational way, on the other hand, sees primitives as functions from bitstrings to bitstrings and also messages and keys are seen as concatenations or lists of bits; so the analysis of a protocol from the computational point of view

may result more correct, due to the fact that we don't impose any condition on the primitives, but in practice more complex and error prone.

The complexity of the computational analysis makes the study of an entire protocol really difficult; the cryptographers use to analyze protocols from the formal point of view, whereas they prefer to use the computational point of view to analyze primitives. An attempt to bridge the gap between these two different approaches has been proposed by Abadi and Rogaway in 2000 [2]; their work is a first step towards a reconciliation of the computational and the formal analysis in which it is provided a computational justification for a formal treatment of an encryption. The idea is that, under some assumption on the formal expression and on the primitives used by an encryption scheme, it is possible to prove that the formal equivalence of two expressions corresponds to computational indistinguishability and so a formal proof is computationally sound. The difficulty in using this approach is that the assumptions needed on the primitives of the encryption scheme are not trivial.
In this thesis we will focus our attention on the computational analysis.

1.2 The structure of security proofs

One of the most used methods to build a security proof from the computational point of view can be described by four steps:

1. Define the security property we are going to prove.

2. Define a realistic model of a potential adversary.

3. Present the cryptographic protocol we are going to study.

4. Reduce the security property of the protocol to a particular assumption.

This list of actions lead us to the setting of our proof, indeed our path will be directed to prove the following statement: *"If there exists an adversary which can break the cryptosystem, then the same adversary is able to break the assumption with little expedients"*.

For example we choose *secrecy* as the security property we want to obtain from a cryptographic scheme and we define the adversary as a probabilistic polynomial-time algorithm (this is the most common definition); then we opt for the RSA protocol and we try to reduce the secrecy property of the RSA protocol to a particular well-known assumption, in this case the factorization of a large integers. It is possible to prove that if there exists a PPT algorithm $\mathcal{A}$ that can decrypt a message encrypted by the RSA protocol then the same algorithm can be used efficiently to build a PPT algorithm $\mathcal{A}'$ that factorize large integers; but, since we know that factorization is a hard problem, the consequence is that the algorithm $\mathcal{A}$ can't exists and so the RSA protocol is secure from the secrecy point of view.

This proof is called *by reduction* and by this way we can establish that if the assumption holds we have that the cryptosystem under control is secure, i.e. the security property is proved. One of the most common ways to prove a security property of a scheme by reduction is to use the game-based proof; such demonstrations are buildt as a sequence of games or experiments $\mathsf{G}_0, \mathsf{G}_1, ..., \mathsf{G}_n$. We will discuss about it in Chapter 2.

In [13] Bellare and Rogaway express a concept clear and free from doubt: "Many proofs in cryptography have become essentially unverifiable. Our field may be approaching a crisis of rigor". As we said previously, with the passing of time, we notice that cryptographic protocols have become more complex and it often happens that the security proofs given by hand are error-prone, so we need new instruments to face this situation. Unfortunately, we assist to a scenario in which the cryptographers community has not chosen a common path to expand; actually, we could say that at the moment the cryptographers research is growing horizontally, that is

proposing always different methods and approaches to this crisis of rigor in proving cryptoschemes. We will observe some of these approaches, in Chapter 2.

As we said previously, a proof structured as a game sequence is one of the most used standard to guarantee the security property of cryptographic constructions, so we decided to remain on this path and to examine in depth the framework of a game-based proof. A critical point in a game-based proof is the choice of the calculus used to model the whole system, indeed we are looking for a language that allows us to describe cryptographic primitives but also to model all the possible and feasible adversaries that try to break the cryptosystem.

When we deal with cryptography there are three fundamental aspects that we have to take into account:

- Interaction.
- Complexity.
- Probability.

Interaction is a key aspect, an adversary must be able to interact with the cryptographic protocol: he must have the possibility to pass arguments (messages, keys and so on) and to observe all the output of the primitives used; this is formalized by saying that the adversary has the complete control of the network and it is necessary to simulate a realistic scene, where we are not able to guarantee the security of the channel used for communications.

Complexity is another key aspect in a security proof. It is very easy to observe that an adversary with unbounded resources is able to break almost all cryptographic schemes, it is so necessary to reduce the possibilities of an adversary in order to face a realistic situation. The cryptographers assumption follows Cobham's thesis [16], that is the reduction of the feasible adversaries to the algorithms computable in polynomial time.

Finally we need to take care of *Probability*. Cryptosystems always deal with probabilistic primitives, because it is necessary to guarantee the right amount of randomness in the outputs of the protocols (it is possible to prove that a deterministic protocol is not semantically secure), furthermore adversaries are modeled as probabilistic algorithms so we need a calculus that allows us to work with probabilistic constructions. Moreover, when we propose a game we have a probabilistic algorithm that plays against a probabilistic scheme, so in the moment we want to say if the adversary wins or not the game in most cases we will have a probabilistic answer and then we need a calculus that allows us to reason about this probability and to evaluate it.

Summing up we are looking for a calculus that allows us to model the *interaction between probabilistic polynomial-time* bounded (PPT in the following) programs.

1.3 Implicit Complexity

We decide to start from the λ-calculus, a formal system defined by Church to analyze functions. The main features of λ-calculus are simplicity in description and expressiveness, it is very useful to model programs and to study their evolutions and behavior. Furthermore typed λ-calculus allows a certain degree of interaction by the definition of higher-order types and, as we have seen, this is a key point when we deal with cryptography.

Traditionally, complexity and probability are not aspects of λ-calculus, but there have been recent progresses that fix these lacks. In the field of the polytime calculus, Hofmann proposed a simply-typed lambda calculus called SLR (Safe Linear Recursion), which generalizes the characterization of the polytime functions studied by Bellantoni and Cook to higher order [12]. Hofmann improve Cobham's characterization of polytime functions applying the work of Bellantoni and Cook which

allows the use of recursion by a separation of variables into "safe" and "normal" ones [26]. This work offered the basis for the creation of several calculi applied to cryptography and game-based proofs, indeed there were proposed many extension of the syntax of SLR by adding probabilistic primitives, in order to obtain the features required to deal with cryptosystems.

One of the first extension was OSLR, which extend SLR with a 0-1 valued oracle, unfortunately it resulted difficult to build a logic upon the language [35]. Years later a new extension was proposed by Zhang, CSLR; the most significant feature of CSLR is the distinction at the type level between deterministic and probabilistic computations [38]. Zhang used CSLR to define a proof system used to justify computational indistinguishability in a direct way. This proof system may look similar to the one proposed by Impagliazzo and Kapron, that introduced two logical systems for reasoning about cryptographic construction [28]; the first logic is based on a non-standard arithmetic model and is proved to capture probabilistic polynomial-time computations, whereas the second one is focused on computational indistinguishability and is used to prove the unpredictability of the pseudorandom generator defined by Goldwasser and Micali [24]. Unfortunately, while the first system can be considered quite wide and complete, the second results not precisely defined and Zhang showed imprecisions in the proof of soundness [38].

An additional extension of CSLR has been developed in 2010, CSLR+; this extension allows for superpolynomial-time computations and also arbitrary uniform choices [37]. This thesis is about the RSLR calculus, another SLR extension presented by Dal Lago and Parisen Toldin for probabilistic polynomial-time computation, that I will discuss in Chapter 3 [31].

In order to develop the game-based technique and to simplify the automation, we need to analyze and implement new procedures that offer improvements without loss of mathematical guarantees. The main contribution of this thesis will be a characterization of *Computational Indistinguishability (CI)*, a key concept in cryp-

tography, in order to simplify the structure of a proof in a way that can be easily automated.

1.4 My contribution: a characterization of CI

As already mentioned, indistinguishability plays a central role in cryptographic proofs, but, what does it mean for two games (or more generally two programs) to be computationally indistinguishable? Briefly speaking, we consider two programs computationally indistinguishable if, for every Probabilistic Polynomial Time algorithm $\mathcal{A}$, the probability of $\mathcal{A}$ to distinguish between them is negligible, that means that it can't distinguish between them.

It is easy to see that the difficulty to establish if two programs are indistinguishable or not is focused on the quantification of all possible algorithms; the purpose of this thesis is to give a characterization of computational indistinguishability that allows to say if two programs are indistinguishable or not in a easier way, by using traces instead of arbitrary algorithms.

In order to get this result we will start from Chapter 4 by defining an equivalence relation based on contexts, terms of RSLR with a hole, that in our system take the place of PPT algorithms and stand for the feasible adversaries; then we will propose another equivalence relation based on traces: traces are elements with a structure simpler than contexts and we will prove that, in our framework, these two equivalence relations coincide. Furthermore we will propose another approach to prove equivalence between RSLR terms based on coinduction. Such an approach will be proved to be sound w.r.t. context equivalence but not complete; it is anyway interesting to observe this different approach because it allows us to work without universal quantifications and so in a very simple way.

Next step, in Chapter 5, will be a generalization of the techniques showed in Chapter 4, by the definition of two different notions of distance, the former based

on contexts and the latter on traces; we will show once again that the two definitions are equivalent, but we need to make a small change in the definition of traces.

Finally, in Chapter 6, we will propose a parametric version of context equivalence and we will show that when we compare base terms (i.e. strings), the parametric context equivalence is equivalent to computational indistinguishability as normally used by cryptographers; as in the previous cases we will give a definition of parametric equivalence based on traces and we will prove that it coincides with the parametric context equivalence. We will conclude this thesis by showing some motivating examples from cryptographic primitives and protocols (Chapter 7).

Chapter 2

Some Approaches to Computational Cryptography

In the Introduction we talked about the difficulties and the crisis in proving the security of a cryptographic scheme due to the exponential growing in the complexity of the proofs. In order to fix this situation several approaches have been proposed to help cryptographers to build cryptographic proofs easily: we will start talking about formal methods that under particular conditions are computationally sound, then we will talk about automated tools and finally we will describe some methods based on process calculi and CI.

2.1 Computational Soundness of Formal Proof

Previously, we introduced two different approaches to study the properties of a security protocol: formal and computational. In the formal approach, also called symbolic, messages and keys are seen as atomic elements and we assume the perfect security of the primitives used in a protocol, whereas in the computational one messages and keys are bitstring and primitives are functions from bitstrings to bitstrings; the formal analysis is simpler and easily mechanizable, the computational one is more precise and therefore more complex, even a small protocol could need a

very complicated proof.

These two approaches evolved separately for almost twenty years, but in 2000, Abadi and Rogaway [2] opened a window onto the possibility to merge the two different approaches, in order to give formal proofs that are computationally sound.

2.1.1 Reconciling Two Views of Cryptography

The goal of Abadi and Rogaway is to call attention to the gap between the computational and the formal points of view and to start to bridge this gap; the main theorem of their work states that if a symbolic notion of equivalence is proved in the formal framework, then also the equivalent notion in the computational system is proved.

The work starts with the description of the formal system and the definition of the set of expressions that will be used:

$$M, N, ... ::= 0 \mid 1 \mid (M, M) \mid K \mid \{M\}_K$$

where $0, 1 \in \mathbf{Bool}$ and $K \in \mathbf{Keys}$ with **Keys** fixed non empty set disjoint from **Bool**. The expression (M, N) is the pairing of two expressions and $\{M\}_K$ is the encryption of M under K; it is important to notice that in this framework we work only with symmetric encryptions.

The next step is a formal definition of equivalence: this definition starts from the entailment relation $M \vdash N$ that intuitively means that N can be deduced from M, and so it is a way to represent what an adversary can deduce from an expression M. For example we have $((\{M\}_K, \{N\}_{K'}), K) \vdash K$ and $((\{M\}_K, \{N\}_{K'}), K) \vdash M$, but $((\{M\}_K, \{N\}_{K'}), K) \nvdash N$.

Once defined the relation $\vdash$, two expressions M, N are defined formally equivalent, written $M \equiv N$, if $pattern(M)$ and $pattern(N)$ are equal. Patterns are extensions of expressions with the add of the symbol $\square$, that represents an expres-

sion undecryptable by an adversary, so a pattern is an expression with some parts that the adversary can't see.

Given a set of keys T and a pattern M we define $p(M,T)$ inductively as follows:

$$\begin{aligned} p(K,T) &= K \\ p(i,T) &= i, \qquad\qquad i \in \mathbf{Bool} \\ p((M,N),T) &= (p(M,T), p(N,T)) \\ p(\{M\}_K,T) &= \{p(M,T)\}_K \text{ if } K \in T, \\ &\quad \square \text{ otherwise.} \end{aligned}$$

Intuitively, $p(M,T)$ is the pattern that an attacker can see by using the keys included in T. The expression $pattern(M)$ stands for $p(M,T)$, where T is the set of keys that can be deduced from M itself.

When Abadi and Rogaway move to the computational point of view, they consider a symmetric encryption scheme $\Pi = (\mathsf{ENC}, \mathsf{DEC}, \mathsf{GEN})$ that is *type-0* secure. The type-0 security is a very strong assumption. Intuitively it means that the encryption hides all the information about the plaintext and the encryption key used. Another important request in this framework is to work only with *acyclic* expressions. An expression is acyclic if it doesn't admit encryption cycles. We have an encryption cycle when the relation between keys K_1 *encrypts* K_2 (that is K_1 encrypts an expression M that contains K_2) is a cyclic relation, for instance $\{K\}_K$ or $(\{K_1\}_{K_2}, \{K_2\}_{K_1})$; this possibility often lead to a weakness of the scheme and so it is denied in Abadi and Rogaway framework.

At this point, given a formal expression M, we associate to M an ensemble $[\![M]\!]_\Pi$, so that we are able to reason on it from a computational point of view: each key symbol K is mapped to a string $\tau(K)$ by using the key generator GEN, the formal bits 0,1 are mapped to their computational representation, a pair (M,N) is encoded by concatenating the images of M and N, $\mathsf{ENC}(\tau(K), M)$ is the computational expression associated to $\{M\}_K$ and each string is tagged with its type (key, bool,

pair, cyphertext) in order to avoid ambiguities.

So, once defined how associate a formal expression M to an ensemble $[\![M]\!]_\Pi$ the main result of [2] is given by the following theorem:

Theorem 2.1 *Let M, N be two acyclic expressions and let Π be a type-0 secure symmetric encryption scheme. Suppose that $M \equiv N$ then $[\![M]\!]_\Pi \approx [\![N]\!]_\Pi$.*

This result links the symbolic equivalence relation defined on patterns to the computational indistinguishability, denoted by $\approx$, showing for the first time a sound symbolic abstraction of CI.

As we can easily see the work of Abadi and Rogaway is just a starting point and over time there have been several extensions of this framework, especially on two points: *logic* and *encryption*. The first logic extensions have been made by Micciancio and Warinschi, that proved by a counter-example that the logic proposed by Abadi and Rogaway was not complete; indeed it could be possible to have cases of false negative, i.e. $M \not\equiv N$, but $[\![M]\!]_\Pi \approx [\![N]\!]_\Pi$.

In order to get completeness, Micciancio and Warischi showed that it is sufficient to request that the encryption scheme is *authenticated*, that means that an adversary is not able to produce valid cyphertexts [34]. A refinement of this completeness result has been proposed by Gligor and Horvitz, that proposed a new security definition both sufficient and necessary, *weak key-authenticity test for expressions* (WKA-EXP) [27].

In [25] Herzog extends Abadi and Rogaway's work to a framework independent of the encryption scheme chosen, in particular he extends the result also to asymmetric encryption schemes that satisfy the (IND-CCA2) security property, a case in which the adversary is given access to a *decryption oracle* that can decrypt arbitrary adversary's requests. In [33] the result is extended to a system in which keys are not elements of a set disjoint form the message set, but are arbitrary expressions. Both of these extensions continue to present the problem of cyclic expressions; this is due to

the fact that in formal frameworks key cycles are not considered a threat, whereas in a computational model the presence of a key cycles can invalidate standard security properties.

Two different way to handle key cycles have been proposed, the first one based on giving to the adversary more power, in order to make it able to deduce a key when it is inside a key cycle [32], the second one based on giving a new security property, called *key-dependant message* (KDM), strictly stronger than (IND-CCA2) and sufficient to ensure soundness even in presence of key cycles [3].

Other extensions can be found in [4], where Adão et al. study *which-key* and *length-key* encryption schemes, [21] where the logic is extended with hash functions and [15], where the logic is extended with *modular exponentiation*.

2.1.2 Static Equivalence Soundness

A different approach to computational soundness of symbolic methods is proposed by Baudet, Cortier and Kremer in [10, 11]; this approach is more general, because it is independent from the set of primitives chosen in the protocol. The idea is to express the symbolic secrecy by using the *static equivalence* instead of patterns. How does this approach work?

- The first step is the definition of an *abstract algebra*, a term algebra defined on a first-order signature with sorts and equipped with an equational theory; for instance, one of the simplest example of equational theory is E_{ENC}, the equational theory of a symmetric encryption system, generated by the rule: $\mathsf{DEC}(\mathsf{ENC}(m,k),k) =_{E_{\mathsf{ENC}}} m$.

- The second step is the definition of two different equivalence relations based on *deducibility*, $\vdash_E$, and *static equivalence*, $\approx_E$, in order to catch the capability of a symbolic adversary to distinguish between terms. Deducibility is used to define the terms that can be evaluated by an adversary from a sequence of

terms and static equivalence to state if two sequence of terms are equivalent or not.

- At this point a method is proposed to turn the abstract algebra into a concrete computational algebra in order to reason about the relations between symbolic and computational model.

- Finally, they start from the equational theory $=_E$ and from the hypothesis that $=_E$ is a reasonable abstraction of the primitives, that means that it is sound and faithful w.r.t. computational indistinguishability (faithfulness is a stronger version of completeness); the main contribution of this work is the proof that also deducibility and static equivalence are two sound and faithful equivalence relations w.r.t. the computational algebra. Furthermore it is showed that for many equational theories $\approx_E$ soundness is a sufficient criterion for all the other notions of faithfulness and soundness.

This framework has been used by Abadi, Baudet and Warinschi in [1] for applications on offline guessing attacks.

Another definition of formal indistinguishability has been proposed by Bana, Mohassel and Stegers, that found out that static equivalence is a definition too rough to reason about many equational theories; for instance they proved that static equivalence is not sound when working with a framework that uses modular exponentiation [7].

2.1.3 Unconditional soundness

Differently from the approaches we talked about hitherto, the framework proposed by Bana and Comon-Lundh in [5, 6] starts from a different concept: they are not interested in defining symbolic properties or adding computational constraints to obtain soundness anymore, but the goal is to define a model in which an adversary is able to perform any action that does not contradict the computational assumptions.

This purpose is reached by giving a list of axioms that reproduces the computational properties of the protocol under study (for instance IND-CCA); once given the axioms, we can consider the greatest symbolic model that satisfies this list and reason formally about its security properties. This way we can obtain the computational soundness almost by definition, because the computational constraints are included in the axioms and so a computational attacker can be modeled symbolically.

The main feature of this framework is to reduce the security of protocols to an *inconsistency* proof for a set of first order formulas: *"If the negation of the security formula is inconsistent with the set of axioms, then the protocol is secure in any model of the axioms "* [6]. This approach lead to several advantages, for instance:

- The proofs are built in a symbolic setting, so they are simpler and, if possible, automated.
- It is very simple to add cryptographic primitives, because it is only necessary to write the corresponding axiomatization.
- It is possible to prove the security properties by using weaker assumption, simply using a weaker axiomatization.
- The use of an axiomatization makes the assumptions needed to ensure security properties very clear.
- In order to strengthen the security assumptions, i.e. when an attack is found, it is sufficient to add an axiom so that we express stronger hypothesis on the computational implementation of the primitives.
- It is possible to handle many situations that previously had been discarded, such as key cycles, dynamic corruption and XOR.

However, this approach has been developed recently, so we don't have concrete applications of this method yet, but the authors are confident that inconstency checks could be implemented efficiently.

2.2 Automated Tools

A significant help in the construction of cryptographic proofs has come from the creation of tools that, given a security protocol try to elaborate (in a automatic or semi-automatic way) a proof that can be easily verified. In the following subsections I will talk about three different tools which are used to demonstrate the security of a protocol by using game-based proofs: CertyCrypt, EasyCrypt and Cryptoverif.

Game-based security proofs are built as a sequence $\mathsf{G}_0, \mathsf{G}_1, ..., \mathsf{G}_n$ of games or experiments. The first game G_0 encodes the interaction between a generic algorithm $\mathcal{A}$ that stands for the potential adversary and the last one G_n encodes the adversary that tries to break the assumption we choose. All the games are in the same probability space, successive games are very similar and for each game we can evaluate the probability of the adversary to win the game and we call it the *advantage*; so, by using this method we are able to put in relation the probability of an adversary to break the cryptosystem with the probability of an adversary to break the assumption. The game-based proof structure succeeds in combining the intuition of a game in describing a security protocol to the accuracy and the formalism of a mathematical demonstration.

2.2.1 CertiCrypt

CertiCrypt is a machine checked framework, which is used to construct cryptographic proofs structured as sequences of games; the pecularity of the CertyCrypt framework is that the proofs are built on top of the Coq proof assistant, thus we have the mathematical guarantee of a certified proof assistant and the possibility to study these proofs and their verification step by step [8]. Some of the features of this framework are:

- It is used an imperative programming language with probabilistic assignments,

structured data types and procedure calls. The choice of such a language is made in order to be closer to the cryptographer standards in game description.

- The framework doesn't return asymptotic results, but it focuses on exact security; this decision is due to the fact that in this way we can set our parameters to obtain concrete security bound.

- Every proof yields a proof object which can be checked automatically and separately.

- The framework is equipped with automated reasoning methods, it's formalized as relational Hoare logic and a theory of observational equivalence.

The probabilistic programming language follows this construction:

$\mathcal{C}$	::=	skip	nop
	|	$\mathcal{C}; \mathcal{C}$	sequence
	|	$\mathcal{V} \leftarrow \mathcal{E}$	assignment
	|	$\mathcal{V} \leftarrow^{\$} \mathcal{DE}$	random sampling
	|	if $\mathcal{E}$ then $\mathcal{C}$ else $\mathcal{C}$	conditional
	|	while $\mathcal{E}$ do $\mathcal{C}$	while loop
	|	$\mathcal{V} \leftarrow \mathcal{P}(\mathcal{E}, ..., \mathcal{E})$	procedure call

where $\mathcal{V}$ represents the set of variables, $\mathcal{E}$ the set of expressions, $\mathcal{DE}$ the set of distribution expression and $\mathcal{P}$ the set of procedures.

In the CertyCrypt framework we have that programs are seen as functions starting from an initial memory m to sub-probability distribution over final memories. To describe the semantics of programs a distribution that maps a [0,1]-valued random variable, i.e. a function in $A \to [0,1]$ to its expected value, is used. This function is defined of type $\mathcal{D}(A) = (A \to [0,1]) \to [0,1]$. Given a distribution $\mu \in \mathcal{D}$ and a function $f : A \to [0,1]$ we have that $\mu(f)$ represents the expected value of the

function f.

So we can consider a program as a function which maps an initial memory m to a distribution on final memories and write its semantics as:

$$[\![c \in \mathcal{C}]\!] : \mathcal{M} \to \mathcal{D}(\mathcal{M})$$

Where $\mathcal{M}$ is the set of memories.

When we study a security protocol we don't know how an adversary could reason and which strategies he could use to break the cryptosystem; the only way to model an adversary without loss of generality is giving him an interface and a set of rules which specify what he can or can't do. An adversary interface consists in a triple$(\mathcal{O}, \mathcal{RW}, \mathcal{R})$, where $\mathcal{O}$ represents the set of procedures he can call, $\mathcal{RW}$ represents the set of variables the adversary can read and write, $\mathcal{R}$ represents the set of variables the adversary can only read. The rules are generic, they are only given in order to be sure that the adversary makes a correct use of variables and procedures.

2.2.2 Game transitions in CertyCrypt Framework

The CertyCrypt framework follows Shoup classification, and divides transitions between games in a proof into three categories:

1. Transition which are based on indistinguishability.
2. Transition based on failure event.
3. Transition based on bridging steps.

All these transitions are justified by using a probabilistic Relational Hoare Logic (pRHL); such approach is useful because it generalizes observational equivalence and allows us to reason about probabilities of events into different games [8]. A pRHL judgment has the following structure:

Definition 1 *We define two programs c_1, c_2 equivalent with respect to pre-condition Ψ and post condition Φ iff:*

$$\vdash c_1 \sim c_2 \ : \Psi \Rightarrow \Phi \ \stackrel{def}{=} \ \forall \ m_1, m_2 . m_1 \Psi m_2 \Rightarrow ([\![c_1]\!] m_1) \Phi^{\#} ([\![c_2]\!] m_2)$$

This definition means that given two arbitrary initial memories m_1, m_2 that satisfy the pre-condition Ψ, i.e. $m_1 \Psi m_2$, we have that two programs c_1, c_2 are equivalent if their evaluation from the initial memories satisfies the post condition Φ, i.e. $([\![c_1]\!] m_1) \Phi^{\#} ([\![c_2]\!] m_2)$.

By using this kind of judgment we can derive the observational equivalence $\simeq$, a particular case of the equivalence defined above and a key property in game transitions; two programs c_1, c_2 are observationally equivalent if they can be proved equivalent by using pre- and post- conditions restricted to $=_I, =_O$, i.e. equalities over a subset of program variables (Input and Output). So we have:

$$\vdash c_1 \simeq_O^I c_2 \ \stackrel{def}{=} \ \vdash c_1 \sim c_2 : =_I \Rightarrow =_O$$

CertyCrypt takes advantage of the observation equivalence property because it results easily mechanizable.

CertyCrypt settles the case of a game transition which depends on what cryptographers call failure event, by using the following fundamental lemma. This case occurs when we have two games G_1, G_2, two events A, B and we face with a situation in which the probability that the event A occurs in G_1 is the same as the probability the event B occurs in G_2 unless the verification of an exact event F, which is called failure event. The fundamental lemma allows to bound the difference of probability of an event in two different games, it tells us that the difference between the probabilities that A occurs in G_1 and B occurs in G_2 is bounded by the greatest probability that F occurs in G_1 or G_2.

Theorem 2.2 (Fundamental Lemma) *Let G_1, G_2 be two games and A, B, F three events.*

If $Pr[G_1 : A \wedge \neg F] = Pr[G_2 : B \wedge \neg F]$, *then we have:*

$$|Pr[G_1 : A] - Pr[G_2 : B]| \leqslant max(Pr[G_1 : F], Pr[G_2 : F])$$

This theorem is important because it states that we can reduce the probability to observe difference between two programs to the probability of the occurence of a sigle event. Roughly speaking, we can say that a program that returns a cyphertext is indistinguishable from a program that returns a random message unless the adversary is able to figure out the key of the cryptographic scheme; so the probability for the adversary to distinguish between the two programs is equal to the probability to get the key.

Finally we have game transition based on bridging steps: it occurs that a fragment of code c_1 in a game G_1 is replaced by an observational equivalent fragment c_2 in a game G_2.
These substitutions are implemented by using techniques such a deadcode elimination, constant folding and propagation, procedure call inlining, swapping statement, common prefix/suffix elimination.

In conclusion, we have a fully automated verification tool based on Coq with an understandable semantics, which uses a clear set of techniques (that are proved to be sound) to obtain verifiable proofs of security property of cryptoprotocols. These tools needs only few Coq lines to establish the security statements and allows the user to study the proof step by step, without being an expert of Coq.

2.2.3 EasyCrypt

In the last years a new automated tool for elaborating security proofs of cryptoprotocols has been developed; indeed, despite the similarity between the CertyCrypt language and the usual cryptographers standards in describing games, the framework didn't achieve resounding success between the community. The reason of the lacking use by the community may be related to the fact that even if CertyCrypt

offers security proofs with high guarantees, building a machine-checked proof results hard and expertise necessary. In order to fill this gap, a new tool has been presented, which is supposed to be easier to use than his predecessors but trustworthy in the same way [9].

EasyCrypt is an automated tool which elaborates security proofs of cryptosystem from proof sketches, that are checked by using off-the-shelf SMT solvers and automated theorem provers. These proofs are given in form of games and the idea behind is the same as in CertyCrypt, that is to study and evaluate relations between game transitions such as:

$$Pr[G : A] \leqslant Pr[G' : A'] + \Delta$$

where G, G' are games, A, A' are events and Δ is a quantity which depends on the oracle calls made by an adversary.

The structure of EasyCrypt is similar to CertyCrypt, indeed the transitions between games are justified firstly by proving the logical relations using the probabilistic Relational Hoare Logic and then by applying information-theoretic reasoning to derive probability claims about the occurring of events. In order to increase the speed of the calculation EasyCrypt implements a procedure that produces a set of verification conditions that are sufficient to establish the validity of a certain judgment. This feature is a key point of the effectiveness of EasyCrypt, indeed a peculiarity of this tool is that the verification conditions are expressed in a first-order logic as follows:

$$\Psi, \Phi := b \mid \neg\Psi \mid \Psi \wedge \Phi \mid \Psi \vee \Phi \mid \Psi \rightarrow \Phi \mid \Psi \leftrightarrow \Phi \mid (\Phi) \mid \forall x.\Phi \mid \exists x.\Phi$$

This kind of expressions avoid the reasoning about probabilities and allow the use of SMT solvers and theorem provers to discharge automatically their validity; probabilities of events are evaluated by additional automated mechanism by using some

elementary rules. Furthermore the procedure that generates the verification condition also adds Coq files which can be checked separately (For instance by using CertyCrypt).

The strategy used to generate the set of verification conditions bases on the following points:

1. The procedures which aren't called by the adversary are canceled from games by inlining their definitions, so that only adversary calls remain.

2. The random assignments are moved upfront, so that the code is divided into two parts, the random and the deterministic one.

3. The deterministic part of the code is studied by a relational weakest precondition calculus by using relational specification to deal with the adversary calls.

4. A map f is used to generate the verification condition $\Psi \Rightarrow_f \Phi$, defined as:

$$\forall m_1, m_2\ t_1, ...t_l.\ m_1 \Psi m_2 \Rightarrow m_1\{\vec{t}/\vec{x}\}\ \Phi\ m_2\{f(t_1, ..., t_l)/\vec{y}\}$$

   The injectivity of the map f is generally a sufficient condition to guarantee that the validity of $\Psi \Rightarrow_f \Phi$ entails the validity of the corresponding pRHL judgment.

5. Off-the-shelf tools establish the validity of the first-order formula $\Psi \Rightarrow_f \Phi$. EasyCrypt generates its verification conditions in Why tool format and then uses the Simplify prover and the alt-ergo SMT solver to discharge the conditions.

As happens in CertyCrypt, a fundamental lemma is given in order to justify transition based on a failure event.

Lemma 2.1 ***(Fundamental Lemma):*** *Let G_1, G_2 be two games and A, B, F events such that:*

$$\models G_1 \sim G_2 : \Psi \Rightarrow (F\langle 1\rangle \leftrightarrow F\langle 2\rangle) \wedge (\neg F\langle 1\rangle \rightarrow (A\langle 1\rangle \leftrightarrow B\langle 2\rangle))$$

Then if $m_1 \Psi m_2$

1. $Pr[G_1, m_1 : A \wedge \neg F] = Pr[G_2, m_2 : B \wedge \neg F]$

2. $|Pr[G_1, m_1 : A] - Pr[G_2, m_2 : B]| \leqslant Pr[G_1, m_1 : F] = Pr[G_2, m_2 : F]$

EasyCrypt presents a limitation in the language, which lacks loops, recursive procedures and drawing from skewed distributions; furthermore it only generates partial verifiable evidences and, as said previously, EasyCrypt only generates proof skeletons for claims about probabilities rather than fully-machine checked proofs.

2.2.4 CryptoVerif

CryptoVerif is a computational sound mechanized prover for cryptographic protocols; it returns results about secrecy and correspondence properties of protocols and also provides generic methods for specifying properties of cryptographic primitives [14]. CryptoVerif works for a bounded number of sessions N, which is polynomial in the security parameter, in presence of active adversaries. As the precedent tools, CryptoVerif focus its goal in results of exact security, so it returns a bound of the probability of a successful attack against the protocol under study.

As seen previously, CryptoVerif gives proofs as sequences of games, so we have that the first game represents a real protocol, while the last stands for an ideal protocol, where the occurring of a security property or not results obvious. We can find the first difference from the two previous tools in the formalization of games, indeed CryptoVerif uses a process calculus inspired from the pi calculus (substantially it uses an extension of the pi calculus); the semantics is pure probabilistic, instead of non-deterministic, and we have an extension which allows us to work with arrays, that replaces lists in cryptographic proofs.

The transitions between games are made by using two techniques: the observational equivalence and the syntactic transformation. We say that two games are observationally equivalent, and we write $\mathsf{G} \approx \mathsf{G}'$ if every adversary has a negligible probability to distinguish them. We define in CryptoVerif an adversary as a context, and we have that $\mathsf{G} \approx \mathsf{G}' \Rightarrow C[\mathsf{G}] \approx C[\mathsf{G}']$ for all acceptable evaluation context. So our purpose is to obtain a sequence of this form:

$$\mathsf{G}_0 \approx \mathsf{G}_1 \approx \cdots \approx \mathsf{G}_m$$

that implies $\mathsf{G}_0 \approx \mathsf{G}_m$.

Transition based on observational equivalence are given as axioms and come from security properties of the primitives inside the protocol. On the other hand syntactic transformation are substantially simplifications of the code and expansions of the arguments: for instance there happens a renaming of variables which are assigned several times, a merge of several variables into a single array variable, a replace of variables with its value and so on.

A particular feature of CryptoVerif is the possibility of making a syntactic transformation manually: it is possible to insert manually an event or an instruction and also to replace a term with another term the the tool verifies to be really equal, before continuing the demonstration.

The results obtained by CryptoVerif are encouraging beyond any doubt, but the tools presents evident limitations; indeed it sometimes happens that the prover fails in proving a particular security property when it doesn't hold. Furthermore sometimes (In some public-key protocols) the tool stops and waits for manual instruction to continue the demonstration. In order to go beyond these limits there will be improved extension turned towards improvements in the proof strategy and handles of more equations.

Summing up, we recalled three different tools that return cryptographic proofs in the form of sequences of games; despite the giant steps made recently these tools have evident limitations, so it is necessary to find new methods that are easily mechanizable in order to enlarge the power of these tools and to extend the set of primitives and protocols that can be analyzed automatically.

2.3 Computational Indistinguishability, Logics, and Calculi

In this section we will talk about a kind of approach that is quite close to the one we expose in this thesis. The idea is to propose a method to reason about CI in a framework based on a process calculus suitably defined to handle cryptographic constructions and to model all and only feasible adversaries. The goal of this approach is to increase the means of cryptographers in building security proofs by showing new methods to prove computational indistinguishability.

2.3.1 A Process Calculus

In [36] Mitchell et al. studied properties of a process calculus designed to analyze security protocols; this calculus is a variant of CCS, where bounded replications and Probabilistic Polynomial-Time (PPT) expressions in messages are allowed. A feature of this calculus is that all the processes evaluate in polynomial time; this choice is obviously made to reason about security problems, where adversaries have a limited power of calculus. Moreover, we have that messages are scheduled probabilistically, rather than nondeterministically in order to avoid inconsistency between security and nondeterminism.

Expressions in this calculus are defined by the following grammar:

$$\mathcal{P} ::= \varnothing \mid \nu(c).\mathcal{P} \mid in(c,x).(\mathcal{P}) \mid out(c,\mathrm{T}).(\mathcal{P}) \mid [\mathrm{T}].(\mathcal{P}) \mid (\mathcal{P}|\mathcal{P}) \mid !_{q(\nu)}.(\mathcal{P})$$

where T is a term that could be a variable or the security parameter η or a random coin or a probabilistic polynomial time function θ of arity k, applied to k terms $\mathrm{T}_1, .., \mathrm{T}_k$.

$\varnothing$ is the empty process, $\nu(c).\mathcal{P}$ is the *channel binding* of the channel c in the process $\mathcal{P}$, $in(c,x).(\mathcal{P})$ is the *input expression*, it waits for an input from the channel c and then performs $\mathcal{P}$, $out(c,\mathrm{T}).(\mathcal{P})$ is the *output expression*, it reduces T in atoms and sends in output the results through the channel c, $[\mathrm{T}].(\mathcal{P})$ is the *match expression*, it proceeds with $\mathcal{P}$ if the guarding term T reduces to 1, $(\mathcal{P}|\mathcal{P})$ is the *parallel composition* and $!_{q(\nu)}.(\mathcal{P})$ is the *bounded replication* of $\mathcal{P}$. *Context expressions* are defined by the following grammar:

$$\mathcal{C}[\cdot] ::= \varnothing \mid in(c,x).(\mathcal{C}[\cdot]) \mid out(c,\mathrm{T}).(\mathcal{C}[\cdot]) \mid [\mathrm{T}].(\mathcal{C}[\cdot])$$

$$\mid (\mathcal{C}[\cdot]|\mathcal{P}) \mid (\mathcal{P}|\mathcal{C}[\cdot]) \mid !_{q(\nu)}.(\mathcal{C}[\cdot])$$

In this framework security properties are expressed by the use of observational equivalence. Two processes $\mathcal{P}, \mathcal{Q}$ are said to be observational equivalent, we write $\mathcal{P} \cong \mathcal{Q}$ if for all contexts $\mathcal{C}[\cdot]$ the behavior of $\mathcal{C}[\mathcal{P}]$ is asymptotically computationally indistinguishable from the behavior of $\mathcal{C}[\mathcal{Q}]$. This is formalized by the following definition:

Definition 2 *Let $\mathcal{P}, \mathcal{Q}$ be two process and let $\mathbf{Val}(\mathcal{P}, \mathcal{Q})$ be the set of all valuations of free variables of $\mathcal{P}$ and $\mathcal{Q}$. We say that $\mathcal{P} \simeq \mathcal{Q}$ if:*

$$\forall \xi \in \mathbf{Val}(\mathcal{P}, \mathcal{Q}).\ \forall \mathcal{C}[\cdot]\ :\ \xi(\mathcal{C}[\mathcal{P}]) \simeq \xi(\mathcal{C}[\mathcal{Q}])$$

The expression $\mathcal{P} \simeq \mathcal{Q}$ it is used to state that $\mathcal{P}, \mathcal{Q}$ are *asymptotically close*, it means that the probability for $\mathcal{P}$ and $\mathcal{Q}$ to generate a different observable is negligible in the security parameter.

The observational equivalence is proved to be a congruence and it is also developed a form of probabilistic bisimulation that works as a sound method for demonstrating observational equivalence and is also used to prove the soundness of a proof system for reasoning about protocols. Furthermore it is proved that two processes are asymptotically observationally equivalent if and only if they are computationally indistinguishable. The idea is that observational equivalence tells us that if we want to analyze a protocol $\mathcal{P}$ and we have a protocol $\mathcal{Q}$ which is an idealized form of $\mathcal{P}$ where all channels are secure, then proving $\mathcal{P} \cong \mathcal{Q}$ means that $\mathcal{P}$ is secure.

Finally, in the last section of the work, there are several application of the calculus and the proof system to well-known cryptographic constructions.

2.3.2 First-Order Logic

In [28], Impagliazzo and Kapron propose two systems to reason about cryptographic construction; the goal of this work is to propose a framework that is sufficiently powerful to study most of the primitives commonly used in cryptography, but also simple enough to be used in the analysis of combinations or changes of primitives when applied in protocols. It can be easily noticed that the difficulties in developing such system are in formulation of security definitions, reasoning about probability and randomness, quantification of the computational power of adversaries, wrong use of induction in security proofs, and in order to overcome this issues concepts from cryptography, implicit computational complexity and proof complexity are mixed together.

The first logic system, called T, is a first-order logic system whose aim is to reason about probabilities, asymptotic and polynomial time functions. The system is composed by:

- A *security parameter*, that is essential to reason about cryptographic constructions and it is assumed to be large enough to handle asymptotic statement.
- *Strings*, used to represent inputs, outputs and random tapes. Strings must be of polynomial length and they can be seen as integers so that it is possible to work with arithmetical operations.
- *Moderate integers*, that is polynomially bounded by the security parameter. They are used for instance to express negligible functions.
- *Feasible functions*, that is polynomial time functions from strings to strings. They can be defined by composition or (particular) recursion.
- *Counting integers*, used to represent sizes of set of strings. These integers are useful when we want to determine the size of the set of strings that satisfy a particular formula and so it is functional when we reason about *probabilities*.
- *Formulas*, defined from the atomic formulas: $t = s, t \leqslant s$ by the connectives $\neg, \wedge, \vee, \rightarrow, \equiv$ and quantifiers $\forall, \exists$.

Once given the syntax of the system it is proposed an axiomatizations made of a list of axioms divided in: logical, security parameter, basic, poly-time functions, counting and induction; this axiomatizations is necessary to introduce the notion of *derivation*.
Given two formulas φ, ψ, we write $T, \varphi \vdash \psi$ if ψ can be derived from φ and instances of the axioms in T; so we get the *soundness theorem*:

Theorem 2.3 *Suppose that $\varphi_1, \varphi_2, \psi$ are bounded formulas such that:*

$$T, \varphi_2(\vec{f}, \vec{f'}) \qquad \forall g \forall \vec{z} \varphi_1(\vec{f}, g, \vec{z}, \mathbf{s}) \vdash \forall g \forall \vec{z} \psi(\vec{f'}, g, \vec{z}, \mathbf{s})$$

where $\vec{f}, \vec{f'}$ are sequences of function variables and $\mathbf{s}$ is the security parameter. We then have the following for all sequences $\vec{\alpha}, \vec{\alpha}'$ of poly-tyme functions: if $\vec{\alpha}, \vec{\alpha}'$ satisfy

φ_2 in $\mathbb{N}$ and for every poly-time function β, φ_1 holds asymptotically in $\mathbb{N}$, then for every poly-time function β, ψ holds asymptotically in $\mathbb{N}$.

The meaning of this theorem is that f is a cryptographic primitive and f' is built from f, we use the formula φ_2 to define f' in function of f. The formulas φ_1 and ψ formalize the security of f and f' respectively, whereas g is a function that stands for a poly-time adversary. The soundness theorem tells us that if it is possible to derive ψ from φ_1 then we have a sound proof of the security of f' by using the assumption on the security of f.

The T system is quite wide and general, it is conceived to reason about arbitrary cryptographic constructions, however it doesn't avoid any explicit reasoning about probabilities. This is why Impagliazzo and Kapron propose a second logic system focused on computational indistinguishability. This system is composed by the rules in Figure 2.1.

$$\frac{T \vdash Q_1, ..., Q_k(s = t)}{\mathsf{let}\ b_1 \mathsf{in}\ ...\mathsf{let}\ b_k \mathsf{in}\ s \approx \mathsf{let}\ b_1 \mathsf{in}\ ...\mathsf{let}\ b_k \mathsf{in}\ t}\ \text{UNIV}$$

$$\frac{u \approx u'}{v\{^u/_x\} \approx v\{^u/_x\}}\ (\text{SUB}) \qquad \frac{\mathsf{let}\ i \leftarrow \mathsf{rand}(p(\mathbf{n}))\ \mathsf{in}\ u \approx \mathsf{let} i \leftarrow \mathsf{rand}(p(\mathbf{n}))\ \mathsf{in}\ u\{^{i+1}/_i\}}{u\{^0/_i\} \approx u\{^{p(1^{\mathbf{n}})}/_i\}}\ (\text{H-IND})$$

$$\frac{}{\mathsf{let}\ \begin{pmatrix} \vec{i} \leftarrow \mathsf{rand}(\vec{p}(\mathbf{n})) \\ x \leftarrow \mathsf{rs}(\sum_{j=1}^{k}(p_j(\mathbf{n}) - i_j)) \end{pmatrix} \mathsf{in}\ x \approx \begin{pmatrix} \vec{i} \leftarrow \mathsf{rand}(\vec{p}(\mathbf{n})) \\ x \leftarrow \mathsf{rs}(p(\mathbf{n})) \end{pmatrix} \mathsf{in}\ \bigcirc_{j=i}^{k} x_{j_{\{1...p_j(\mathbf{n})-i_j\}}}}\ (\text{EDIT})$$

Figure 2.1: Rules for CI (Impagliazzo, Kapron)

We don't go into details of the second system syntax, we just explain the meaning of each rule. In the rule (UNIV), each Q_i is a universal quantifier and b_i a random bitstring, the aim of this rule is to relate the universally quantified equality statement provable in T to $\approx$. The (SUB) rule tells us that we are able to substitute terms that are computationally indistinguishable into PPT contexts. The (EDIT) rule

allows us to merge, split and/or shorten random strings and we get a result that is indistinguishable from a random string of the appropriate length. Finally (H-IND) can be seen as an induction rule.

The soundness proof of this system is made by analyzing each rule and interpreting it in the general system. Finally this system is used to prove the correctness of a pseudorandom generator built by using the property of next-bit unpredictability.

2.3.3 A λ-Calculus

A similar approach to computational indistinguishability has been made by Zhang, that developed a logic for reasoning about CI starting from a language called *computational SRL*(CSLR) [38]; this language is an extension of Hofmann's SRL and its main feature is to capture the class of probabilistic polynomial time computation so that it is very useful to model cryptographic constructions and adversaries.

The syntax of CSLR is defined starting from terms of SLR:

$$e_1, e_2, ... ::= x \mid \mathsf{nil} \mid \mathsf{B_0} \mid \mathsf{B_1} \mid \mathsf{case}_\tau \mid \mathsf{rec}_\tau \mid \lambda x.e \mid e_1 e_2$$

$$\langle e_1, e_2 \rangle \mid \mathsf{proj_1} e \mid \mathsf{proj_2} e \mid e_1 \otimes e_2 \mid \mathsf{let}\ x \otimes y = e_1 \mathsf{in}\ e_2$$

and adding terms for probabilistic computations as:

$$e_1, e_2, ... ::= ... \mid \mathsf{rand} \mid \mathsf{val}(e) \mid \mathsf{bind}\ x = e_1 \mathsf{in}\ e_2$$

In this syntax we have that nil is the empty string, $\mathsf{B_0}, \mathsf{B_1}$ are bitstring constructors, case_τ is the term for case distinction, rec_τ is for safe recursion, $\lambda x.e$ is a lambda abstraction, $e_1 e_2$ is an application, $\langle e_1, e_2 \rangle$ is a product, $\mathsf{proj_i}$ are product projections, $e_1 \otimes e_2$ is a tensor product and $\mathsf{let}\ x \otimes y = e_1 \mathsf{in}\ e_2$ is a tensor projection. Furthermore, rand is an oracle bit that returns 0 or 1 with probability $1/2$, $\mathsf{val}(e)$ is the deterministic computation and $\mathsf{bind}\ x = e_1 \mathsf{in}\ e_2$ is a sequential computation.

The type system, necessary to ensure the polytime soundness, is also inherited from SLR; types are defined by:

$$\tau, \tau', ... ::= \mathsf{Bits} \mid \tau \times \tau' \mid \tau \otimes \tau' \mid \square\, \tau \to \tau' \mid \tau \to \tau' \mid \tau \multimap \tau' \mid \mathsf{T}\tau$$

Bits is the base type, $\tau \times \tau'$ is the cartesian product, $\tau \otimes \tau'$ is the tensor product; $\square\tau \to \tau'$ is for modal function with no restrictions on arguments, $\tau \to \tau'$ is for non-modal functions where the arguments must be *safe* arguments and $\tau \multimap \tau'$ is for linear functions. The type $\mathsf{T}\tau$ is part of the extension and it is called monadic or computation type; it is used for computations that return a value of type τ.

The proof system developed by Zhang is composed by two different set of rules: the first one is used to reason about semantic equivalence, denoted with $\equiv$, whereas the second is made by rules to justify computational indistinguishability, denoted with $\simeq$. We focus now our attention to the second one, in figure 2.2, that is is quite similar to the one proposed by Impagliazzo and Kapron.

$$\frac{\vdash e_i : \square\mathsf{Bits} \to \tau \quad (i = 1, 2) \qquad e_1 \equiv e_2}{e_1 \simeq e_2}\ \text{EQUIV}$$

$$\frac{\vdash e_i : \square\mathsf{Bits} \to \tau \quad (i = 1, 2, 3) \qquad e_1 \simeq e_2 \qquad e_2 \simeq e_3}{e_1 \simeq e_3}\ \text{TRANS-INDIST}$$

$$\frac{x : \mathsf{Bits}, y : \tau \vdash e : \tau' \qquad \vdash e_i : \square\mathsf{Bits} \to \tau \quad (i = 1, 2) \qquad e_1 \simeq e_2}{\lambda x.e\left[{}^{e_1(x)}/_{y}\right] \simeq \lambda x.e\left[{}^{e_2(x)}/_{y}\right]}\ \text{SUB}$$

$$\frac{\begin{array}{c} x : \mathsf{Bits}, n\mathsf{Bits} \vdash e : \tau \qquad \lambda n.e\left[{}^{u}/_{x}\right] \text{ is numerical for all bitstrings } u \\ \lambda x.e\left[{}^{i(x)}/_{n}\right] \simeq \lambda x.e\left[{}^{\mathsf{B}_1 i(x)}/_{n}\right] \text{ for all canonical polynomial } i \text{ such that } |i| \leqslant |p| \end{array}}{\lambda x.e\left[{}^{\mathsf{nil}}/_{n}\right] \simeq \lambda x.e\left[{}^{p(x)}/_{n}\right]}\ \text{H-IND}$$

Figure 2.2: Rules for CI (Zhang)

One of the difference of this system from the one proposed by Impagliazzo and Kapron is the absence of a rule (EDIT); this is due to the fact that in CSLR there

are no primitives that modify bitstrings, except for the two bitstring constructor. Another difference can be found in the H-IND rules, because in this logic there is not a primitive that returns uniformly a number smaller than a polynomial. It is finally important to notice that the TRANS-INDIST rule does not break any assumption about the complexity constraint. By the add of some useful lemmas, Zhang shows how this proof system can be used to analyze cryptographic examples of pseudorandom generators.

In his joint work with Nowak, Zhang propose CSLR+, an extension of the language realised to allow the possibility to work with games that use superpolynomial time computations or arbitrary uniform distributions [37]; this feature is necessary to handle security definitions, but the construction of adversaries and cryptographic primitives is still bounded polinomially. This add lead Nowak and Zhang to introduce the notion of *game indistinguishability*, a definition that is not stronger than computational indistinguishability, but it is more appropriate in a game-based proof framework.

One of the contribution of this work is the proof that computational indistinguishability implies game indistinguishability, so it is still possible to use the proof system in [38] in proof. Nowak and Zhang also show concrete applications of the proof system, by implementing in CSRL the public key encryption scheme of ElGamal and proving it secure; this proof relies on the formalization of the *decisional Diffie-Hellmann assumption*, i.e. the nonexistence of a computational algorithm able to distinguish between the triples $(\gamma^x, \gamma^y, \gamma^{xy})$ and $(\gamma^x, \gamma^y, \gamma^z)$ when x, y, z are chosen randomly. Another contribution is the implementation and the security definition of the Blum-Blum-Shub pseudorandom generator.

In this chapter we discussed some of the approaches used to enlarge the set of primitives and protocols that can be studied by the computational point of view: we started from formal methods computationally sound, then we talked about dif-

ferent authomated tools and we end with the description of some process calculi endowed with a proof system to reason aboun computational indistinguishability, an approach very close to the one proposed in this thesis. In the following chapter we will introduce the calculus we will use to model our framework, RSLR, a typed λ-calculus for PPT computations, that we will use to model cryptographic schemes and adversaries.

Chapter 3

A Calculus for PPT Computation: RSLR

In this first section we present RSLR, a $\lambda-$calculus for probabilistic polynomial time computations. The choice to use RSLR is based on the fact that the final goal of this thesis is to offer a method to study computational indistinguishability, where an adversary $\mathcal{A}$, an algorithm with a polynomial power of calculus, takes in input two different programs once each and tries to distinguish between them; as we will see in the following, the main feature of this calculus is that the set of probabilistic functions that can be computed by RSLR terms coincides with the polytime computable ones, so essentially, the idea behind this choice is that we can use an RSLR term to describe whatever algorithm $\mathcal{A}$. We will formalize this concept when we will talk about contexts, RSLR terms with a hole, that represent in this system the adversaries that takes in input a term and, once studied, return an output.

The other feature of RSLR that pushed us to this choice is the presence of a probabilistic operator. The possibility to work with probability and to write probabilistic programs is crucial when we deal with cryptography, indeed it is well-known that a deterministic cryptographic primitive is not semantically secure.

RSLR, which stands for Random Safe Linear Recursion, is obtained by extending Hofmann's SLR with an operator for binary probabilistic choice. One of the dif-

ferences between RSLR and other languages obtained from Hofmann's SLR is that polynomial time soundness is proved operationally instead of semantically and so it brings to some necessary restrictions from the original SLR. Furthermore in SLR we have two different function spaces, whereas in RSLR these two spaces collapse into one; this difference comes from the fact that, in presence of higher-order duplication, it results very difficult to control the size of reducts when we normalize. Thus, as a consequence, RSLR merges the two function spaces and, by using a strict type system, prevents the duplication of arguments of higher-order type.

In this work we will use a version of RSLR which is slightly different from the original one proposed in [31]: we consider a version in which base terms are bistring instead of natural numbers and a call-by-value reduction; this way we obtain a simpler exposition of the theory without losing expressiveness.

3.1 Syntax and Semantics

As we disclosed, RSLR is a typed lambda calculus for probabilistic polynomial time computation; the type system adopted by RSLR is crucial to ensure the polynomial complexity and it is based on the idea that variables of a certain type can appear in a term only once. We introduce the type system by defining the category of type.

Definition 3 (Types) *Types in* RSLR *are defined as follows:*

$$\mathsf{A} ::= \mathsf{Str} \mid \blacksquare\mathsf{A} \to \mathsf{A} \mid \square\,\mathsf{A} \to \mathsf{A}$$

We can easily see that we have one base type Str which stands for bitstrings and two different function spaces: the first one, characterized by the type $\blacksquare\mathsf{A} \to \mathsf{A}$ describes all the function that are evaluated in constant time whereas the the one, $\square\mathsf{A} \to \mathsf{A}$ is for the functions that require a time of computation that is polynomial in the size of the argument.

In order to highlight the fact that $\blacksquare\mathsf{A} \to \mathsf{A}$ is a subtype of $\square\mathsf{A} \to \mathsf{A}$, RSLR provides for a notion of *aspects*, denoted with metavariables a, b, which is used to formalize the concept of subtyping.

Definition 4 *We define an aspect as* $\blacksquare$ *or* $\square$*; we define a partial order between aspects by using the binary relation* $\{(\square, \square), (\square, \blacksquare), (\blacksquare, \blacksquare)\}$*, that is noted with* $<:$*.*

Now we are able to give the system of subtyping rules, as described in figure 3.1.

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

Figure 3.1: Subtyping Rules

At this point we go into the core of the language by describing the syntax of RSLR.

Definition 5 *The synctatical categories of* values *and* terms *are defined by the following grammar:*

$$v \ ::= \underline{\mathrm{m}} \ \Big| \ \lambda x : \mathsf{aA}.t;$$

$$t \ ::= x \ \Big| \ v \ \Big| \ \mathsf{0}(t) \ \Big| \ \mathsf{1}(t) \ \Big| \ \mathsf{tail}(t) \ \Big| \ tt \ \Big| \ \mathsf{case_A}(t,t,t,t) \ \Big| \ \mathsf{rec_A}(t,t,t,t) \ \Big| \ \mathsf{rand};$$

As we can see we have that values are given by strings, denoted with $\underline{\mathrm{m}}$, which range over the set of finite binary strings $\{0,1\}^*$ and by lambda abstractions; terms are given by variables, where x ranges over a denumerable set of variables $\mathbf{X}$, two string constructors $\mathsf{0}, \mathsf{1}$ and a string destructor tail. Furthermore we have applications and a nonstandard constant rand that returns $\underline{0}$ or $\underline{1}$ with probability $1/2$. The terms $\mathsf{case_A}(t, t_0, t_1, t_\epsilon)$ and $\mathsf{rec_A}(t, t_0, t_1, t_\epsilon)$ are terms for case distinction and safe recursion, in which the first argument specifies the term (of tipe Str) which guides the process.

We give in figure 3.2 a one step semantics in order to explain better the behaviour of terms in RSLR; we use $\underline{\epsilon}$ to denote the empty string, $\underline{\mathsf{b}} \in \{0,1\}$ to denote a single bit, and $\{0^{\frac{1}{2}}, 1^{\frac{1}{2}}\}$ for the distribution that assigns the values $0, 1$ with probability $\frac{1}{2}$.

$$
\begin{array}{rlrl}
x & \to x; & v & \to v; \\
\mathsf{0}(\underline{\mathrm{m}}) & \to \underline{0\mathrm{m}}; & \mathsf{1}(\underline{\mathrm{m}}) & \to \underline{1\mathrm{m}}; \\
\mathsf{tail}(\underline{\epsilon}) & \to \underline{\epsilon}; & \mathsf{tail}(\underline{\mathrm{bm}}) & \to \underline{\mathrm{m}}; \\
(\lambda x.t)v & \to t\{^{v}/_{x}\}; & \mathsf{rand} & \to \{\underline{0}^{\frac{1}{2}}, \underline{1}^{\frac{1}{2}}\}; \\
\mathsf{case}_{\mathsf{A}}(\underline{\epsilon}, t_0, t_1, t_\epsilon) & \to t_\epsilon; & \mathsf{rec}_{\mathsf{A}}(\underline{\epsilon}, t_0, t_1, t_\epsilon) & \to t_\epsilon \\
\mathsf{case}_{\mathsf{A}}(\underline{0\mathrm{m}}, t_0, t_1, t_\epsilon) & \to t_0; & \mathsf{rec}_{\mathsf{A}}(\underline{0\mathrm{m}}, t_0, t_1, t_\epsilon) & \to (t_0\underline{0\mathrm{m}})(\mathsf{rec}_{\mathsf{A}}(\underline{\mathrm{m}}, t_0, t_1, t_\epsilon)) \\
\mathsf{case}_{\mathsf{A}}(\underline{1\mathrm{m}}, t_0, t_1, t_\epsilon) & \to t_1; & \mathsf{rec}_{\mathsf{A}}(\underline{1\mathrm{m}}, t_0, t_1, t_\epsilon) & \to (t_1\underline{1\mathrm{m}})(\mathsf{rec}_{\mathsf{A}}(\underline{\mathrm{m}}, t_0, t_1, t_\epsilon))
\end{array}
$$

Figure 3.2: RSLR One-Step Semantics Rules

The presence of the probabilistic term rand in the syntax of RSLR makes the operational semantics probabilistic; indeed we have that any closed term does not evaluate to a single value but to a value distribution. A *value distribution* is a function $\mathcal{D} : \mathrm{V} \to \mathbb{R}_{[0,1]}$ such that $\sum_{v \in \mathrm{V}} \mathcal{D}(v) = 1$; if $\sum_{v \in \mathrm{V}} \mathcal{D}(v) < 1$ then $\mathcal{D}$ is called *value sub-distribution*. The evaluation of a term t to a value distribution $\mathcal{D}$ is expressed by using the judgment $t \Downarrow \mathcal{D}$; the set of the values v such that $\mathcal{D}(v) \neq 0$ is called the *support* of $\mathcal{D}$ and it's denoted by $\mathsf{S}(\mathcal{D})$. We denote a value distribution $\mathcal{D}$ with $\{(v_i)^{p_i}\}_{i \in I}$ where $\{v_i\}_{i \in I} = \mathsf{S}(\mathcal{D})$ and $p_i = \mathcal{D}(v_i)$ for all $i \in I$. Given two value distributions (or sub-distribution) $\mathcal{D}, \mathcal{E}$ and a number $p \in [0,1]$ we denote:

$$\mathcal{D} + \mathcal{E} = \{(v)^{\mathcal{D}(v)+\mathcal{E}(v)}\}_{v \in \mathsf{S}(\mathcal{D}) \cup \mathsf{S}(\mathcal{E})}$$

$$p \cdot \mathcal{D} = \{(v)^{p \cdot \mathcal{D}(v)}\}_{v \in \mathsf{S}(\mathcal{D})}$$

$$\frac{}{v \Downarrow \{v^1\}} \qquad \frac{}{\mathsf{rand} \Downarrow \{\underline{0}^{\frac{1}{2}}, \underline{1}^{\frac{1}{2}}\}} \qquad \frac{t \Downarrow \mathcal{D} \qquad s \Downarrow \mathcal{E} \qquad \{r\{^v/_x\} \Downarrow \mathcal{F}_{r,v}\}_{\lambda x.r,v}}{ts \Downarrow \sum_{\lambda x.r,v} \mathcal{D}(\lambda x.r) \cdot \mathcal{E}(v) \cdot \mathcal{F}_{r,v}}$$

$$\frac{t \Downarrow \{(\underline{\mathrm{m}}_i)^{p_i}\}}{\mathsf{0}(t) \Downarrow \{(\underline{0\mathrm{m}}_i)^{p_i}\}} \qquad \frac{t \Downarrow \{(\underline{\mathrm{m}}_i)^{p_i}\}}{\mathsf{1}(t) \Downarrow \{(\underline{1\mathrm{m}}_i)^{p_i}\}} \qquad \frac{t \Downarrow \mathcal{D} \qquad \mathbf{T}(\underline{\mathrm{m}}_i) = \{\mathsf{0}\underline{\mathrm{m}}_i, \mathsf{1}\underline{\mathrm{m}}_i\}}{\mathsf{tail}(t) \Downarrow \{(\underline{\mathrm{m}}_i)^{\mathcal{D}(\mathbf{T}(\underline{\mathrm{m}}_i))}\}}$$

$$\frac{t \Downarrow \mathcal{D} \qquad t_0 \Downarrow \mathcal{D}_0 \qquad t_1 \Downarrow \mathcal{D}_1 \qquad t_\epsilon \Downarrow \mathcal{D}_\epsilon}{\mathsf{case}_{\mathsf{A}}(t, t_0, t_1, t_\epsilon) \Downarrow \sum_{\underline{\mathrm{m}}} \mathcal{D}(\underline{0\mathrm{m}}) \cdot \mathcal{D}_0 + \sum_{\underline{\mathrm{m}}} \mathcal{D}(\underline{1\mathrm{m}}) \cdot \mathcal{D}_1 + \mathcal{D}(\underline{\epsilon}) \cdot \mathcal{D}_\epsilon}$$

$$\frac{\begin{array}{cc} t \Downarrow \mathcal{D} & \{(t_0\underline{\mathrm{m}})(\mathsf{rec}_{\mathsf{A}}(\underline{\mathrm{n}}, t_0, t_1, t_\epsilon)) \Downarrow \mathcal{D}_{\underline{\mathrm{m}}}\}_{\underline{\mathrm{m}}=\underline{0\mathrm{n}}} \\ t_\epsilon \Downarrow \mathcal{D}_{\underline{\epsilon}} & \{(t_1\underline{\mathrm{m}})(\mathsf{rec}_{\mathsf{A}}(\underline{\mathrm{n}}, t_0, t_1, t_\epsilon)) \Downarrow \mathcal{D}_{\underline{\mathrm{m}}}\}_{\underline{\mathrm{m}}=\underline{1\mathrm{n}}} \end{array}}{\mathsf{rec}_{\mathsf{A}}(t, t_0, t_1, t_\epsilon) \Downarrow \sum_{\underline{\mathrm{m}}} \mathcal{D}(\underline{\mathrm{m}})\mathcal{D}_{\underline{\mathrm{m}}}}$$

Figure 3.3: Big-Step Semantics Rules

The operational semantics of a term in RSLR is given by the rules in figure 3.3.

Now we can go into details and illustrate the whole type system that is used in RSLR. First of all we define a *typing context* Γ as a finite set of assignments of an aspect and a type to a variable, where every variable occurs at most once; an assignment is indicated with $x : \mathsf{aA}$. We use the expression with a simple comma Γ, Δ for the disjoint union of two typing contexts Γ and Δ; when we want emphasize that Γ only involve variables of base type Str we use the semicolon as in $\Gamma; \Delta$. *Typing judgment* are in the form $\Gamma \vdash t : \mathsf{A}$ and the complete type system is given by the rules in figure 3.4. It is easy to observe how this type system enforces variables of higher order type to occur free at most once and outside the scope of a recursion; moreover the type of a term that serves as step-function in a recursion is assumed to be $\square$-free and these are key points that allow the calculus to characterize polytime functions. We define $\mathrm{T}^{\mathsf{A}}_{\Gamma}, \mathrm{V}^{\mathsf{A}}_{\Gamma}$ as the sets of terms and values of type A under the typing context Γ. In particular, we are interested in $\mathrm{T}^{\mathsf{A}}_{\varnothing}, \mathrm{V}^{\mathsf{A}}_{\varnothing}$, the sets of *closed terms* and *closed values* (i.e. without free variables).

$$\frac{x : \mathsf{aA} \in \Gamma}{\Gamma \vdash x : \mathsf{A}} \qquad \frac{}{\Gamma \vdash \underline{\mathrm{m}} : \mathsf{Str}} \qquad \frac{\Gamma \vdash t : \mathsf{Str}}{\Gamma \vdash 0(t) : \mathsf{Str}} \qquad \frac{\Gamma \vdash t : \mathsf{Str}}{\Gamma \vdash : 1(t) : \mathsf{Str}}$$

$$\frac{\Gamma \vdash t : \mathsf{Str}}{\Gamma \vdash \mathsf{tail}(t) : \mathsf{Str}} \qquad \frac{}{\vdash \mathsf{rand} : \mathsf{Str}} \qquad \frac{\Gamma \vdash t : \mathsf{A} \qquad \mathsf{A} <: \mathsf{B}}{\Gamma \vdash t : \mathsf{B}}$$

$$\frac{\begin{array}{c}\Gamma; \Delta_1 \vdash t : \mathsf{Str} \quad \Gamma; \Delta_3 \vdash t_1 : \mathsf{A} \\ \Gamma; \Delta_2 \vdash t_0 : \mathsf{A} \quad \Gamma; \Delta_4 \vdash t_\epsilon : \mathsf{A}\end{array}}{\Gamma; \Delta_1, \Delta_2, \Delta_3, \Delta_4 \vdash \mathsf{case}_{\mathsf{A}}(t, t_0, t_1, t_\epsilon) : \mathsf{A}} \qquad \frac{\begin{array}{c}\Gamma; \Delta_1 \vdash t : \mathsf{aA} \to \mathsf{B} \\ \Gamma; \Delta_2 \vdash s : \mathsf{A} \qquad \Gamma, \Delta_2 <: \mathsf{a}\end{array}}{\Gamma; \Delta_1, \Delta_2 \vdash ts : \mathsf{B}}$$

$$\frac{\begin{array}{cc}\Gamma_1; \Delta_1 \vdash t : \mathsf{Str} & \Gamma_1, \Gamma_2, \Gamma_3; \Delta_2 \vdash t_\epsilon : \mathsf{A} \\ \Gamma_1, \Gamma_2 \vdash t_0 : \square\mathsf{Str} \to \blacksquare\mathsf{A} \to \mathsf{A} & \Gamma_1, \Delta_1 <: \square \\ \Gamma_1, \Gamma_3 \vdash t_1 : \square\mathsf{Str} \to \blacksquare\mathsf{A} \to \mathsf{A} & \mathsf{A} \text{ is } \square\text{-free}\end{array}}{\Gamma_1, \Gamma_2, \Gamma_3; \Delta_1, \Delta_2 \vdash \mathsf{rec}_{\mathsf{A}}(t, t_0, t_1, t_\epsilon) : \mathsf{A}} \qquad \frac{\Gamma, x : \mathsf{aA} \vdash t : \mathsf{B}}{\Gamma \vdash \lambda x : \mathsf{aA}.t : \mathsf{aA} \to \mathsf{B}}$$

Figure 3.4: RSLR Typing System

Lemma 3.1 (Subject reduction) *Given a term t such that $\vdash t : \mathsf{A}$, if it reduces to $t_1, ..., t_n$, we have that $\vdash t_i : \mathsf{A}$.*

Proof: This is proved by induction on the type derivation [31]. □

Lemma 3.2 *For every term $t \in \mathrm{T}^{\mathsf{A}}_{\varnothing}$ there is a unique value distribution $\mathcal{D}$ such that $t \Downarrow \mathcal{D}$ and we denote it with $[\![t]\!]$. Moreover, if $v \in \mathsf{S}(\mathcal{D})$ then $v \in \mathrm{V}^{\mathsf{A}}_{\varnothing}$.*

Proof: We proceed by induction on the structure of t.

- If we have a value v, then by the rules it converges to $\{v^1\}$.
- Similarly if we have a term rand the only distribution it can converge is $\{\underline{0}^{\frac{1}{2}}, \underline{1}^{\frac{1}{2}}\}$.
- Suppose now to have $t_1 t_2$, and suppose $t_1 t_2 \Downarrow \mathcal{D}, t_1 t_2 \Downarrow \mathcal{D}'$, i.e. that two

distributions exist for t_1, t_2. By construction we have:

$$\mathcal{D} = \sum_{\lambda x.t,v} \mathcal{D}_1(\lambda x.t) \cdot \mathcal{D}_2(v) \cdot \mathcal{D}_{t,v} \qquad \mathcal{D}' = \sum_{\lambda x.t',v'} \mathcal{D}'_1(\lambda x.t') \cdot \mathcal{D}'_2(v') \cdot \mathcal{D}'_{t',v'}$$

But, by induction hypothesis we have $\mathcal{D}_1 = \mathcal{D}'_1, \mathcal{D}_2 = \mathcal{D}'_2$ and so also $\mathcal{D}_{t,v} = \mathcal{D}'_{t',v'}$ and this means $\mathcal{D} = \mathcal{D}'$.

- All the other cases are similar.

The second point comes from the RSLR property of subject reduction, so by combining the fact that the type is preserved by reduction and the uniqueness of $\mathcal{D}$ we have that for all $v \in \mathsf{S}(\mathcal{D}), \vdash v : \mathsf{A}$. □

The main feature of RSLR is the polytime soundness and completeness; before presenting this result, we define a *probabilistic function* on $\{0,1\}^*$ as a function $F : \{0,1\}^* \to \mathbb{P}_{\{0,1\}^*}$, where $\mathbb{P}_{\{0,1\}^*}$ is the set of probabilistic distributions. A term $t \in \mathrm{T}_{\varnothing}^{\mathsf{aA}\to\mathsf{B}}$ is said to compute F if for every string $\underline{\mathrm{m}} \in \{0,1\}^*$ it holds that $t\underline{\mathrm{m}} \Downarrow \mathcal{D}$, where $\mathcal{D}(\underline{\mathrm{n}}) = F(\underline{\mathrm{m}})(\underline{\mathrm{n}})$, for every $\underline{\mathrm{n}} \in \{0,1\}^*$. This means that $t\underline{\mathrm{m}}$ evaluates to the same probability distribution of $F\underline{\mathrm{m}}$, so, the probability to get the value $\underline{\mathrm{n}}$ as result of $t\underline{\mathrm{m}}$ is the same as the probabilty to have $\underline{\mathrm{n}}$ as result of $F\underline{\mathrm{m}}$. Now we can recall the main characteristic of RSLR, a result which is well-known and can be proved in various ways.

Theorem 3.1 *The set of probabilistic functions which can be computed by* RSLR *terms coincide with the polytime computable ones.*

3.2 Examples of Programs in RSLR

We use this section to propose some programs written as RSLR terms; the purpose is to describe the calculus and to introduce some programs that will be useful in the following when we'll deal with security examples.

We start with two programs t, s that receive a string in input. The first one is the identity. The second one, instead, produces a random string and compare it to the one received in input; if they are different it is the identity, otherwise it returns the opposite.

$$t := \lambda x : \square\mathsf{Str}.x \qquad s := \lambda x : \square\mathsf{Str}.\mathsf{case}_{\mathsf{Str}}(x = (\mathtt{RBG}\ x), x, \neg x, \neg x)$$

Where:

$$\mathtt{RBG} := \lambda y : \square\mathsf{Str}.\mathsf{rec}_{\mathsf{Str}}(y, \mathtt{f}_{\mathtt{RBG}}, \mathtt{f}_{\mathtt{RBG}}, \underline{\epsilon})$$
$$\mathtt{f}_{\mathtt{RBG}} := \lambda w : \square\mathsf{Str}.\lambda z : \blacksquare\mathsf{Str}.\mathsf{case}_{\mathsf{Str}}(\mathsf{rand}, 0(z), 1(z), \underline{\epsilon})$$

Notice that, even if we haven't defined $=$ and $\neg$, they are easily implementable in RSLR.

We give now a simple example of how the big step semantics of a RSLR term is evaluated; we observe the term RBG applied to a string $\underline{01}$.

$$\begin{aligned}[\![\mathtt{RBG}\ \underline{01}]\!] =& [\![\mathtt{RBG}]\!](\lambda y.\mathsf{rec}_{\mathsf{Str}}(y, \mathtt{f}_{\mathtt{RBG}}, \mathtt{f}_{\mathtt{RBG}}, \underline{\epsilon})) \cdot [\![\underline{01}]\!](\underline{01}) \cdot [\![\mathsf{rec}_{\mathsf{Str}}(\underline{01}, \mathtt{f}_{\mathtt{RBG}}, \mathtt{f}_{\mathtt{RBG}}, \underline{\epsilon})]\!] \\ =& 1 \cdot 1 \cdot [\![\mathsf{rec}_{\mathsf{Str}}(\underline{01}, \mathtt{f}_{\mathtt{RBG}}, \mathtt{f}_{\mathtt{RBG}}, \underline{\epsilon})]\!] = \\ =& [\![\underline{01}]\!](\underline{01}) \cdot [\![(\mathtt{f}_{\mathtt{RBG}}\underline{01})(\mathsf{rec}_{\mathsf{Str}}(\underline{1}, \mathtt{f}_{\mathtt{RBG}}, \mathtt{f}_{\mathtt{RBG}}, \underline{\epsilon}))]\!] = [\![(\mathtt{f}_{\mathtt{RBG}}\underline{01})(\mathsf{rec}_{\mathsf{Str}}(\underline{1}, \mathtt{f}_{\mathtt{RBG}}, \mathtt{f}_{\mathtt{RBG}}, \underline{\epsilon}))]\!]\end{aligned}$$

We can easily say that $[\![\mathtt{f}_{\mathtt{RBG}}\underline{01}]\!] = [\![\mathtt{f}_{\mathtt{RBG}}\{\underline{01}/w\}]\!] = \{(\lambda z.\mathsf{case}_{\mathsf{Str}}(\mathsf{rand}, 0(z), 1(z), \underline{\epsilon}))^1\}$. Furthermore we have:

$$[\![\mathsf{rec}_{\mathsf{Str}}(\underline{1}, \mathtt{f}_{\mathtt{RBG}}, \mathtt{f}_{\mathtt{RBG}}, \underline{\epsilon})]\!] = [\![\underline{1}]\!](\underline{1}) \cdot [\![(\mathtt{f}_{\mathtt{RBG}}\underline{1})(\mathsf{rec}_{\mathsf{Str}}(\underline{\epsilon}, \mathtt{f}_{\mathtt{RBG}}, \mathtt{f}_{\mathtt{RBG}}, \underline{\epsilon}))]\!]$$

So, by the fact that $[\![\mathsf{rec}_{\mathsf{Str}}(\underline{\epsilon}, \mathtt{f}_{\mathtt{RBG}}, \mathtt{f}_{\mathtt{RBG}}, \underline{\epsilon})]\!] = \{\underline{\epsilon}^1\}$ we have:

$$\begin{aligned}[\![\mathsf{rec}_{\mathsf{Str}}(\underline{1}, \mathtt{f}_{\mathtt{RBG}}, \mathtt{f}_{\mathtt{RBG}}, \underline{\epsilon})]\!] =& [\![\mathsf{case}_{\mathsf{Str}}(\mathsf{rand}, 0(\underline{\epsilon}), 1(\underline{\epsilon}), \underline{\epsilon})]\!] = \\ & [\![\mathsf{rand}]\!](\underline{0}) \cdot [\![0(\underline{\epsilon})]\!] + [\![\mathsf{rand}]\!](\underline{1}) \cdot [\![1(\underline{\epsilon})]\!] = \{\underline{0}^{\frac{1}{2}}, \underline{1}^{\frac{1}{2}}\}\end{aligned}$$

So, by substituting we have:

$$\begin{aligned}
[\![\mathtt{RBG}\ \underline{01}]\!] =& [\![(\mathtt{f}_{\mathtt{RBG}}\underline{01})(\mathsf{rec}_{\mathsf{Str}}(\underline{1}, \mathtt{f}_{\mathtt{RBG}}, \mathtt{f}_{\mathtt{RBG}}, \underline{\epsilon}))]\!] = \\
=& \frac{1}{2} \cdot [\![\mathsf{case}_{\mathsf{Str}}(\mathsf{rand}, 0(\underline{0}), 1(\underline{0}), \underline{\epsilon})]\!] + \frac{1}{2} \cdot [\![\mathsf{case}_{\mathsf{Str}}(\mathsf{rand}, 0(\underline{1}), 1(\underline{1}), \underline{\epsilon})]\!] = \\
=& \frac{1}{2} \cdot \{\underline{00}^{\frac{1}{2}}, \underline{10}^{\frac{1}{2}}\} + \frac{1}{2} \cdot \{\underline{01}^{\frac{1}{2}}, \underline{11}^{\frac{1}{2}}\} = \\
=& \{\underline{00}^{\frac{1}{4}}, \underline{01}^{\frac{1}{4}}, \underline{10}^{\frac{1}{4}}, \underline{11}^{\frac{1}{4}}\}
\end{aligned}$$

Summing up, in this section we have introduced the calculus we will use in the following of the thesis: RSLR; we showed its properties and so we have a calculus for probabilistic polynomial time computation that is perfect to model cryptographic primitives, protocol and potential adversaries. Now we are interested in observing relations between terms of RSLR, in particular we want to find out a method that proves the impossibility to observe differences between two terms.

So, we start from the following chapter, where we will talk about equivalence of terms in RSLR and how to prove them in way easier than the quantification over all possible observers.

Chapter 4

Equivalences

In the previous chapter we described a calculus to probabilistic polynomial time computation, RSLR; in this section we introduce different methods to compare programs written in RSLR, in particular we observe when we can say that to terms are *equivalent*.

Why is this important? As we have seen in the introduction studying if two programs are equivalent or not is a key point in security game-based proof, indeed equivalence can be used to justify transitions from a game to another one. Intuitively, we can say that two programs are equivalent if no one can distinguish between them by observing their external visible behavior; a formalization of this intuition is given by proposing the concept of *context*. A contexts is defined by taking the syntax of terms and allowing one or more subterms to be a special variable $[\cdot]$ that is called *hole*.

In this thesis we will focus our attention on contexts with at most one hole, we will work particularly on *linear* contexts, that are contexts in which the hole lies outside the scope of any recursion operator; once defined the syntax of contexts we will give a definition of context equivalence and then we will propose two different methods that characterize it.

The first method is based on traces; we will define how a term evolves by performing a trace and then we will observe the differences between two terms performing the same trace. Even if this method does not solve the problem of a universal quantification, it allows us to work with traces that have a structure which is simpler than contexts.

The second method is based on a coinductive approach; we will define a labeled transition system by using a Markov chain and then we will define a binary relation called *applicative bisimulation* on it. The strong point of this method is that we don't have a universal quantification, furthermore this is a method that is sound w.r.t. all contexts, even if they are not linear; the weak point is that this method is sound but not complete as we will see later.

4.1 Linear Contexts

Definition 6 *A* context *is a term with a unique hole, defined by the following grammar:*

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

As we said before we focus our attention on *linear* contexts, indeed in order to get a *nonlinear* context we have to extend the grammar above with the constructs $\mathsf{rec_A}(t,C,t,t), \mathsf{rec_A}(t,t,C,t), \mathsf{rec_A}(t,t,t,C)$, but, at the moment, this is not our interest.

The purpose of contexts is to test terms and to find some difference between them, so, given a term t we define $C[t]$ as the RSLR term we obtain by substituting all the occurrences of $[\cdot]$, if any, with t. We consider only contexts that are non-binding, it means that they can be filled only by closed terms, so, in order to be

more precise we give a typing system for contexts as happens as for terms, by the rules showed in figure 4.1. A judgment of the form $\Gamma \vdash C[\vdash \mathsf{A}] : \mathsf{B}$ means that C

$$\frac{\Gamma \vdash t : \mathsf{A}}{\Gamma \vdash t[\varnothing] : \mathsf{A}} \qquad \frac{}{\vdash [\vdash \mathsf{A}] : \mathsf{A}} \qquad \frac{\Gamma \vdash C[\vdash \mathsf{A}] : \mathsf{Str}}{\Gamma \vdash 0(C[\vdash \mathsf{A}]), 1(C[\vdash \mathsf{A}]), \mathsf{tail}(C[\vdash \mathsf{A}]) : \mathsf{Str}}$$

$$\frac{x : \mathsf{bB}, \Gamma \vdash C[\vdash \mathsf{A}] : \mathsf{C}}{\Gamma \vdash \lambda x.C[\vdash \mathsf{A}] : \mathsf{bB} \to \mathsf{C}} \qquad \frac{\begin{array}{c}\Gamma; \Delta_1 \vdash C[\vdash \mathsf{A}] : \mathsf{bB} \to \mathsf{C} \\ \Gamma; \Delta_2 \vdash t : \mathsf{B} \qquad \Gamma, \Delta_2 <: \mathsf{b}\end{array}}{\Gamma; \Delta_1, \Delta_2 \vdash Ct[\vdash \mathsf{A}] : \mathsf{C}}$$

$$\frac{\begin{array}{c}\Gamma; \Delta_1 \vdash t : \mathsf{bB} \to \mathsf{C} \\ \Gamma; \Delta_2 \vdash C[\vdash \mathsf{A}] : \mathsf{B} \quad \Gamma, \Delta_2 <: \mathsf{b}\end{array}}{\Gamma; \Delta_1, \Delta_2 \vdash tC[\vdash \mathsf{A}] : \mathsf{C}} \qquad \frac{\begin{array}{c}\Gamma; \Delta_1 \vdash C[\vdash \mathsf{A}] : \mathsf{Str} \quad \Gamma; \Gamma_3 \vdash t_1 : \mathsf{B} \\ \Gamma; \Delta_2 \vdash t_0 : \mathsf{B} \qquad \Gamma; \Delta_4 \vdash t_\epsilon : \mathsf{B}\end{array}}{\Gamma; \Delta_1, \Delta_2, \Delta_3, \Delta_4 \vdash \mathsf{case}_{\mathsf{B}}(C, t_0, t_1, t_\epsilon)[\vdash \mathsf{A}] : \mathsf{B}}$$

$$\frac{\begin{array}{c}\Gamma; \Delta_1 \vdash t : \mathsf{Str} \qquad \Gamma; \Delta_3 \vdash C_1[\vdash \mathsf{A}] : \mathsf{B} \\ \Gamma; \Delta_2 \vdash C_0[\vdash \mathsf{A}] : \mathsf{B} \quad \Gamma; \Delta_4 \vdash C_\epsilon[\vdash \mathsf{A}] : \mathsf{B}\end{array}}{\Gamma; \Delta_1, \Delta_2, \Delta_3, \Delta_4 \vdash \mathsf{case}_{\mathsf{B}}(t, C_0, C_1, C_\epsilon)[\vdash \mathsf{A}] : \mathsf{B}}$$

$$\frac{\begin{array}{c}\Gamma_1; \Delta_1 \vdash C[\vdash \mathsf{A}] : \mathsf{Str} \qquad \Gamma_1, \Gamma_2; \Gamma_3; \Delta_2 \vdash t_\epsilon : \mathsf{B} \\ \Gamma_1, \Gamma_2 \vdash t_0 : \square\mathsf{Str} \to \blacksquare\mathsf{B} \to \mathsf{B} \qquad \Gamma_1, \Delta_1 <: \square \\ \Gamma_1, \Gamma_3 \vdash t_1 : \square\mathsf{Str} \to \blacksquare\mathsf{B} \to \mathsf{B} \qquad \mathsf{B} \ \square\text{-free}\end{array}}{\Gamma_1, \Gamma_2, \Gamma_3; \Delta_1, \Delta_2 \vdash \mathsf{rec}_{\mathsf{B}}(C, t_0, t_1, t_\epsilon)[\vdash \mathsf{A}] : \mathsf{B}}$$

Figure 4.1: Context Typing Rules

must be filled only by a term of type A and, given $t \in \mathrm{T}^{\mathsf{A}}$ we have $C[t] \in \mathrm{T}^{\mathsf{B}}_{\Gamma}$.

Now that the notion of a context is properly defined, we can give one of the central notions of this thesis, the definition of context equivalence.

Definition 7 (Context Equivalence) *Given two terms t, s such that $\vdash t, s : \mathsf{A}$, we say that t and s are* context equivalent *iff for every context C such that $\vdash C[\vdash \mathsf{A}] : \mathsf{Str}$ we have that:*

$$[\![C[t]]\!](\underline{\epsilon}) = [\![C[s]]\!](\underline{\epsilon})$$

.

What we are intuitively doing is taking all contexts and saying that two terms are context equivalent if the probability that they return the empty string when filled with the first one is the same than when they are filled with the second one; notice that it is not restrictive to observe only the empty string as output, because we are quantifying over all the contexts. Indeed if we suppose there exists a context C such that $[\![C[t]]\!](\underline{\mathrm{m}}) \neq [\![C[s]]\!](\underline{\mathrm{m}})$ for a certain string $\underline{\mathrm{m}}$, then we can immediately build a context C' such that $[\![C'[t]]\!](\underline{\epsilon}) \neq [\![C'[s]]\!](\underline{\epsilon})$. By using some syntactic sugar we have:

$$C'[\cdot] := \mathsf{if}\ (C[\cdot] = \underline{\mathrm{m}})\ \mathsf{then}\ \underline{\epsilon}\ \mathsf{else}\ \underline{0}$$

and so:

$$[\![C'[t]]\!](\underline{\epsilon}) = [\![C[t]]\!](\underline{\mathrm{m}}) \neq [\![C[s]]\!](\underline{\mathrm{m}}) = [\![C'[s]]\!](\underline{\epsilon})$$

As we said before, it is obvious that a quantification over all contexts, even if only linear, requires an huge amount of resources, so we need to find a simpler method to prove this kind of equivalence.

4.2 Traces

The first method we propose to characterize context equivalence is based on traces, so first of all we give a definition of trace.

Definition 8 *A* trace *is a sequence of actions of the form* $a_1 \cdot a_2 \cdots a_n$ *such that* $a_i \in \{\mathsf{pass}(v), \mathsf{view}(\underline{\mathrm{m}}) \mid v \in \mathrm{V}, \underline{\mathrm{m}} \in \mathrm{V}^{\mathsf{Str}}\}$. *Traces are indicated with metavariables like* T, S.

Given a term $t : \mathsf{A}$, what a trace intuitively does is to pass a value if the term is a function or to observe it if the term is a string; for this reason we give a notion of compatibility of a trace with a type.

Definition 9 *The* compatibility *of a trace* T *with a type* A *(we write* $\mathsf{T} : \mathsf{A}$*) is defined inductively on the structure of* A*. The empty trace* ϵ *is compatible with every type; if* $\mathsf{A} = \mathsf{Str}$ *then* $\mathsf{T} = \mathsf{view}(\underline{\mathrm{m}})$*, with* $\underline{\mathrm{m}} \in \mathrm{V}^{\mathsf{Str}}$*, or* $\mathsf{T} = \epsilon$*, otherwise, if* $\mathsf{A} = \mathsf{bB} \to C$ *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 to the underlying type; furthermore we say that a trace is *complete* if it ends with the action $\mathsf{view}(\cdot)$ or *incomplete* otherwise.

But, what does it mean for a term t to perform a compatible trace T? By the probabilistic nature of our calculus we know that a term evolves to a distribution of values, so when a trace passes a value or observes a string what it is actually doing is passing a value to a distribution of functions or observing a distribution of strings. By these reasons we think that it is more convenient to work directly with term distributions, i.e. distributions whose support is a set of closed term of a certain type A. We denote term distributions with metavariables like $\mathcal{T}, \mathcal{S}$ and we formalize the effect that traces have on term distributions by introducing the following binary relations:

- The first relation is defined between term distributions and is called $\Rrightarrow$; intuitively $\mathcal{T} \Rrightarrow \mathcal{S}$ iff $\mathcal{T}$ evolves to $\mathcal{S}$ by performing some *internal* moves.

- The second relation is defined between term distributions and is called $\Rightarrow^{\cdot}$; it models internal and *external* moves.

- Finally we define a third relation $\mapsto^{\cdot}$ between term distributions and *real numbers*, it captures the probability of a term distribution to accept a certain trace.

Furthermore we will sometimes use the relation $\to$ to indicate a single internal or external move. These three relations are defined inductively by the rules in figure

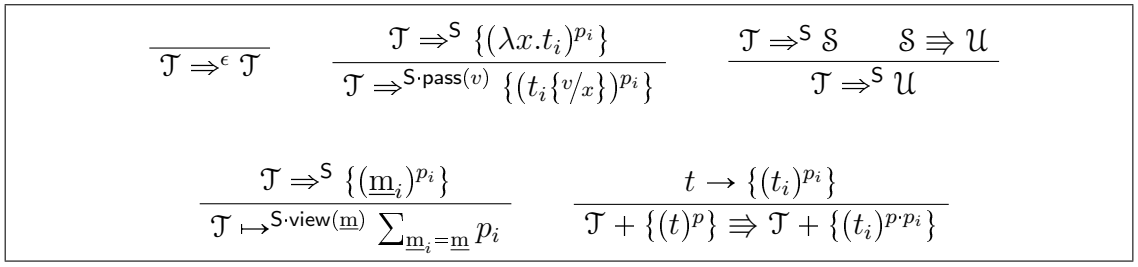

Figure 4.2: Binary Relations Rules

4.2.

The following gives basic, easy, results about the relations we have introduced:

Lemma 4.1 *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* $\Pr(\mathcal{T}, \mathsf{T})$.

Proof:

- Suppose that $\mathcal{T}$ is normal, i.e. all elements in the support are values, then we have $\mathcal{T} = \mathcal{D}$ and then the thesis.

- If $\mathcal{T}$ is not normal then there exists a set of indexes J such that $\{(t_j)^{p_j}\}_{j\in J} \subseteq \mathcal{T}$ aren't values. We know by a previous lemma that for all $j \in J$ there exists a unique $\mathcal{D}_j$, value distribution, such that $t_j \Downarrow \mathcal{D}_j$ in a finite number of steps. So, if we set $\mathcal{D} = \mathcal{T}\backslash\{(t_j)^{p_j}\}_{j\in J} + \sum_{j\in J} p_j \cdot \mathcal{D}_j$ we have $\mathcal{T} \Rrightarrow^* \mathcal{D}$ with $\mathcal{D}$ normal and we can say that for all $\mathcal{T}$ there exists $\mathcal{T}'$ normal such that $\mathcal{T} \Rightarrow^{\epsilon} \mathcal{T}'$.
  For example if we consider $\mathcal{T} = \{(\mathsf{case}_{\mathsf{Str}}(\mathsf{rand}, \underline{0}, \underline{1}, \underline{\epsilon}))^{\frac{1}{2}},\ (\underline{0})^{\frac{1}{2}}\}$ we have that $\mathsf{case}_{\mathsf{Str}}(\mathsf{rand}, \underline{0}, \underline{1}, \underline{\epsilon})$ is not a value, but we know:

$$\mathsf{case}_{\mathsf{Str}}(\mathsf{rand}, \underline{0}, \underline{1}, \underline{\epsilon}) \Downarrow \{(\underline{0})^{\frac{1}{2}},\ (\underline{1})^{\frac{1}{2}}\}$$

so if we set

$$\mathcal{D} = \{(\underline{0})^{\frac{1}{2}}\} + \frac{1}{2} \cdot \{(\underline{0})^{\frac{1}{2}},\ (\underline{1})^{\frac{1}{2}}\} = \{(\underline{0})^{\frac{3}{4}},\ (\underline{1})^{\frac{1}{4}}\}$$

we have $\mathcal{T} \Rrightarrow^* \mathcal{D}$, with $\mathcal{D}$ value distribution.

- Given $\mathsf{T} = \mathsf{S} \cdot \mathsf{pass}(v)$ we have by induction hypothesis that $\mathcal{T} \Rightarrow^{\mathsf{S}} \{(\lambda x.t_i)^{p_i}\}$. Then, by performing the action $\mathsf{pass}(v)$ we have $\mathcal{T} \Rightarrow^{\mathsf{S}\cdot\mathsf{pass}(v)} \{(t_i\{^v/_x\})^{p_i}\}$, but, by applying the previous point there exists $\mathcal{T}'$ normal such that $\{(t_i\{^v/_x\})^{p_i}\} \Rrightarrow^* \mathcal{T}'$ and then by the binary relations rules we have $\mathcal{T} \Rightarrow^{\mathsf{S}\cdot\mathsf{pass}(v)} \mathcal{T}'$ normal distribution.

- Suppose now that $\mathsf{T} = \mathsf{S} \cdot \mathsf{view}(\underline{\mathrm{m}})$ then we have by induction hypothesis $\mathcal{T} \Rightarrow^{\mathsf{S}} \mathcal{T}' = \{(\underline{\mathrm{m}}_i)^{p_i}\}$ with $\mathcal{T}'$ unique; so, if we perform the action $\mathsf{view}(\underline{\mathrm{m}})$ we have $\mathcal{T} \mapsto^{\mathsf{S}\cdot\mathsf{view}(\underline{\mathrm{m}})} p = \sum_{i;\underline{\mathrm{m}}_i=\underline{\mathrm{m}}} p_i$, that is unique by construction.

□

Now that we have defined the meaning of a term distribution to perform a trace we are able to give the definition of trace equivalence.

Definition 10 *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 $\simeq^{\mathsf{T}}$ is an equivalence relation.

Proposition 4.1 *Trace equivalence is an equivalence relation.*

Proof: We have to prove that trace equivalence is reflexive, symmetric and transitive.

- $\simeq^{\mathsf{T}}$ is *Reflexive:*
  For all $\mathcal{T}$ we have $\Pr(\mathcal{T}, \mathsf{T}) = p$ and p is unique. So for all T compatible, $\Pr(\mathcal{T}, \mathsf{T}) = \Pr(\mathcal{T}, \mathsf{T})$ and then $\mathcal{T} \simeq^{\mathsf{T}} \mathcal{T}$.

- $\simeq^{\mathsf{T}}$ is *Symmetric:*
  For all $\mathcal{T}, \mathcal{S}$, if $\mathcal{T} \simeq^{\mathsf{T}} \mathcal{S}$ for all T compatible, $\Pr(\mathcal{T}, \mathsf{T}) = \Pr(\mathcal{S}, \mathsf{T})$, then $\Pr(\mathcal{S}, \mathsf{T}) = \Pr(\mathcal{T}, \mathsf{T})$ and so $\mathcal{S} \simeq^{\mathsf{T}} \mathcal{T}$.

- $\simeq^{\mathsf{T}}$ is *Transitive:*
  For all $\mathcal{T}, \mathcal{S}, \mathcal{U}$ if $\mathcal{T} \simeq^{\mathsf{T}} \mathcal{S}, \mathcal{S} \simeq^{\mathsf{T}} \mathcal{U}$ then for all compatible traces T we have $\Pr(\mathcal{T}, \mathsf{T}) = \Pr(\mathcal{S}, \mathsf{T})$ and $\Pr(\mathcal{S}, \mathsf{T}) = \Pr(\mathcal{U}, \mathsf{T})$. But this means $\Pr(\mathcal{T}, \mathsf{T}) = \Pr(\mathcal{U}, \mathsf{T})$ and so $\mathcal{T} \simeq^{\mathsf{T}} \mathcal{U}$

□

We list now some of the properties of trace equivalence, that will be useful in the following.

Lemma 4.2 *Given two term distributions $\mathcal{T}, \mathcal{S}$ such that $\mathcal{T} \simeq^{\mathsf{T}} \mathcal{S}$ then we have:*

1. *If $\mathcal{T} \Rrightarrow \mathcal{T}'$ then $\mathcal{T}' \simeq^{\mathsf{T}} \mathcal{S}$.*
2. *If $\mathcal{T} \rightarrow^{\mathsf{pass}(v)} \mathcal{T}'$ and $\mathcal{S} \rightarrow^{\mathsf{pass}(v)} \mathcal{S}'$ then $\mathcal{T}' \simeq^{\mathsf{T}} \mathcal{S}'$*

Proof:

1. If $\mathcal{T} \Rrightarrow \mathcal{T}'$ then by the binary relations rules we have $\mathcal{T} \Rightarrow^{\epsilon} \mathcal{T}'$. But, for all traces T we have that $\Pr(\mathcal{T}, \mathsf{T}) = \Pr(\mathcal{T}, \epsilon \cdot \mathsf{T})$, this means:

   $$\Pr(\mathcal{T}, \mathsf{T}) = \Pr(\mathcal{T}, \epsilon \cdot \mathsf{T}) = \Pr(\mathcal{T}', \mathsf{T})$$

   and so we have that if $\mathcal{P} \Rrightarrow \mathcal{P}'$ then for all traces T:

   $$\Pr(\mathcal{T}', \mathsf{T}) = \Pr(\mathcal{T}, \mathsf{T}) = \Pr(\mathcal{S}, \mathsf{T})$$

   and so $\mathcal{T}' \simeq^{\mathsf{T}} \mathcal{S}$.

2. This point comes by the fact that we quantify over a smaller set of traces. Indeed if we suppose by contradiction that $\mathcal{T}', \mathcal{S}'$ are not trace equivalent then there exists a trace T such that $\Pr(\mathcal{T}', \mathsf{T}) \neq \Pr(\mathcal{S}', \mathsf{T})$, but if this is true then:

$$\Pr(\mathcal{T}, \mathsf{pass}(v) \cdot \mathsf{T}) = 1 \cdot \Pr(\mathcal{T}', \mathsf{T}) \neq 1 \cdot \Pr(\mathcal{S}', \mathsf{T}) = \Pr(\mathcal{S}, \mathsf{pass}(v) \cdot \mathsf{T})$$

so we have that $\mathcal{T}, \mathcal{S}$ are not trace equivalent and then a contradiction.

□

4.3 Full Abstraction

As we have seen, 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: this happens because the operational semantics we defined does not allow to observe how a term behaves *in a context* and so we have a lack of information that makes the proof impossible. Following [19], 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.

Definition 11 *A* context pair *is a pair of the form* $(C, \mathcal{T})$*, where the first term is a context* $C[\vdash \mathsf{A}] : \mathsf{B}$ *and the second is a term distribution such that all terms are of type* A*. We define a* (context) pair distribution $\mathcal{P} = \{(C_i, \mathcal{T}_i)^{p_i}\}$ *as a distribution over context pairs such that for all* i *we have* $\vdash C_i[\vdash \mathsf{A}] : \mathsf{B}$ *and* $\mathcal{T}_i : \mathsf{A}$*.*

We say that a pair distribution $\mathcal{P} = \{(C_i, \mathcal{T}_i)^{p_i}\}$is *normal* if for all i and for all $t \in \mathsf{S}(\mathcal{T}_i)$ we have that $C_i[t]$ is a value.

The purpose of using this approach based on pairs is to separate the reduction of the contexts and the evolution of the term distributions in order to observe that

the trace equivalence is preserved; by this reason we will observe how a pair $(C, \mathcal{T})$ evolves following a trace step-by-step, and so we give a one-step semantics, defined by the rules in figure 4.3. and a small-step semantics, by the rules in figure 4.4.

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

Lemma 4.3 *Given a context C and a term distribution $\mathcal{T}$ if $C[\mathcal{T}] \to \{(C_i[\mathcal{T}_i])^{p_i}\}$ then $(C, \mathcal{T}) \to \{(C_i, \mathcal{T}_i)^{p_i}\}$*

Proof: The proof is given by observing the possible reductions.

- Suppose C to be term t, so we have $C[\mathcal{T}] = t$. If it reduces $t \to \{(t_i)^{p_i}\}$, then we have by the one step rules $(t, \mathcal{T}) \to \{(t_i, \mathcal{T})^{p_i}\}$.

- If $C = [\cdot]$ then $C[\mathcal{T}] = \mathcal{T}$ so if $\mathcal{T}$ reduces, we have $\mathcal{T} \to \mathcal{T}'$, By the one step rules we have $([\cdot], \mathcal{T}) \to \{([\cdot], \mathcal{T}')^1\}$.

- If the context is in the form Cv, we have that if $C = \lambda x.C'$ then

$$C[\mathcal{T}]v \to \{(C'\{v/x\}[\mathcal{T}])^1\}$$

  Similarly:

$$(Cv, \mathcal{T}) \to \{(C'\{v/x\}, \mathcal{T})^1\}$$

  Otherwise if $C = [\cdot]$ then:

$$[\mathcal{T}]v \to \{(\mathcal{T}')^1\}$$

  but, similarly:

$$([\cdot]v, \mathcal{T}) \to \{([\cdot], \mathcal{T}')^1\}$$

- The other cases are similar, a simple application of the one step rules.

□

$$\frac{\mathcal{T} \to^{\mathsf{pass}(v)} \mathcal{T}'}{([\cdot], \mathcal{T}) \to^{\mathsf{pass}(v)} \{([\cdot], \mathcal{T}')^1\}} \qquad \frac{}{(\lambda x.C, \mathcal{T}) \to^{\mathsf{pass}(v)} \{(C\{^v/_x\}, \mathcal{T})^1\}}$$

$$\frac{}{(\underline{\mathrm{m}}, \mathcal{T}) \mapsto^{\mathsf{view}(\underline{\mathrm{m}})} 1} \qquad \frac{}{(\underline{\mathrm{m}}', \mathcal{T}) \mapsto^{\mathsf{view}(\underline{\mathrm{m}})} 0} \qquad \frac{\mathcal{T} = \{(\underline{\mathrm{m}}_j)^{p_j}\}}{([\cdot], \mathcal{T}) \to \{(\underline{\mathrm{m}}_j, \mathcal{T}_j)^{p_j}\}}$$

$$\frac{(C, \mathcal{T}) \to \{(C_i, \mathcal{T}_i)^{p_i}\}}{(\mathsf{0}(C), \mathcal{T}) \to \{(\mathsf{0}(C_i), \mathcal{T}_i)^{p_i}\}} \qquad \frac{(C, \mathcal{T}) \to \{(C_i, \mathcal{T}_i)^{p_i}\}}{(\mathsf{1}(C), \mathcal{T}) \to \{(\mathsf{1}(C_i), \mathcal{T}_i)^{p_i}\}} \qquad \frac{(C, \mathcal{T}) \to \{(C_i, \mathcal{T}_i)^{p_i}\}}{(\mathsf{tail}(C), \mathcal{T}) \to \{(\mathsf{tail}(C_i), \mathcal{T}_i)^{p_i}\}}$$

$$\frac{t \to \{(t_i)^{p_i}\}}{(t, \mathcal{T}) \to \{(t_i, \mathcal{T}_i)^{p_i}\}} \qquad \frac{\mathcal{T} \to \mathcal{T}'}{([\cdot], \mathcal{T}) \to \{([\cdot], \mathcal{T}')^1\}}$$

$$\frac{(C, \mathcal{T}) \to^{\mathsf{pass}(v)} \{(C', \mathcal{T}')^1\}}{(Cv, \mathcal{T}) \to \{(C', \mathcal{T}')^1\}}$$

$$\frac{(C, \mathcal{T}) \to \{(C_i, \mathcal{T}_i)^{p_i}\}}{(Ct, \mathcal{T}) \to \{(C_i t, \mathcal{T}_i)^{p_i}\}} \qquad \frac{t \to \{(t_i)^{p_i}\} \qquad C[v] \text{ value, } \forall v \in \mathsf{S}(\mathcal{T})}{(Ct, \mathcal{T}) \to \{(Ct_i, \mathcal{T}_i)^{p_i}\}}$$

$$\frac{(C, \mathcal{T}) \to \{(C_i, \mathcal{T}_i)^{p_i}\}}{(vC, \mathcal{T}) \to \{(vC_i, \mathcal{T}_i)^{p_i}\}} \qquad \frac{t \to \{(t_i)^{p_i}\}}{(tC, \mathcal{T}) \to \{(t_i C, \mathcal{T}_i)^{p_i}\}}$$

$$\frac{}{((\lambda x.C)v, \mathcal{T}) \to \{(C\{^v/_x\}, \mathcal{T})^1\}} \qquad \frac{(C, \mathcal{T}) \in \mathrm{V}}{((\lambda x.t)C, \mathcal{T}) \to \{(t\{^C/_x\}, \mathcal{T})^1\}}$$

$$\frac{t \to \{(t_i)^{p_i}\}}{(\mathsf{case}_{\mathsf{A}}(t, C_0, C_1, C_\epsilon), \mathcal{T}) \to \{(\mathsf{case}_{\mathsf{A}}(t_i, C_0, C_1, C_\epsilon), \mathcal{T}_i)^{p_i}\}}$$

$$\frac{(C, \mathcal{T}) \to \{(C_i, \mathcal{T}_i)^{p_i}\}}{(\mathsf{case}_{\mathsf{A}}(C, t_0, t_1, t_\epsilon), \mathcal{T}) \to \{(\mathsf{case}_{\mathsf{A}}(C_i, t_0, t_1, t_\epsilon), \mathcal{T}_i)^{p_i}\}}$$

$$\frac{}{(\mathsf{case}_{\mathsf{A}}(\underline{\mathsf{0}\mathrm{m}}, C_0, C_1, C_\epsilon), \mathcal{T}) \to \{(C_0, \mathcal{T})^1\}} \qquad \frac{}{(\mathsf{case}_{\mathsf{A}}(\underline{\mathsf{1}\mathrm{m}}, C_0, C_1, C_\epsilon), \mathcal{T}) \to \{(C_1, \mathcal{T})^1\}}$$

$$\frac{}{(\mathsf{case}_{\mathsf{A}}(\underline{\epsilon}, C_0, C_1, C_\epsilon), \mathcal{T}) \to \{(C_\epsilon, \mathcal{T})^1\}} \qquad \frac{(C, \mathcal{T}) \to \{(C_i, \mathcal{T}_i)^{p_i}\}}{(\mathsf{rec}_{\mathsf{A}}(C, t_0, t_1, t_\epsilon), \mathcal{T}) \to \{(\mathsf{rec}_{\mathsf{A}}(C_i, t_0, t_1, t_\epsilon))^{p_i}, \mathcal{T}_i)^{p_i}\}}$$

Figure 4.3: Context Pairs: One-Step Semantics

$$\frac{}{\mathcal{P} \Rightarrow^{\epsilon} \mathcal{P}} \qquad \frac{\mathcal{P} \Rightarrow^{\mathsf{S}} \mathcal{P}' \quad \mathcal{P}' \Rrightarrow \mathcal{P}''}{\mathcal{P} \Rightarrow^{\mathsf{S}} \mathcal{P}''}$$

$$\frac{\mathcal{P} \Rightarrow^{\mathsf{S}} \{(C_i, \mathcal{T}_i)^{p_i}\} \quad (C_i, \mathcal{T}_i) \rightarrow^{\mathsf{pass}(v)} \{(C_i', \mathcal{T}_i')^1\}}{\mathcal{P} \Rightarrow^{\mathsf{S} \cdot \mathsf{pass}(v)} \{(C_i', \mathcal{T}_i')^{p_i}\}}$$

$$\frac{\mathcal{P} \Rightarrow^{\mathsf{S}} \{(\underline{\mathrm{m}}_i, \mathcal{T}_i)^{p_i}\}}{\mathcal{P} \mapsto^{\mathsf{S} \cdot \mathsf{view}(\underline{\mathrm{m}})} \sum_{i; \underline{\mathrm{m}}_i = \underline{\mathrm{m}}} p_i} \qquad \frac{(C, \mathcal{T}) \rightarrow \{(C_i, \mathcal{T}_i)^{p_i}\}}{\mathcal{P} + \{(C, \mathcal{T})^p\} \Rrightarrow \mathcal{P} + \{(C_i, \mathcal{T}_i)^{p \cdot p_i}\}}$$

Figure 4.4: Small-Step Rules

Lemma 4.4 *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* $\Pr(C[\mathcal{T}], \mathsf{S}) = p$.

Proof:

- If $\mathsf{S} = \epsilon$, we know that if $C[\mathcal{T}] \Rightarrow^{\epsilon} \{(C_i[\mathcal{T}_i])^{p_i}\}$ also $(C, \mathcal{T}) \Rightarrow^{\epsilon} \{(C_i, \mathcal{T}_i)^{p_i}\}$ because the correspondence is preserved by internal reductions.

- If $\mathsf{S} = \mathsf{S}' \cdot \mathsf{pass}(v)$ then we have $C[\mathcal{T}] \Rightarrow^{\mathsf{S}'} \{(C_i[\mathcal{T}_i])^{p_i}\}$ and, by induction hypothesis $(C, \mathcal{T}) \Rightarrow^{\mathsf{S}'} \{(C_i, \mathcal{T}_i)^{p_i}\}$. For all i, if $C_i = [\cdot]$ then

  $$C_i[\mathcal{T}] = \mathcal{T}_i \rightarrow^{\mathsf{pass}(v)} \{(\mathcal{T}_i')^1\}$$

  but, similarly:

  $$(C_i, \mathcal{T}_i) = ([\cdot], \mathcal{T}_i) \rightarrow^{\mathsf{pass}(v)} \{([\cdot], \mathcal{T}_i')\}$$

  Otherwise, if $C_i[\mathcal{T}_i] = \lambda x.C_i'[\mathcal{T}_i]$ then

  $$\lambda x.C_i'[\mathcal{T}_i] \rightarrow^{\mathsf{pass}(v)} \{C_i'\{v/x\}[\mathcal{T}_i]^1\}$$

  Similarly:

  $$(\lambda x.C_i', \mathcal{T}_i) \rightarrow^{\mathsf{pass}(v)} \{(C_i'\{v/x\}, \mathcal{T}_i)^1\}$$

and so the thesis.

- If $\mathsf{S} = \mathsf{S}' \cdot \mathsf{view}(\underline{\mathrm{m}})$ with S' incomplete trace, by the previous point we have that $C[\mathcal{T}] \Rightarrow^{\mathsf{S}'} \{(\underline{\mathrm{m}}_i)^{p_i}\}$ and $(C, \mathcal{T}) \Rightarrow^{\mathsf{S}'} \{(\underline{\mathrm{m}}_i, \mathcal{T}_i)^{p_i}\}$. So we have

$$\Pr(C, \mathsf{S}) = \sum p_i \cdot (\underline{\mathrm{m}}_i[\mathcal{T}_i] \mapsto^{\mathsf{view}(\underline{\mathrm{m}})}) = \sum p_i \cdot \begin{array}{ll} 1, & \text{if } \underline{\mathrm{m}}_i = \underline{\mathrm{m}}; \\ 0, & \text{if } \underline{\mathrm{m}}_i \neq \underline{\mathrm{m}}; \end{array} =$$

$$= \sum p_i \cdot ((\underline{\mathrm{m}}_i, \mathcal{T}_i) \mapsto^{\mathsf{view}(\underline{\mathrm{m}})}) = (C, \mathcal{T}) \mapsto^{\mathsf{S}}$$

□

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 12 (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 $\mathcal{T}_i \simeq^{\mathsf{T}} \mathcal{S}_i$.*

The first observation about trace relatedness has to do with stability with respect to internal reduction; before going into the details of the proof we observe the different cases in which a couple $(C, \mathcal{T})$ reduces to a distribution $\{(C_i, \mathcal{T}_i)^{p_i}\}$.
In the following we analyze the four situations in which a couple reduces and the basic cases.

1. **Term Distribution Reduction**
   This is the case in which we have a reduction inside the hole without interaction between the term distribution and the context, that is the case in which the

derivation starts from the premise $\mathcal{T} \to \mathcal{T}'$:

$$\frac{\mathcal{T} \to \mathcal{T}'}{([\cdot], \mathcal{T}) \to \{([\cdot], \mathcal{T}')^1\}} \qquad \frac{\mathcal{T} \to \mathcal{T}'}{(v[\cdot], \mathcal{T}) \to \{(v[\cdot], \mathcal{T}')^1\}}$$

$$\frac{\mathcal{T} \to \mathcal{T}'}{([\cdot]t, \mathcal{T}) \to \{([\cdot]t, \mathcal{T}')^1\}} \qquad \frac{\mathcal{T} \to \mathcal{T}'}{(\mathsf{case_A}([\cdot], t_0, t_1, t_\epsilon), \mathcal{T}) \to \{(\mathsf{case_A}([\cdot], t_0, t_1, t_\epsilon), \mathcal{T}')^1\}}$$

$$\frac{\mathcal{T} \to \mathcal{T}'}{(\mathsf{rec_A}([\cdot], t_0, t_1, t_\epsilon), \mathcal{T}) \to \{(\mathsf{rec_A}([\cdot], t_0, t_1, t_\epsilon), \mathcal{T}')^1\}}$$

2. **Mixed Reduction**

   This is the case in which the context passes some value v to the term distribution. These derivations start from the premise $\mathcal{T} \to^{\mathsf{pass}(v)} \mathcal{T}'$.

$$\frac{\mathcal{T} \to^{\mathsf{pass}(v)} \mathcal{T}'}{([\cdot]v, \mathcal{T}) \to \{([\cdot], \mathcal{T}')^1\}} \qquad \frac{\mathcal{T} \to^{\mathsf{pass}(v)} \mathcal{T}'}{(\mathsf{case_A}([\cdot]v, t_0, t_1, t_\epsilon), \mathcal{T}) \to \{(\mathsf{case_A}([\cdot], t_0, t_1, t_\epsilon), \mathcal{T}')^1\}}$$

$$\frac{\mathcal{T} \to^{\mathsf{pass}(v)} \mathcal{T}'}{(\mathsf{rec_A}([\cdot]v, t_0, t_1, t_\epsilon), \mathcal{T}) \to \{(\mathsf{rec_A}([\cdot], t_0, t_1, t_\epsilon), \mathcal{T}')^1\}}$$

3. **Reduction Without Observation**

   This is the case in which the context reduces without interacting with the term distribution. These reductions starts from the premise $t \to \{t_i^{p_i}\}$.

$$\frac{t \to \{t_i^{p_i}\}}{(t, \mathcal{T}) \to \{(t_i, \mathcal{T})^{p_i}\}} \qquad \frac{t \to \{t_i^{p_i}\}}{(tC, \mathcal{T}) \to \{(t_iC, \mathcal{T})^{p_i}\}} \qquad \frac{t \to \{t_i^{p_i}\} \qquad C \text{ is a value}}{(Ct, \mathcal{T}) \to \{(Ct_i, \mathcal{T})^{p_i}\}}$$

$$\frac{t \to \{t_i^{p_i}\}}{(\mathsf{case_A}(t, C_0, C_1, C_\epsilon), \mathcal{T}) \to \{(\mathsf{case_A}(t_i, C_0, C_1, C_\epsilon), \mathcal{T})^{p_i}\}}$$

4. **Reduction with Observation**

   This is the case in which the term distribution is a string distribution and so

the context is allowed to observe it. These reductions start from the premise $\mathcal{T} = \{\underline{\mathrm{m}}_j^{p_j}\}$.

$$\frac{\mathcal{T} = \{\underline{\mathrm{m}}_j^{p_j}\}}{([\cdot], \mathcal{T}) \to \{(\underline{\mathrm{m}}_j, \mathcal{T})^{p_j}\}} \qquad \frac{\mathcal{T} = \{\underline{\mathrm{m}}_j^{p_j}\}}{(\mathsf{0}([\cdot]), \mathcal{T}) \to \{(\mathsf{0}(\underline{\mathrm{m}}_j), \mathcal{T})^{p_j}\}}$$

$$\frac{\mathcal{T} = \{\underline{\mathrm{m}}_j^{p_j}\}}{(\mathsf{1}([\cdot]), \mathcal{T}) \to \{(\mathsf{1}(\underline{\mathrm{m}}_j), \mathcal{T})^{p_j}\}} \qquad \frac{\mathcal{T} = \{\underline{\mathrm{m}}_j^{p_j}\}}{(\mathsf{tail}([\cdot]), \mathcal{T}) \to \{(\mathsf{tail}(\underline{\mathrm{m}}_j), \mathcal{T})^{p_j}\}}$$

$$\frac{\mathcal{T} = \{\underline{\mathrm{m}}_j^{p_j}\}}{(\lambda x.t[\cdot], \mathcal{T}) \to \{(\lambda x.t\ \underline{\mathrm{m}}_j, \mathcal{T})^{p_j}\}}$$

$$\frac{\mathcal{T} = \{\underline{\mathrm{m}}_j^{p_j}\}}{(\mathsf{case_A}([\cdot], t_0, t_1, t_\epsilon), \mathcal{T}) \to \{(\mathsf{case_A}(\underline{\mathrm{m}}_j, t_0, t_1, t_\epsilon), \mathcal{T})^{p_j}\}}$$

$$\frac{\mathcal{T} = \{\underline{\mathrm{m}}_j^{p_j}\}}{(\mathsf{rec_A}([\cdot], t_0, t_1, t_\epsilon), \mathcal{T}) \to \{(\mathsf{rec_A}(\underline{\mathrm{m}}_j, t_0, t_1, t_\epsilon), \mathcal{T})^{p_j}\}}$$

Notice that, after the observation, the context becomes a family of simple terms.

Now we can go into details and prove that relatedness between two pair distributions is preserved by internal reductions.

Lemma 4.5 (Internal Stability) *Let $\mathcal{P}, \mathcal{Q}$ two pair distributions such that $\mathcal{P} \triangledown \mathcal{Q}$, then if $\mathcal{P} \Rrightarrow \mathcal{P}'$ there exists $\mathcal{Q}'$ such that $\mathcal{Q} \Rightarrow^{\epsilon} \mathcal{Q}'$ and $\mathcal{P}' \triangledown \mathcal{Q}'$.*

Proof: Let's see all the possible reductions of a pair distribution $\mathcal{P}$

1. **Term distribution reduction**, $(C, \mathcal{T}) \to \{(C, \mathcal{T}')^1\}$.
   By definition we know that, given $(C, \mathcal{T})^p \in \mathcal{P}$, there exists $(C, \mathcal{S})^p \in \mathcal{Q}$ such that $\mathcal{T} \simeq^{\mathsf{T}} \mathcal{S}$. Then by the trace equivalence properties we know $\mathcal{T}' \simeq^{\mathsf{T}} \mathcal{S}$, so we have:
   $$\mathcal{P} \Rrightarrow \mathcal{P}' = \mathcal{P} \backslash \{(C, \mathcal{T})^p\} \cup \{(C, \mathcal{T}')^p\}$$
   and obviously $\mathcal{P}' \triangledown \mathcal{Q}$

2. **Mixed Reduction** $(C, \mathcal{T}) \to \{(C', \mathcal{T}')^1\}$
   This is the case in which the context passes a value to the hole, then we have a transition with probability 1. Suppose $\mathcal{P} \ni (C, \mathcal{T})^p \to \{(C', \mathcal{T}')^p\}$ then there exists $\mathcal{Q} \ni (C, \mathcal{S})^p \to \{(C', \mathcal{S}')^p\}$, because the context are the same by definition and $\mathcal{T}' \simeq^{\mathsf{T}} \mathcal{S}'$ by the trace equivalence properties. Then we have:

$$\mathcal{P} \Rrightarrow \mathcal{P}' = \mathcal{P} \backslash \{(C, \mathcal{T})^p\} \cup \{(C', \mathcal{T}')^p\}$$

$$\mathcal{Q} \Rrightarrow \mathcal{Q}' = \mathcal{Q} \backslash \{(C, \mathcal{S})^p\} \cup \{(C', \mathcal{S}')^p\}$$

   with $\mathcal{P}' \triangledown \mathcal{Q}'$.

3. **Reduction without observation** $(C, \mathcal{T}) \to \{(C_i, \mathcal{T})^{p_i}\}$.
   By definition we know that there exists $(C, \mathcal{S})^p \in \mathcal{Q}$ that reduces the same way, so we have:

$$\mathcal{P} \Rrightarrow \mathcal{P}' = \mathcal{P} \backslash \{(C, \mathcal{T})^p\} \cup \{(C_i, \mathcal{T})^{p \cdot p_i}\}$$

$$\mathcal{Q} \Rrightarrow \mathcal{Q}' = \mathcal{Q} \backslash \{(C, \mathcal{S})^p\} \cup \{(C_i, \mathcal{S})^{p \cdot p_i}\}$$

   and obviously $\mathcal{P}' \triangledown \mathcal{Q}'$.

4. **Observation Reduction** $(C, \mathcal{T}) \to \{(t_j, \mathcal{T})^{p_j}\}$.
   Suppose that $\mathcal{T} = \{(\underline{\mathrm{m}}_j)^{p_j}\}$, then we can also suppose that $\mathcal{S} = \{\underline{\mathrm{m}}_j^{q_j}\}$ and $\mathcal{T} \simeq^{\mathsf{T}} \mathcal{S}$ (Otherwise we have $\mathcal{S} \to^* \mathcal{S}' = \{\underline{\mathrm{m}}_j^{q_j}\}$ with $\mathcal{T} \simeq^{\mathsf{T}} \mathcal{S}'$ by the trace equivalence properties). Then we have:

$$\mathcal{P} \Rrightarrow \mathcal{P} \backslash \{(C, \mathcal{T})^p\} \cup \{(t_j, \mathcal{T}_j)^{p \cdot p_j}\}$$

$$\mathcal{Q} \Rrightarrow \mathcal{Q} \backslash \{(C, \mathcal{S})^p\} \cup \{(t_j, \mathcal{S}_j)^{p \cdot q_j}\}$$

   but $\mathcal{T} \simeq^{\mathsf{T}} \mathcal{S}$ so we have $p_j = q_j$ for all j and so the thesis.

□

Once Internal Stability is proved, and since the relation $\Rrightarrow$ can be proved to be strongly normalizing also for context pair distributions, one gets that:

Lemma 4.6 (Bisimulation, Internally) *If $\mathcal{P}, \mathcal{Q}$ are pair distributions, with $\mathcal{P} \triangledown \mathcal{Q}$ then there are $\mathcal{P}', \mathcal{Q}'$ normal distributions such that $\mathcal{P} \Rightarrow^{\epsilon} \mathcal{P}'$, $\mathcal{Q} \Rightarrow^{\epsilon} \mathcal{Q}'$ and $\mathcal{P}' \triangledown \mathcal{Q}'$.*

Proof: The proof comes from the fact that, given $\mathcal{P}$ if it is not normal, there is $\mathcal{P}'$ normal such that $\mathcal{P} \Rrightarrow^{*} \mathcal{P}'$, and by the previous lemma we have $\mathcal{P}' \triangledown \mathcal{Q}$. Then if $\mathcal{Q}$ isn't normal we can repeat the procedure and get $\mathcal{Q}'$ such that $\mathcal{Q} \Rrightarrow^{*} \mathcal{Q}'$ and $\mathcal{P}' \triangledown \mathcal{Q}'$. □

The next step consists in proving that context pair distributions which are trace related are not only bisimilar as for internal reduction, but also for external reduction:

Lemma 4.7 (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 $\mathcal{P} \Rightarrow^{\mathsf{S}} \mathcal{R}$, with $\mathcal{R}$ normal distribution, then $\mathcal{Q} \Rightarrow^{\mathsf{S}} \mathcal{W}$, where $\mathcal{R} \triangledown \mathcal{W}$ and $\mathcal{W}$ is a normal distribution too.*
2. *If $\mathcal{P} \mapsto^{\mathsf{S}} p$ and $\mathcal{Q} \mapsto^{\mathsf{S}} q$ then $p = q$.*

Proof: We act by induction on the length of S.

- If $\mathsf{S} = \epsilon$ then by lemma 4.6 we get the thesis. Suppose now $\mathsf{S} = \mathsf{S}' \cdot \mathsf{pass}(v)$ then we have by induction hypothesis: $\mathcal{P} \Rightarrow^{\mathsf{S}'} \{(C_i, \mathcal{T}_i)^{p_i}\}_{i \in I}$ and $\mathcal{Q} \Rightarrow^{\mathsf{S}'} \{(C_i, \mathcal{S}_i)^{p_i}\}_{i \in I}$ with $\mathcal{T}_i \simeq^{\mathsf{T}} \mathcal{S}_i$ for all $i \in I$ and the two pair distributions normal. But, by the one-step rules we have only two possible derivations for an action $\mathsf{pass}(v)$:

$$\frac{}{(\lambda x.C, \mathcal{T}) \rightarrow^{\mathsf{pass}(v)} \{(C\{^{v}/_{x}\}, \mathcal{T})^{1}\}} \qquad \frac{\mathcal{T} \rightarrow^{\mathsf{pass}(v)} \mathcal{T}'}{([\cdot], \mathcal{T}) \rightarrow^{\mathsf{pass}(v)} \{([\cdot], \mathcal{T}')^{1}\}}$$

  So if we set $J = \{j \in I \mid C_j = \lambda x.C'_j\}, K = \{k \in I \mid C_i = [\cdot]\}$ we have:

$$\mathcal{P} \Rightarrow^{\mathsf{S}'} \{(\lambda x.C'_j, \mathcal{T}_j)^{p_j}\} + \{([\cdot], \mathcal{T}_k)^{p_k}\} \qquad \mathcal{Q} \Rightarrow^{\mathsf{S}'} \{(\lambda x.C'_j, \mathcal{S}_j)^{p_j}\} + \{([\cdot], \mathcal{S}_k)^{p_k}\}$$

  At this point, if $\mathcal{T}_k \rightarrow^{\mathsf{pass}(v)} \mathcal{T}'_k$ and $\mathcal{S}_k \rightarrow^{\mathsf{pass}(v)} \mathcal{S}'_k$ we know $\mathcal{T}'_k \simeq^{\mathsf{T}} \mathcal{S}'_k$ for all k, so by using the one step rule we set:

$$\mathcal{P}' = \{(C_j\{^{v}/_{x}\}, \mathcal{T}_j)^{p_j}\} + \{([\cdot], \mathcal{T}'_k)^{p_k}\} \qquad \mathcal{Q}' = \{(C_j\{^{v}/_{x}\}, \mathcal{S}_j)^{p_j}\} + \{([\cdot], \mathcal{S}'_k)^{p_k}\}$$

and we have $\mathcal{P} \Rightarrow^{\mathsf{S}'\cdot\mathsf{pass}(v)} \mathcal{P}'$, $\mathcal{Q} \Rightarrow^{\mathsf{S}\cdot\mathsf{pass}(v)} \mathcal{Q}'$ with $\mathcal{P}''\triangledown\mathcal{Q}''$; and so by applying lemma 4.6 we get the (1) thesis.

- Suppose now $\mathsf{S} = \mathsf{S}' \cdot \mathsf{view}(\underline{\mathrm{m}})$.
  By induction we know that $\mathcal{P} \Rightarrow^{\mathsf{S}'} \{(\underline{\mathrm{m}}_i, \mathcal{T}_i)^{p_i}\}, \mathcal{Q} \Rightarrow^{\mathsf{S}'} \{(\underline{\mathrm{m}}_i, \mathcal{S}_i)^{p_i}\}$ with $\mathcal{T}_i \simeq^{\mathsf{T}} \mathcal{S}_i$ for all $i \in I$ and that the two pair distributions are normal. So we know:

  $$\mathcal{P} \Rightarrow^{\mathsf{S}'} \{(\underline{\mathrm{m}}_j, \mathcal{T}_j)^{p_j}\} \qquad \mathcal{Q} \Rightarrow^{\mathsf{S}'} \{(\underline{\mathrm{m}}_j, \mathcal{S}_j)^{p_j}\}$$

  So we have:

  $$\mathcal{P} \mapsto^{\mathsf{S}'\cdot\mathsf{view}(\underline{\mathrm{m}})} p = \sum_{\underline{\mathrm{m}}_j = \underline{\mathrm{m}}} p_j \qquad \mathcal{Q} \mapsto^{\mathsf{S}'\cdot\mathsf{view}(\underline{\mathrm{m}})} q = \sum_{\underline{\mathrm{m}}_j = \underline{\mathrm{m}}} p_j$$

  and so $p = q$

□

Lemma 4.8 *Given two term distributions $\mathcal{T}, \mathcal{S}$ such that $\mathcal{T} \simeq^{\mathsf{T}} \mathcal{S}$, then for all contexts C, for all traces S we have:* $\Pr(C[\mathcal{T}], \mathsf{S}) = \Pr(C[\mathcal{S}], \mathsf{S})$

Proof:

- If the trace S doesn't end with the action $\mathsf{view}(\cdot)$ then $\Pr(C[\mathcal{T}], \mathsf{S}) = 1 = \Pr(C[\mathcal{S}], \mathsf{S})$.
- Otherwise we know that $(C, \mathcal{T}) \mapsto^{\mathsf{S}} p$, we can write $\Pr((C, \mathcal{T}), \mathsf{S}) = p$, and by Lemma 4.7 we know $(C, \mathcal{S}) \mapsto^{\mathsf{S}} p$. But by Lemma 4.4 we know $\Pr(C[\mathcal{T}], \mathsf{S}) = \Pr((C, \mathcal{T}), \mathsf{S}) = \Pr((C, \mathcal{S}), \mathsf{S}) = \Pr(C[\mathcal{S}], \mathsf{S})$ and then the thesis.

□

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

Theorem 4.1 *Trace equivalence is a congruence.*

Proof: We have to prove that, given two terms t, s such that $t \simeq^{\mathsf{T}} s$ then for all contexts C, we have that $C[t] \simeq^{\mathsf{T}} C[s]$, i.e., for all traces S we have $\Pr(C[t], \mathsf{S}) = \Pr(C[s], \mathsf{S})$. But by Lemma 4.4 and Lemma 4.7 we have, indeed, that $\Pr(C[t], \mathsf{S}) = \Pr((C, \{t^1\}), \mathsf{S}) = \Pr((C, \{s^1\}, \mathsf{S}) = \Pr(C[s], \mathsf{S})$, because the two pair distributions $\{(C, \{t^1\})^1\}$ and $\{((C, \{s^1\}))^1\}$ are trace-related. □

Corollary 4.1 (Soundness) *Trace equivalence is included into context equivalence.*

Proof: If $t \simeq^{\mathsf{T}} s$, then by the previous theorem we have that for all contexts C we have $C[t] \simeq^{\mathsf{T}} C[s]$ and this means that if we choose a trace $\mathsf{T} = \mathsf{view}(\underline{\epsilon})$ then we have $[\![C[t]]\!](\underline{\epsilon}) = \Pr(C[t], \mathsf{view}(\underline{\epsilon})) = \Pr(C[s], \mathsf{view}(\underline{\epsilon})) = [\![C[s]]\!](\underline{\epsilon})$, and so the thesis. □

Theorem 4.2 (Full Abstraction) *Context equivalence coincides with trace equivalence*

Proof: For any admissible trace T for A, there is a context $C_{\mathsf{T}}[\cdot]$ such that $\Pr(t, \mathsf{T}) = [\![C_{\mathsf{T}}[t]]\!](\underline{\epsilon})$, which can be proved by induction on the structure of A. □

4.4 Typed Relations and 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 [30, 17].

Applicative bisimulation is a typed relation defined on a Labeled Transition System, so in this section we will start by introducing the concept of typed relation and some useful propertis, then we will present the Labeled Transition System on which we define the applicative bisimulation and we will show that it is a sound method to prove that two terms are context equivalent (even if the context is not linear), but not complete.

Definition 13 *We define a* typed relation $\mathcal{R}$ *as a family* $\mathcal{R} = \{\mathcal{R}^{\mathsf{A}}_{\Gamma}\}_{\Gamma,\mathsf{A}}$*, where each relation* $\mathcal{R}^{\mathsf{A}}_{\Gamma}$ *is a binary relation on* $\mathrm{T}^{\mathsf{A}}_{\Gamma}$*.*
Given two terms $t, s \in \mathrm{T}^{\mathsf{A}}_{\Gamma}$ *we write* $\Gamma \vdash t\ \mathcal{R}\ s : \mathsf{A}$ *to say that they are in relation* $\mathcal{R}^{\mathsf{A}}_{\Gamma}$*.*

Definition 14 *Let* $\mathcal{R}$ *be a typed relation, we say that* $\mathcal{R}$ *is:*

- Reflexive*: If for all* $t \in \mathrm{T}^{\mathsf{A}}_{\Gamma}$ *we have* $\Gamma \vdash t\ \mathcal{R}\ t : \mathsf{A}$*.*
- Symmetric*: If* $\Gamma \vdash t\ \mathcal{R}\ s : \mathsf{A}$ *implies* $\Gamma \vdash s\ \mathcal{R}\ t : \mathsf{A}$*.*
- Transitive*: If* $\Gamma \vdash t\ \mathcal{R}\ s : \mathsf{A}$ *and* $\Gamma \vdash s\ \mathcal{R}\ r : \mathsf{A}$ *imply* $\Gamma \vdash t\ \mathcal{R}\ r : \mathsf{A}$

The other concept which plays a key role in typed relations and then in this work is *compatibility*.

Definition 15 *We define a typed relation* $\mathcal{R}$ compatible *if it satisfies the the following conditions:*

1. *For all* $x : \mathsf{aA} \in \Gamma$*:*
$$\Gamma \vdash x\ \mathcal{R}\ x : \mathsf{aA}$$
2. *For all* t, s, x
$$\Gamma, x : \mathsf{aA} \vdash t\ \mathcal{R}\ s : \mathsf{B}\ \Rightarrow\ \Gamma \vdash \lambda x.t\ \mathcal{R}\ \lambda x.s : \mathsf{aA} \to \mathsf{B}$$

3. For all t_1, t_2, s_1, s_2*:*

$$\Gamma; \Delta_1 \vdash t_1 \ \mathcal{R} \ t_2 : \mathsf{aA} \to \mathsf{B} \ \wedge \ \Gamma; \Delta_2 \vdash s_1 \ \mathcal{R} \ s_2 : \mathsf{A} \ \Rightarrow$$
$$\Rightarrow \ \Gamma; \Delta_1, \Delta_2 \vdash t_1 s_1 \ \mathcal{R} \ t_2 s_2 : \mathsf{B}$$

4. For all $t, t_0, t_1, t_\varepsilon, s, s_0, s_1, s_\varepsilon$*:*

$$\Gamma; \Delta \vdash t \ \mathcal{R} \ s : \mathsf{Str} \ \wedge \ \Gamma; \Delta_0 \vdash t_0 \ \mathcal{R} \ s_0 : \mathsf{A}$$
$$\Gamma; \Delta_1 \vdash t_1 \ \mathcal{R} \ s_1 : \mathsf{A} \ \wedge \ \Gamma; \Delta_\varepsilon \vdash t_\varepsilon \ \mathcal{R} \ s_\varepsilon : \mathsf{A} \ \Rightarrow$$
$$\Rightarrow \ \Gamma; \Delta, \Delta_0, \Delta_1, \Delta_\varepsilon \vdash \mathsf{case}_\mathsf{A}(t, t_0, t_1, t_\varepsilon) \ \mathcal{R} \ \mathsf{case}_\mathsf{A}(t, t_0, t_1, t_\varepsilon)$$

5. For all $t, t_0, t_1, t_\varepsilon, s, s_0, s_1, s_\varepsilon$*:*

$$\Gamma_1; \Delta_1 \vdash t \ \mathcal{R} \ s : \mathsf{Str} \ \wedge \ \Gamma_1, \Gamma_2 \vdash t_0 \ \mathcal{R} \ s_0 : \Box\mathsf{Str} \to \blacksquare\mathsf{A} \to \mathsf{A} \ \wedge$$
$$\Gamma_1, \Gamma_3 \vdash t_1 \ \mathcal{R} \ s_1 : \Box\mathsf{Str} \to \blacksquare\mathsf{A} \to \mathsf{A} \ \wedge \ \Gamma_1, \Gamma_2, \Gamma_3; \Delta_2 \vdash t_\varepsilon \ \mathcal{R} \ s_\varepsilon : \mathsf{A} \ \Rightarrow$$
$$\Rightarrow \ \Gamma_1, \Gamma_2, \Gamma_3; \Delta_1, \Delta_2 \vdash \mathsf{rec}_\mathsf{A}(t, t_0, t_1, t_\varepsilon) \ \mathcal{R} \ \mathsf{rec}_\mathsf{A}(t, t_0, t_1, t_\varepsilon)$$

A relation $\mathcal{R}$ is said a *precongruence* relation if it is transitive and compatible; if a precongruence relation is symmetric we call it a *congruence* relation.

Another important concept when we talk about binary relations is the *open extension* of a relation $\mathcal{R}$; in order to define it we need to introduce the Γ-*closure* of a term.

Definition 16 *Given a typing context* $\Gamma = x_1 : \mathsf{a}_1\mathsf{A}_1, ..., x_n : \mathsf{a}_n\mathsf{A}_n$*, we call a map* $\xi : \{x_i\}_{i=1,...,n} \to \{\mathrm{V}^{\mathsf{A}_i}\}_{i=1,...,n}$ *that assigns a value* $v_i \in \mathrm{V}^{\mathsf{A}_i}$ *to each variable* x_i *a* Γ-closure*.*
Given a term t *such that* $\Gamma \vdash t : \mathsf{A}$ *then we write* $t\xi$ *to indicate the term obtained by subsitituting the free varibles of* t *by the* Γ*-closure* ξ*.*

Definition 17 *If* $\mathcal{R}$ *is a typed binary relation, we define the* open extension $\mathcal{R}_\circ$*, by saying that two terms* t, s*, such that* $\Gamma \vdash t, s : \mathsf{A}$ *are in relation* $\mathcal{R}_\circ$*, we write* $t \ \mathcal{R}_\circ \ s$*, iff for all* Γ*-closures* ξ *we have* $\vdash t\xi \ \mathcal{R} \ s\xi : \mathsf{A}$*.*

As we said before, applicative bisimulation is defined on a Labeled Transition System, so now we introduce the LTS we choose: the Labeled Markov Chain.

Definition 18 *We define a Labeled Markov Chain as a triple* $\mathcal{M} = (\mathrm{S}, \mathsf{L}, \mathcal{P})$ *where:*

- $\mathrm{S} = \{(t, \mathsf{A}) \mid t \in \mathrm{T}^{\mathsf{A}}\} \biguplus \{(v, \mathsf{A}) \mid v \in \mathrm{V}^{\mathsf{A}}\}$ *is the set of states and it is composed by the disjoint union of the set of terms and values coupled with their type.*
- $\mathsf{L} = \{\mathsf{eval}, \mathsf{pass}(\cdot), \mathsf{view}(\cdot)\}$ *is the set of labels. The labels* $\mathsf{pass}(\cdot), \mathsf{view}(\cdot)$ *represent the same concept we used in traces, the* eval *label is the evaluation of a term to a value (It will be more clear when we will define the probability measure* $\mathcal{P}$*).*
- $\mathcal{P} : (\mathrm{S}, \mathrm{A}) \times \mathsf{L} \times (\mathrm{S}, \mathrm{A}) \to \mathbb{R}_{[0,1]}$ *is a probability measure; it returns the probability to pass from a state of the Markov chain to another by a certain label. It is defined as follows:*

$$\mathcal{P}((t, \mathsf{A}), \mathsf{eval}, (v, \mathsf{A})) = [\![t]\!](v)$$
$$\mathcal{P}((\lambda x.t, \mathsf{aA} \to \mathsf{B}), \mathsf{pass}(v), (t\{v/x\}, \mathsf{B})) = 1$$
$$\mathcal{P}((\underline{\mathrm{m}}, \mathsf{Str}), \mathsf{view}(\underline{\mathrm{m}}), (\underline{\mathrm{m}}, \mathsf{Str})) = 1$$
$$\mathcal{P}((\underline{\mathrm{m}}, \mathsf{Str}), \mathsf{view}(\underline{\mathrm{m}})', (\underline{\mathrm{m}}, \mathsf{Str})) = 0$$

Now we are able to give the definition of *applicative bisimulation.*

Definition 19 *Given a Labeled Markov Chain* $\mathcal{M} = (\mathrm{S}, \mathsf{L}, \mathcal{P})$ *a* probabilistic applicative bisimulation *is an equivalence relation* $\mathcal{R}$ *between the states of the Markov Chain such that, given two states* t, s *we have* $t \ \mathcal{R} \ s : \mathsf{A}$ *if and only if for each equivalence class* E *modulo* $\mathcal{R}$ *we have:*

$$\mathcal{P}((t, \mathsf{A}), l, E) = \mathcal{P}((s, \mathsf{A}), l, E)$$

We say that two terms are bisimilar if there exists a bisimulation between them.

We give an alternative definition of applicative bisimulation which is equivalent to the previous one but it will be more useful in the congruence proof.

Definition 20 *A* probabilistic applicative bisimulation *is defined to be any type-indexed family of relations* $\{\mathcal{R}_{\mathsf{A}}\}_{\mathsf{A}\in\mathsf{A}}$ *such that for each* A, $\mathcal{R}_{\mathsf{A}}$ *is an equivalence relation over the set of closed terms of type* A, *and moreover the following holds:*

- *If* $t\ \mathcal{R}_{\mathsf{A}}\ s$, *then for every equivalence class* E *modulo the relation* $\mathcal{R}_{\mathsf{A}}$, *it holds that* $[\![t]\!](E) = [\![s]\!](E)$, *where* $[\![t]\!](E) = \sum_{v\in E}[\![t]\!](v)$.
- *If* $(\lambda x.t)\ \mathcal{R}_{\mathsf{aA}\to\mathsf{B}}\ (\lambda x.s)$, *then for every closed value* v *of type* A, *it holds that* $(t\{^v/_x\})\ \mathcal{R}_{\mathsf{B}}\ (s\{^v/_x\})$.
- *If* $\underline{\mathrm{m}}\ \mathcal{R}_{\mathsf{Str}}\ \underline{\mathrm{n}}$, *then* $\underline{\mathrm{m}} = \underline{\mathrm{n}}$.

It is possible to prove that the greatest applicative bisimulation exists [30, 17], it consists of the union (at any type) of all bisimulation relations and it is denoted with $\sim$. We call $\sim$ (applicative)*bisimilarity* and we can generalize it to a relation $\sim_\circ$ on open terms by the usual open extension.

How do we prove that bisimilarity is included in context equivalence? We prove that $\sim_\circ$ is a congruence by using the so called *Howe's method*, a general way for establishing congruence properties of a relation. Given a binary relation $\mathcal{R}$, Howe's method consists in defining a lifting of this relation $\mathcal{R}^H$, which is easy to prove that is compatible; it is simple to see that $\mathcal{R} \subseteq \mathcal{R}^H$ so, by showing that $\mathcal{R}^H \subseteq \mathcal{R}$ we will prove that $\mathcal{R} = \mathcal{R}^H$ and so if $\mathcal{R}^H$ is a congruence also $\mathcal{R}$ is.

Definition 21 *We define* Howe's lifting $\sim_\circ^H$ *of the bisimilarity* $\sim_\circ$ *by the rules in the following:*

$$\frac{\underline{\mathrm{m}} \sim_\circ t}{\underline{\mathrm{m}} \sim_\circ^H t} \qquad \frac{x \sim_\circ t}{x \sim_\circ^H t} \qquad \frac{t \sim_\circ^H s \quad \lambda x.s \sim r}{\lambda x.t \sim_\circ^H r} \qquad \frac{t_1 \sim_\circ^H s_1 \quad t_2 \sim_\circ^H s_2 \quad s_1 s_2 \sim r}{t_1 t_2 \sim_\circ^H r}$$

$$\frac{\begin{array}{ll} t \sim_\circ^H s & \\ t_0 \sim_\circ^H s_0 & t_\epsilon \sim_\circ^H s_\epsilon \\ t_1 \sim_\circ^H s_1 & \mathsf{case}_{\mathsf{A}}(s, s_0, s_1, s_\epsilon) \sim r \end{array}}{\mathsf{case}_{\mathsf{A}}(t, t_0, t_1, t_\epsilon) \sim_\circ^H r} \qquad \frac{\begin{array}{ll} t \sim_\circ^H s & \\ t_0 \sim_\circ^H s_0 & t_\epsilon \sim_\circ^H s_\epsilon \\ t_1 \sim_\circ^H s_1 & \mathsf{rec}_{\mathsf{A}}(s, s_0, s_1, s_\epsilon) \sim r \end{array}}{\mathsf{rec}_{\mathsf{A}}(t, t_0, t_1, t_\epsilon) \sim_\circ^H r}$$

It is easy to prove that for every binary relation $\mathcal{R}$ in RSLR the following results are valid.

Lemma 4.9 *If $\mathcal{R}$ is reflexive, then $\mathcal{R}^H$ is compatible. Moreover if $\Gamma \vdash t\ \mathcal{R}\ s : \mathsf{A}$ then $\Gamma \vdash t\ \mathcal{R}^H\ s : \mathsf{A}$ and so $\mathcal{R}^H$ is reflexive.*

Proof: This is proved by induction on the structure of t. □

Definition 22 *We define $(\mathcal{R}_\circ^H)^+$ as the* transitive closure *of $\mathcal{R}_\circ^H$ by the following rules:*

$$\frac{t\ \mathcal{R}_\circ^H s}{t(\mathcal{R}_\circ^H)^+ s} \qquad \frac{t(\mathcal{R}_\circ^H)^+ s \qquad s(\mathcal{R}_\circ^H)^+ r}{t(\mathcal{R}_\circ^H)^+ r}$$

Proposition 4.2 *If $\mathcal{R}$ is reflexive and compatible then also $(\mathcal{R})^+$ is compatible*

Proof: The proof is given by induction on the derivation of $(\sim_\circ^H)^+$. □
So we know that $\sim_\circ$ is an equivalence relation, so is reflexive and then $\sim_\circ^H$ is compatible; but $\sim_\circ^H$ is also reflexive and then $(\sim_\circ^H)^+$ is compatible. Our goal is now to show that $(\sim_\circ^H)^+ \subseteq \sim_\circ$, but since we defined $\sim_\circ$ as the union of all bisimulations it is sufficient to prove that $(\sim_\circ^H)^+$ is a bisimulation to obtain that inclusion. The first step to reach this result is given by the following lemma.

Lemma 4.10 (Key Lemma) *Given two terms t, s, we have:*

- *If $\vdash t \sim_\circ^H s : \mathsf{aA} \to \mathsf{B}$, then for all $E \in \mathrm{T}^{\mathsf{B}}_{x:\mathsf{aA}}/_{\sim_\circ^H}$ equivalence class modulo $\sim_\circ^H$ it holds that $[\![t]\!](\lambda x.E) = [\![s]\!](\lambda x.E)$.*
- *If $\vdash t \sim_\circ^H s : \mathsf{Str}$, then for all $\underline{\mathrm{m}} \in \mathrm{V}^{\mathsf{Str}}$ we have $[\![t]\!](\underline{\mathrm{m}}) = [\![s]\!](\underline{\mathrm{m}})$.*

Proof: We work by induction on the derivation of $[\![t]\!]$.

- Suppose t is a value. If $t = \underline{\mathrm{m}}'$ by Howe's derivation rules we know that $\vdash \underline{\mathrm{m}}' \sim_\circ^H s$ iff:

$$\frac{\vdash \underline{\mathrm{m}}' \sim_\circ s}{\vdash \underline{\mathrm{m}}' \sim_\circ^H s}$$

that means $\underline{\mathrm{m}}' \sim_\circ s$ and then for all $\underline{\mathrm{m}} \in \mathrm{V}^{\mathsf{Str}}$,

$$[\![\underline{\mathrm{m}}']\!](\underline{\mathrm{m}}) = [\![s]\!](\underline{\mathrm{m}})$$

So the thesis. Similarly, if $t = \lambda x.t'$, by Howe's derivation rules we know:

$$\frac{t' \sim_\circ^H s' \quad \lambda x.s' \sim_\circ s}{\lambda x.t' \sim_\circ^H s}$$

So, given $E \in \mathrm{T}^{\mathsf{B}}_{x:\mathsf{aA}}/_{\sim_\circ^H}$ we have:

$$[\![\lambda x.t']\!](\lambda x.E) = \begin{cases} 1, & \text{if } t' \in E; \\ 0, & \text{otherwise.} \end{cases} = \begin{cases} 1, & \text{if } s' \in E; \\ 0, & \text{otherwise.} \end{cases} =$$
$$= [\![\lambda x.s']\!](\lambda x.E) = [\![s]\!](\lambda x.E)$$

- Suppose now $t = \mathsf{case}_{\mathsf{aA}\to\mathsf{B}}(t', t'_0, t'_1, t'_\epsilon)$, then: $\vdash \mathsf{case}_{\mathsf{aA}\to\mathsf{B}}(t', t'_0, t'_1, t'_\epsilon) \sim_\circ^H s : \mathsf{aA} \to \mathsf{B}$, which is derived from:

$$\frac{\begin{array}{ll} t' \sim_\circ^H s' & \\ t'_0 \sim_\circ^H s'_0 & t'_\epsilon \sim_\circ^H s'_\epsilon \\ t'_1 \sim_\circ^H s'_1 & \mathsf{case}_{\mathsf{aA}\to\mathsf{B}}(s', s'_0, s''_1, s_\epsilon) \sim s \end{array}}{\mathsf{case}_{\mathsf{aA}\to\mathsf{B}}(t', t'_0, t'_1, t'_\epsilon) \sim_\circ^H s}$$

Then for all $E \in \mathrm{T}^{\mathsf{B}}_{x:\mathsf{aA}}/_{\sim^H_\circ}$ we have by induction hypothesis:

$$\begin{aligned}
[\![t]\!](\lambda x.E) =& [\![\mathsf{case}_{\mathsf{aA}\to\mathsf{B}}(t', t'_0, t'_1, t'_\epsilon)]\!](\lambda x.E) = \\
=& [\![t']\!](\underline{\epsilon})[\![t'_\epsilon]\!](\lambda x.E) + \sum_{\underline{\mathrm{m}}} [\![t']\!](\underline{0\mathrm{m}})[\![t'_0]\!](E) + \sum_{\underline{\mathrm{m}}} [\![t']\!](\underline{1\mathrm{m}})[\![t'_1]\!](E) = \\
=& [\![s']\!](\underline{\epsilon})[\![s'_\epsilon]\!](\lambda x.E) + \sum_{\underline{\mathrm{m}}} [\![s']\!](\underline{0\mathrm{m}})[\![s'_0]\!](E) + \sum_{\underline{\mathrm{m}}} [\![s']\!](\underline{1\mathrm{m}})[\![s'_1]\!](E) = \\
=& [\![s]\!](\lambda x.E)
\end{aligned}$$

If $t = \mathsf{case}_{\mathsf{Str}}(t', t'_0, t'_1, t'_\epsilon) : \mathsf{Str}$ the proof is similar to the previous case.

- Suppose now $t = t_1 t_2 : \mathsf{aA} \to \mathsf{B}$ and so we have $\vdash t_1 t_2 \sim^H_\circ s : \mathsf{aA} \to \mathsf{B}$ that is derived from:

$$\frac{\begin{array}{c} t_1 \sim^H_\circ s_1 \\ t_2 \sim^H_\circ s_2 \quad s_1 s_2 \sim r \end{array}}{t_1 t_2 \sim^H_\circ r}$$

We have to face two different cases: $t_2 \in \mathrm{T}^{\mathsf{Str}}$ and $t_2 \in \mathrm{T}^{\mathsf{cC}\to\mathsf{D}}$. If $t_2 \in \mathrm{T}^{\mathsf{Str}}$ then for all $E \in \mathrm{T}^{\mathsf{B}}_{x:\mathsf{aA}}/_{\sim^H_\circ}$ we have:

$$\begin{aligned}
[\![t]\!](\lambda x.E) =& [\![t_1 t_2]\!](\lambda x.E) = \\
=& \sum_{\underline{\mathrm{m}}' \in \mathrm{V}^{\mathsf{Str}}} [\![t_2]\!](\underline{\mathrm{m}}') \left(\sum_{E_r \in \mathrm{V}^{\mathsf{aA}\to\mathsf{B}}_{y:\mathsf{aStr}}} \sum_{r \in E_r} [\![t_1]\!](\lambda y.r)[\![r\{\underline{\mathrm{m}}'/y\}]\!](\lambda x.E) \right) = \\
=& \sum_{\underline{\mathrm{m}}' \in \mathrm{V}^{\mathsf{Str}}} [\![t_2]\!](\underline{\mathrm{m}}') \left(\sum_{E_r \in \mathrm{V}^{\mathsf{aA}\to\mathsf{B}}_{y:\mathsf{aStr}}} [\![t_1]\!](\lambda y.E_r)[\![E_r\{\underline{\mathrm{m}}'/y\}]\!](\lambda x.E) \right) = \\
=& \sum_{\underline{\mathrm{m}}' \in \mathrm{V}^{\mathsf{Str}}} [\![s_2]\!](\underline{\mathrm{m}}') \left(\sum_{E_r \in \mathrm{V}^{\mathsf{aA}\to\mathsf{B}}_{y:\mathsf{aStr}}} [\![s_1]\!](\lambda y.E_r)[\![E_r\{\underline{\mathrm{m}}'/y\}]\!](\lambda x.E) \right) = \\
=& [\![s_1 s_2]\!](\lambda x.E) = [\![s]\!](\lambda x.E)
\end{aligned}$$

If $t_2 \in \mathrm{T}^{\mathsf{cC}\to\mathsf{D}}$ then we have:

$$\begin{aligned}
[\![t]\!](\lambda x.E) &= [\![t_1 t_2]\!](\lambda x.E) = \\
&= \sum_{E_v \in \mathrm{V}^{\mathsf{D}}_{z:\mathsf{cC}}} \sum_{v \in E_v} [\![t_2]\!](\lambda z.v) \left(\sum_{E_r \in \mathrm{V}^{\mathsf{aA}\to\mathsf{B}}_{y:\mathsf{cC}\to\mathsf{D}}} \sum_{r \in E_r} [\![t_1]\!](\lambda y.r)[\![r\{^{\lambda z.v}/_{y}\}]\!](\lambda x.E) \right) = \\
&= \sum_{E_v \in \mathrm{V}^{\mathsf{D}}_{z:\mathsf{cC}}} [\![t_2]\!](\lambda z.E_v) \left(\sum_{E_r \in \mathrm{V}^{\mathsf{aA}\to\mathsf{B}}_{y:\mathsf{cC}\to\mathsf{D}}} [\![t_1]\!](\lambda y.E_r)[\![E_r\{^{\lambda z.E_v}/_{y}\}]\!](\lambda x.E) \right) = \\
&= \sum_{E_v \in \mathrm{V}^{\mathsf{D}}_{z:\mathsf{cC}}} [\![s_2]\!](\lambda z.E_v) \left(\sum_{E_r \in \mathrm{V}^{\mathsf{aA}\to\mathsf{B}}_{y:\mathsf{cC}\to\mathsf{D}}} [\![s_1]\!](\lambda y.E_r)[\![E_r\{^{\lambda z.E_v}/_{y}\}]\!](\lambda x.E) \right) = \\
&= [\![s_1 s_2]\!](\lambda x.E) = [\![s]\!](\lambda x.E)
\end{aligned}$$

The case $t = t_1 t_2 : \mathsf{Str}$ is similar to the previous one.

- Finally, if $t = \mathsf{rec}_{\mathsf{aA}\to\mathsf{B}}(t', t'_0, t'_1, t'_\epsilon)$ then we have $\vdash \mathsf{rec}_{\mathsf{aA}\to\mathsf{B}}(t', t'_0, t'_1, t'_\epsilon) \sim^H_\circ s$ which is derived from:

$$\frac{\begin{array}{ccc} t' \sim^H_\circ s' & & \\ t'_0 \sim^H_\circ s_0 & & t'_\epsilon \sim^H_\circ s'_\epsilon \\ t'_1 \sim^H_\circ s_1 & \mathsf{rec}_{\mathsf{aA}\to\mathsf{B}}(s', s'_0, s'_1, s'_\epsilon) \sim s & \end{array}}{\mathsf{rec}_{\mathsf{aA}\to\mathsf{B}}(t', t'_0, t'_1, t'_\epsilon) \sim^H_\circ s}$$

then for all $E \in \mathrm{V}^{\mathsf{B}}_{x:\mathsf{a}}/_{\sim^H_\circ}$ we have:

$$\begin{aligned}
[\![t]\!](\lambda x.E) &= [\![\mathsf{rec}_{\mathsf{aA}\to\mathsf{B}}(t', t'_0, t'_1, t'_\epsilon)]\!](\lambda x.E) = \\
&= [\![t']\!](\underline{\epsilon})[\![t_\epsilon]\!](\lambda x.E) + \\
&\quad + \sum_{\underline{\mathrm{m}} \in \mathrm{V}^{\mathsf{Str}}} [\![t']\!](\underline{0\mathrm{m}})[\![(t'_0\underline{0\mathrm{m}})(\mathsf{rec}_{\mathsf{aA}\to\mathsf{B}}(\underline{\mathrm{m}}, t'_0, t'_1, t_\epsilon))]\!](\lambda x.E) + \\
&\quad + \sum_{\underline{\mathrm{m}} \in \mathrm{V}^{\mathsf{Str}}} [\![t']\!](\underline{1\mathrm{m}})[\![(t'_1\underline{1\mathrm{m}})(\mathsf{rec}_{\mathsf{aA}\to\mathsf{B}}(\underline{\mathrm{m}}, t'_1, t'_1, t_\epsilon))]\!](\lambda x.E) = \\
&= [\![s']\!](\underline{\epsilon})[\![s_\epsilon]\!](\lambda x.E) + \\
&\quad + \sum_{\underline{\mathrm{m}} \in \mathrm{V}^{\mathsf{Str}}} [\![s']\!](\underline{0\mathrm{m}})[\![(s'_0\underline{0\mathrm{m}})(\mathsf{rec}_{\mathsf{aA}\to\mathsf{B}}(\underline{\mathrm{m}}, s'_0, s'_1, s_\epsilon))]\!](\lambda x.E) + \\
&\quad + \sum_{\underline{\mathrm{m}} \in \mathrm{V}^{\mathsf{Str}}} [\![s']\!](\underline{1\mathrm{m}})[\![(s'_1\underline{1\mathrm{m}})(\mathsf{rec}_{\mathsf{aA}\to\mathsf{B}}(\underline{\mathrm{m}}, s'_1, s'_1, s_\epsilon))]\!](\lambda x.E) = \\
&= [\![s]\!](\lambda x.E)
\end{aligned}$$

The case $t = \mathsf{rec}_{\mathsf{Str}}(t', t'_0, t'_1, t'_\epsilon)$ is similar to the previous one.

So we have the thesis. □

Theorem 4.3 *$(\sim^H_\circ)^+$ is a bisimulation.*

Proof: We work by induction on the derivation of $(\sim^H_\circ)^+$, proving that $(\sim^H_\circ)^+$ satisfies the three points of the definition above. This, in particular, relies on the Key Lemma. □

Theorem 4.4 *Bisimilarity is a congruence.*

Proof: The proof comes easily from the fact that $(\sim^H_\circ)^+$ is a congruence. Indeed it is transitive and symmetric by definition and also compatible. By the definition of $(\sim^H_\circ)^+$ we have that $\sim_\circ \subseteq \sim^H_\circ \subseteq (\sim^H_\circ)^+$, but the theorem above tells us that $(\sim^H_\circ)^+$ is a bisimulation, and that it must be included in $\sim_\circ$, the symmetric and transitive

closure of all the bisimulations. So we have $\sim_\circ \subseteq (\sim_\circ^H)^+ \wedge (\sim_\circ^H)^+ \subseteq \sim_\circ$ which means that $\sim_\circ = (\sim_\circ^H)^+$, and we get the thesis, namely that $\sim_\circ$ is a congruence. □

As usual, being a congruence has soundness as an easy corollary:

Corollary 4.2 (Soundness) *Bisimilarity is included in context equivalence.*

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 [30]. Consider the following two terms:

$$t = \lambda x.\mathsf{if}\ \mathsf{rand}\ \mathsf{then}\ \mathtt{true}\ \mathsf{else}\ \mathtt{false}$$

$$s = \mathtt{if}\ \mathsf{rand}\ \mathtt{then}\ (\lambda x.\mathtt{true})\ \mathtt{else}\ (\lambda x.\mathtt{false})$$

where we have used some easy syntactic sugar. It is easy to show that t and s are trace equivalent, thus context equivalent. On the other hand, t and s cannot be bisimilar. 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.

Summing up, in this chapter we started to compare RSLR terms, discussing about equivalences; we showed how we can use traces to compare terms instead of arbitrary contexts and obtain the same results. Furthermore we propose a different approach based on coinduction, the applicative bisimulation, which is easier, due to the fact that eliminates the universal quantifications, but too strict; indeed we propose an example that reveals that this method is not complete. In the following chapter we will relax our constraints and observe a method to evaluate distances between terms, so that we are able to say how different two terms are.

Chapter 5

Metrics

In this section we move from equivalences to metrics. As we have seen we consider two programs t, s equivalent when for all linear contexts C we have that the probability that $C[t]$ outputs the empty string is *exactly* the same of $C[s]$; we also proved that this is equivalent to observe the probability to perform all traces.

Actually, we can easily notice that asking exactly for the same probability is a constraint which is too strong; in security and cryptography what we are requested to prove is that an adversary is not able to distinguish between two program, so this means that if the adversary is provided of a limited power of calculus it is not necessary that two programs behave exactly in the same way, what we need is that they are similar enough so that he can't distinguish between them.

For this reason, in this section we generalize the concept of equivalence by defining a notion of *distance* between programs so that we can evaluate how far two programs are, and this will be useful to declare how an adversary is able to separate them. The first step will be the definition of two distances, the first one based on contexts and the second of traces; then, in order to prove that the two distances are the same we will show that it is necessary to make a small change on the structure of traces, in particular we will show that it is necessary to modify the argument of the action $\mathsf{view}(\cdot)$.

5.1 Context and Trace Metrics

Definition 23 *The* context distance *is a function* $\delta^{\mathsf{C}}_{\mathsf{A}} : \mathrm{T}^{\mathsf{A}} \times \mathrm{T}^{\mathsf{A}} \to \mathbb{R}_{[0,1]}$ *defined for every type* A *such that, given two terms* $t, s \in \mathrm{T}^{\mathsf{A}}$ *we have:*

$$\delta^{\mathsf{C}}(t,s) = \sup_{\vdash C[\vdash \mathsf{A}]:\mathsf{Str}} \mid [\![C[t]]\!](\underline{\epsilon}) - [\![C[s]]\!](\underline{\epsilon}) \mid$$

For every type A the function $\delta^{\mathsf{C}}_{\mathsf{A}}$ is a *pseudometric* on the space of closed terms, indeed it is obvious that:

- $\delta^{\mathsf{C}}(t,s) = 0$ iff $t \simeq^{\mathsf{C}} s$,
  because for all C we have $[\![C[t]]\!](\underline{\epsilon}) = [\![C[s]]\!](\underline{\epsilon})$.
- $\delta^{\mathsf{C}}(t,s) = \delta^{\mathsf{C}}(s,t)$,
  by the property of absolute value.
- $\delta^{\mathsf{C}}(t,s) \leqslant \delta^{\mathsf{C}}(t,r) + \delta^{\mathsf{C}}(r,s)$,
  indeed we have:

$$\begin{aligned}
\delta^{\mathsf{C}}(t,s) &= \sup_{\vdash C[\vdash \mathsf{A}]:\mathsf{Str}} \mid [\![C[t]]\!](\underline{\epsilon}) - [\![C[s]]\!](\underline{\epsilon}) \mid = \\
&= \sup_{\vdash C[\vdash \mathsf{A}]:\mathsf{Str}} \mid [\![C[t]]\!](\underline{\epsilon}) - [\![C[r]]\!](\underline{\epsilon}) + [\![C[r]]\!](\underline{\epsilon}) - [\![C[s]]\!](\underline{\epsilon}) \mid \leqslant \\
&\leqslant \sup_{\vdash C[\vdash \mathsf{A}]:\mathsf{Str}} \mid [\![C[t]]\!](\underline{\epsilon}) - [\![C[r]]\!](\underline{\epsilon}) \mid + \sup_{\vdash C[\vdash \mathsf{A}]:\mathsf{Str}} \mid [\![C[r]]\!](\underline{\epsilon}) - [\![C[s]]\!](\underline{\epsilon}) \mid = \\
&= \delta^{\mathsf{C}}_{\mathsf{A}}(t,r) + \delta^{\mathsf{C}}_{\mathsf{A}}(r,s)
\end{aligned}$$

With δ^{C} we refer to the family $\{\delta^{\mathsf{C}}_{\mathsf{A}}\}_{\mathsf{A} \in \mathrm{A}}$.

Now by using a similar reasoning we can give the definition of trace distance.

Definition 24 *For every type* A *we define* trace distance *the function* δ^{T} *such that* $\delta^{\mathsf{T}}_{\mathsf{A}} : \mathrm{T}^{\mathsf{A}} \times \mathrm{T}^{\mathsf{A}} \to \mathbb{R}_{[0,1]}$ *and, given* $t, s \in \mathrm{T}^{\mathsf{A}}$*:*

$$\delta^{\mathsf{T}}_{\mathsf{A}}(t,s) = \sup_{\mathsf{T}:\mathsf{A}} \mid \Pr(t,\mathsf{T}) - \Pr(s,\mathsf{T}) \mid$$

It is easy to realize that δ^{T} is a pseudometric; we have:

- $\delta^{\mathsf{T}}_{\mathsf{A}}(t,s) = 0$ iff $t \simeq^{\mathsf{T}} s$,
  because for all T we have $\Pr(t,\mathsf{T}) = \Pr(s,\mathsf{T})$.

- $\delta^{\mathsf{T}}(t,s) = \delta^{\mathsf{T}}(s,t)$,
  by the property of the absolute value.

- $\delta^{\mathsf{T}}(t,s) \leqslant \delta^{\mathsf{T}}(t,r) + \delta^{\mathsf{T}}(r,s)$.
  Indeed we have:

$$\begin{aligned}
\delta^{\mathsf{T}}(t,s) =& \sup_{\mathsf{T}} \mid \Pr(t,\mathsf{T}) - \Pr(s,\mathsf{T}) \mid = \\
=& \sup_{\mathsf{T}} \mid \Pr(t,\mathsf{T}) - \Pr(r,\mathsf{T}) + \Pr(r,\mathsf{T}) - \Pr(s,\mathsf{T}) \mid \leqslant \\
\leqslant& \sup_{\mathsf{T}} \mid \Pr(t,\mathsf{T}) - \Pr(r,\mathsf{T}) \mid + \sup_{\mathsf{T}} \mid \Pr(r,\mathsf{T}) - \Pr(s,\mathsf{T}) \mid = \\
=& \delta^{\mathsf{T}}_{\mathsf{A}}(t,r) + \delta^{\mathsf{T}}_{\mathsf{A}}(r,s)
\end{aligned}$$

We refer to δ^{T} to the family $\{\delta^{\mathsf{T}}_{\mathsf{A}}\}_{\mathsf{A}\in\mathrm{A}}$ and now we show some properties of the trace distance δ^{T} applied on term distributions.

Lemma 5.1 *Given two term distributions $\mathcal{T},\mathcal{S}$ such that $\delta^{\mathsf{T}}(\mathcal{T},\mathcal{S}) = d$ then we have:*

1. *If $\mathcal{T} \Rrightarrow \mathcal{T}'$ then $\delta^{\mathsf{T}}(\mathcal{T}',\mathcal{S}) = d$.*

2. *If $\mathcal{T} \Rightarrow^{\mathsf{pass}(v)} \mathcal{T}', \mathcal{S} \Rightarrow^{\mathsf{pass}(v)} \mathcal{S}'$ the we have $\delta^{\mathsf{T}}(\mathcal{T}',\mathcal{S}') \leqslant d$*

Proof:

1. If $\mathcal{T} \Rrightarrow \mathcal{T}'$ we have by the small-step rules that $\mathcal{T} \Rightarrow^{\epsilon} \mathcal{T}'$ and then:

$$\Pr(\mathcal{T},\mathsf{T}) = \Pr(\mathcal{T},\epsilon\cdot\mathsf{T}) = \Pr(\mathcal{T}',\mathsf{T})$$

   so, if for all traces T we have that $\Pr(\mathcal{T},\mathsf{T}) = \Pr(\mathcal{T}',\mathsf{T})$ then we have the thesis, $\delta^{\mathsf{T}}(\mathcal{T},\mathcal{S}) = \delta^{\mathsf{T}}(\mathcal{T}',\mathcal{S})$.

2. It comes from the fact that the quantification is over a smaller set of traces, so the distance can't be greater.

□

5.2 Full Abstraction

How do we proceed if we want to prove that trace metric coincides with context metric? Could we proceed more or less like in the equivalences case? The answer is positive, but we need to be careful, let's observe the following example. Suppose $\mathcal{T}, \mathcal{S}$ be two string distributions such that:

$$\begin{aligned}\mathcal{T} =&\{(\underline{00})^{\frac{1}{4}-d}, (\underline{01})^{\frac{1}{4}-d}, (\underline{10})^{\frac{1}{4}+d}, (\underline{11})^{\frac{1}{4}+d}\}\\ \mathcal{S} =&\{(\underline{00})^{\frac{1}{4}}, (\underline{01})^{\frac{1}{4}}, (\underline{10})^{\frac{1}{4}}, (\underline{11})^{\frac{1}{4}}\}\end{aligned}$$

It is easy to evaluate the trace distance between $\mathcal{T}$ and $\mathcal{S}$ that is: $\delta^{\mathsf{T}}(\mathcal{T}, \mathcal{S}) = d$, but, what about the context distance δ^{C}?. Let's consider the context $C[\vdash \mathsf{Str}] : \mathsf{Str}$:

$$C := \mathsf{case}_{\mathsf{Str}}([\cdot], \underline{\epsilon}, \underline{0}, \underline{0})$$

We have:

$$\begin{aligned}[\![C[\mathcal{T}]]\!] =&\{(\underline{\epsilon})^{\frac{1}{2}-2\cdot d}, (\underline{0})^{\frac{1}{2}+2\cdot d}\}\\ [\![C[\mathcal{S}]]\!] =&\{(\underline{\epsilon})^{\frac{1}{2}}, (\underline{0})^{\frac{1}{2}}\}\end{aligned}$$

ando so: $\delta^{\mathsf{C}}(\mathcal{T}, \mathcal{S}) \geqslant 2 \cdot d > \delta^{\mathsf{T}}(\mathcal{T}, \mathcal{S})$.

The solution is a slight change on the definition of traces, in the action $\mathsf{view}()$ instead of observing a *single* trace, we need to observe a *finite set* of strings $\underline{\mathsf{M}}$. Furthermore we need to find something that plays the role of compatibility, since the latter is a property of *equivalences* and not of *metrics*; in metrics the corresponding of compatibility is *non-expansiveness*. A distance δ is said to be non-expansive iff for

every pair of terms $t, s : \mathsf{A}$ we have that $\delta(C[t], C[s]) \leqslant \delta(t, s)$ for every $C[\vdash \mathsf{A}] : \mathsf{B}$.

The way we prove δ^{T} is non-expansive is similar to the way we prove that $\simeq^{\mathsf{T}}$ is a congruence, that is by using pairs of the form $(C, \mathcal{T}), (C, \mathcal{S})$; what we need to change is the notion of *relatedness*, we need to adapt it to the case of distance.

Definition 25 *Given two pair distributions $\mathcal{P}, \mathcal{Q}$, we say that they are $d-$related, we write $\mathcal{P} \nabla_d \mathcal{Q}$, if there exist:*

- $\{C_i\}_{i \in I}$ *contexts,* $\{\mathcal{T}_i\}_{i \in I}, \{\mathcal{S}_i\}_{i \in I}$ *term distributions.*
- $\{t_j\}_{j \in J}$ *terms,* $\{\mathcal{T}_j\}_{j \in_J}, \{\mathcal{S}_j\}_{j \in J}$ *term distributions,*
- $\{r_i\}_{i \in I}, \{p_j\}_{j \in J}, \{q_j\}_{j \in J}$ *with*

$$\sum_{i \in I} r_i = 1 - R, \qquad \sum_{j \in J} p_j = \sum_{j \in J} q_j = R$$

such that:

$$\mathcal{P} = \{(C_i, \mathcal{T}_i)^{r_i}\}_{i \in I} \cup \{(t_j, \mathcal{T}_j)^{p_j}\}_{j \in J}$$

$$\mathcal{Q} = \{(C_i, \mathcal{S}_i)^{r_i}\}_{i \in I} \cup \{(t_j, \mathcal{S}_j)^{q_j}\}_{j \in J}$$

and:

$$\delta^{\mathsf{T}}(\mathcal{T}_i, \mathcal{S}_i) \leqslant d \qquad |\sum_{j \in \bar{J}} p_j - q_j| \leqslant R \cdot d$$

For all $i \in I, \bar{J} \subseteq J$.

So now, the goal is to prove that this relation is preserved by internal and external reductions.

Lemma 5.2 (Internal d-stability) *Given two pair distributions $\mathcal{P}, \mathcal{Q}$ with $\mathcal{P} \nabla_d \mathcal{Q}$ then if there exists $\mathcal{P}'$ such that $\mathcal{P} \Rrightarrow \mathcal{P}'$ then there exists $\mathcal{Q}'$ such that $\mathcal{Q} \Rightarrow^{\epsilon} \mathcal{Q}'$ or $\mathcal{Q} \Rrightarrow \mathcal{Q}'$ and $\mathcal{P}' \nabla_d \mathcal{Q}'$.*

Proof: The pair distribution $\mathcal{P}$ can reduce to $\mathcal{P}'$ in different ways: let's prove the statement for all the cases. The first two cases are simple, because they represent transitions that happens with probability 1. The other two cases are more complicated, because they involve probabilistic reductions and so we have to check that d-relatedness is preserved evaluating the differences between the variuos probabilities.

1. **Term distribution reduction**, $(C, \mathcal{T}) \to \{(C, \mathcal{T}')^1\}$.
   By definition we know that, given $(C, \mathcal{T})^r \in \mathcal{P}$, there exists $(C, \mathcal{S})^r \in \mathcal{Q}$ such that $\delta^{\mathsf{T}}(\mathcal{T}, \mathcal{S}) \leqslant d$. Then by the trace metric properties we know $\delta^{\mathsf{T}}(\mathcal{T}', \mathcal{S}) \leqslant d$, so we have:
   $$\mathcal{P} \Rrightarrow \mathcal{P}' = \mathcal{P} \backslash \{(C, \mathcal{T})^r\} \cup \{(C, \mathcal{T}')^r\}$$
   and obviously $\mathcal{P}' \nabla_d \mathcal{Q}$

2. **Mixed Reduction** $(C, \mathcal{T}) \to \{(C', \mathcal{T}')^1\}$
   This is the case in which the context passes a value to the hole, then we have a transition with probability 1.
   Suppose $\mathcal{P} \ni (C, \mathcal{T})^r \to \{(C', \mathcal{T}')^r\}$ then there exists $\mathcal{Q} \ni (C, \mathcal{S})^r \to \{(C', \mathcal{S}')^r\}$, because the contexts are the same by definition and $\delta^{\mathsf{T}}(\mathcal{T}', \mathcal{S}') \leqslant d$ by the trace distance properties.
   Then we have:
   $$\mathcal{P} \Rrightarrow \mathcal{P}' = \mathcal{P} \backslash \{(C, \mathcal{T})^r\} \cup \{(C', \mathcal{T}')^r\}$$
   $$\mathcal{Q} \Rrightarrow \mathcal{Q}' = \mathcal{Q} \backslash \{(C, \mathcal{S})^r\} \cup \{(C', \mathcal{S}')^r\}$$
   with $\mathcal{P}' \nabla_d \mathcal{Q}'$.

3. **Reduction without observation**.
   We have to consider two different cases: a context reduction and a term reduction.

(a) Context reduction $(C, \mathcal{T}) \to \{(C_k, \mathcal{T})^{r_k}\}_{k\in K}$.

By definition we know that there exists $(C, \mathcal{S})^r \in \mathcal{Q}$ that reduces the same way, so we have:

$$\mathcal{P} \Rrightarrow \mathcal{P}' = \mathcal{P}\backslash\{(C, \mathcal{T})^r\} \cup \{(C_k, \mathcal{T})^{r\cdot r_k}\}_{k\in K}$$

$$\mathcal{Q} \Rrightarrow \mathcal{Q}' = \mathcal{Q}\backslash\{(C, \mathcal{S})^r\} \cup \{(C_k, \mathcal{S})^{r\cdot r_k}\}_{k\in K}$$

and obviously $\mathcal{P}'\nabla_d\mathcal{Q}'$.

(b) Term reduction $(t, \mathcal{T}) \to \{(t_k, \mathcal{T}_k)^{r_k}\}$.

In this case we need to be careful, we have:

$$\mathcal{P} = \{(C_i, \mathcal{T}_i)^{r_i}\}_{i\in I} \cup \{(t_j, \mathcal{T}_j)^{p_j}\}_{j\in J}$$

$$\mathcal{Q} = \{(C_i, \mathcal{S}_i)^{r_i}\}_{i\in I} \cup \{(t_j, \mathcal{S}_j)^{q_j}\}_{j\in J}$$

With $\sum_{i\in I} r_i = 1 - R$.

Suppose that there exists $j' \in J$ such that $t_{j'} \to \{(t_k)^{r_k}\}_{k\in K}$ (it is the index of the term that reduces), then, if we set $J' = J\backslash\{j'\}$ we have:

$$\mathcal{P} \Rrightarrow \mathcal{P}' = \{(C_i, \mathcal{T}_i)^{r_i}\}_{i\in I} \cup \{(t_j, \mathcal{T}_j)^{p_j}\}_{j\in J'} \cup \{(t_k, \mathcal{T}_k)^{p_{j'}\cdot r_k}\}_{k\in K}$$

$$\mathcal{Q} \Rrightarrow \mathcal{Q}' = \{(C_i, \mathcal{S}_i)^{r_i}\}_{i\in I} \cup \{(t_j, \mathcal{S}_j)^{q_j}\}_{j\in J'} \cup \{(t_k, \mathcal{S}_k)^{q_{j'}\cdot r_k}\}_{k\in K}$$

Now, if we take $\bar{J} \subseteq J' \cup K$ we have:

$$\begin{aligned}
|\sum_{j\in\bar{J}} p_j - q_j| &= |\sum_{j\in\bar{J}\cap J'} p_j - q_j + \sum_{j\in\bar{J}\cap K} p_j - q_j| = \\
&= |\sum_{j\in\bar{J}\cap J'} p_j - q_j + \sum_{k\in K\cap\bar{J}} p_{j'}\cdot r_k - q_{j'}\cdot r_k| = \\
&= |\sum_{j\in\bar{J}\cap J'} p_j - q_j + (p_{j'} - q_{j'})\cdot \sum_{k\in K\cap\bar{J}} r_k| \leqslant \\
&= |\sum_{j\in\bar{J}\cap J'} p_j - q_j + (p_{j'} - q_{j'})| = |\sum_{j\in\bar{J}\cap J} p_j - q_j| \leqslant R\cdot d
\end{aligned}$$

4. **Observation Reduction** $(C, \mathcal{T}) \to \{(t_j, \mathcal{T})^{p_j}\}_{j \in J'}$, where $p_j = \mathcal{T}(\underline{\mathrm{m}}_j)$.
   Suppose there exists $i' \in I$ such that $\mathcal{T}_{i'} = \{\underline{\mathrm{m}}_j^{p_j}\}_{j \in J'}$.
   We can also suppose that $\mathcal{S}_{i'} = \{\underline{\mathrm{m}}_j^{q_j}\}_{j \in J'}$ (Otherwise we have $\mathcal{S}_{i'} \to^* \mathcal{S}'_{i'} = \{\underline{\mathrm{m}}_j^{q_j}\}_{j \in J'}$ with $\delta^{\mathsf{T}}(\mathcal{T}_{i'}, \mathcal{S}'_{i'}) \leqslant d$ by the trace distance properties).
   So, if we set $I' = I \backslash \{i'\}$, we have:

$$\mathcal{P} \Rrightarrow \mathcal{P}' = \{(C_i, \mathcal{T}_i)^{r_i}\}_{i \in I'} \cup \{(t_j, \mathcal{T}_j)^{p_j}\}_{j \in J} \cup \{(t_j, \mathcal{T}_{i'})^{p_j \cdot r_{i'}}\}_{j \in J'}$$

$$\mathcal{Q} \Rrightarrow \mathcal{Q}' = \{(C_i, \mathcal{S}_i)^{r_i}\}_{i \in I'} \cup \{(t_j, \mathcal{S}_j)^{q_j}\}_{j \in J} \cup \{(t_j, \mathcal{S}_{i'})^{q_j \cdot r_{i'}}\}_{j \in J'}$$

   We know $\sum_{i \in I'} r_i = 1 - (R + r_{i'})$, so, given $\bar{J} \subseteq J \cup J'$ we have:

$$\begin{aligned} |\sum_{j \in / \bar{J}} p_j - q_j| =& |\sum_{j \in \bar{J} \cap J} p_j - q_j + \sum_{j \in \bar{J} \cap J'} r_{i'} \cdot p_j - r_{i'} \cdot q_j| \leqslant \\ \leqslant& d \cdot R + |r_{i'} \sum_{j \in \bar{J} \cap J'} p_j - q_j| \leqslant d \cdot R + d \cdot r_{i'} = (R + r_{i'}) \cdot d \end{aligned}$$

   because if we set $\underline{\mathsf{M}} = \{\underline{\mathrm{m}}_j\}_{j \in \bar{J} \cap J'}$ we have

$$|\sum_{j \in \bar{J} \cap J'} p_j - q_j| = |\Pr(\mathcal{T}_{i'}, \mathsf{view}(\underline{\mathsf{M}})) - \Pr(\mathcal{S}_{i'}, \mathsf{view}(\underline{\mathsf{M}}))| \leqslant d$$

□

Lemma 5.3 (d-Relatedness, internally) *Given two pair distributions $\mathcal{P}, \mathcal{Q}$ with $\mathcal{P} \nabla_d \mathcal{Q}$ then there exist $\mathcal{P}', \mathcal{Q}'$ normal pair distributions with $\mathcal{P} \Rightarrow^{\epsilon} \mathcal{P}'$ and $\mathcal{Q} \Rightarrow^{\epsilon} \mathcal{Q}'$ such that $\mathcal{P}' \nabla_d \mathcal{Q}'$.*

Proof: The proof comes from the fact that, given $\mathcal{P}$ if it is not normal, there is $\mathcal{P}'$ normal such that $\mathcal{P} \Rrightarrow^* \mathcal{P}'$, and by the previous lemma we have $\mathcal{P}' \nabla_d \mathcal{Q}$. Then if $\mathcal{Q}$ isn't normal we can repeat the procedure and get $\mathcal{Q}'$ such that $\mathcal{Q} \Rrightarrow^* \mathcal{Q}'$ and $\mathcal{P}' \nabla_d \mathcal{Q}'$.
□

Lemma 5.4 (d-Relatedness, Externally) *Given two pair distributions $\mathcal{P} \nabla_d \mathcal{Q}$ then for every trace S:*

1. *If* S *is incomplete then there exist* $\mathcal{P}', \mathcal{Q}'$ *normal pair distributions such that* $\mathcal{P} \Rightarrow^{\mathsf{S}} \mathcal{Q}'$ *and* $\mathcal{Q} \Rightarrow^{\mathsf{S}} \mathcal{Q}'$ *with* $\mathcal{P}' \nabla_d \mathcal{Q}'$.

2. *If* S *is a complete trace then if* $\mathcal{P} \Rightarrow^{\mathsf{S}} p$ *and* $\mathcal{Q} \Rightarrow^{\mathsf{S}} q$ *we have* $\mid p - q \mid \leqslant d$.

Proof: We act by induction on the length of S first by proving the first case and then the second one.

1. If $\mathsf{S} = \epsilon$ then we get the thesis by the previous lemma.
   If $\mathsf{S} = \mathsf{S}' \cdot \mathsf{pass}(v)$ then by the small-step rules we have

$$\mathcal{P} \Rightarrow^{\mathsf{S}'} \mathcal{P}' = \{(C_i, \mathcal{T}_i)^{r_i}\}_{i \in I} \cup \{(\lambda x.t_j, \mathcal{P}_j)^{p_i}\}_{j \in J}$$

$$\mathcal{Q} \Rightarrow^{\mathsf{S}'} \mathcal{Q}' = \{(C_i, \mathcal{S}_i)^{r_i}\}_{i \in I} \cup \{(\lambda x.t_j, \mathcal{S}_j)^{q_j}\}_{j \in J}$$

   and by induction hypothesis we have $\mathcal{P}' \nabla_d \mathcal{Q}'$.
   By the small step rules we know that the action $\mathsf{pass}(v)$ is allowed only if $C = \lambda x.C'$ or $C = [\cdot]$ and $\mathcal{T} = \{(\lambda x.t_h)^{p_h}\}$.
   So if we set

$$I_1 = \{i \in I \mid C_i = \lambda x.C_i'\}, I_2 = \{i \in I \mid C_i = [\cdot]\}$$

   by the small-step rules we have:

$$\mathcal{P}' \Rightarrow^{\mathsf{pass}(v)} \mathcal{P}'' = \{(C_i'\{v/x\}, \mathcal{T}_i)^{r_i}\}_{i \in I_1} \cup \{([\cdot], \mathcal{T}_i')^{r_i}\}_{i \in I_2} \cup \{(t_j\{v/x\}, \mathcal{T}_j)^{p_j}\}_{j \in J}$$
$$\mathcal{Q}' \Rightarrow^{\mathsf{pass}(v)} \mathcal{Q}'' = \{(C_i'\{v/x\}, \mathcal{S}_i)^{r_i}\}_{i \in I_1} \cup \{([\cdot], \mathcal{S}_i')^{r_i}\}_{i \in I_2} \cup \{(t_j\{v/x\}, \mathcal{S}_j)^{q_j}\}_{j \in J}$$

   but by a previous lemma $\delta^{\mathsf{T}}(\mathcal{T}_i', \mathcal{S}_i') \leqslant d$ so we have evidently $\mathcal{P}'' \nabla_d \mathcal{Q}''$ and by applying the previous lemma we get the thesis.

2. If $\mathsf{S} = \mathsf{S}' \cdot \mathsf{view}(\underline{\mathsf{M}})$ is a complete trace then we have by induction hypothesis

$$\mathcal{P} \Rightarrow^{\mathsf{S}'} \mathcal{P}' = \{(\underline{\mathrm{m}}_j, \mathcal{T}_j)^{p_j}\}_{j \in J}, \qquad \mathcal{Q} \Rightarrow^{\mathsf{S}'} \mathcal{Q}' = \{(\underline{\mathrm{m}}_j, \mathcal{S}_j)^{q_j}\}_{j \in J}$$

with $\mathcal{P}'\nabla_d\mathcal{Q}'$ and $R = 1$ (Because there are no r_i).

By the small-step rules we know that

$$\mathcal{P}' \mapsto^{\mathsf{view}(\underline{\mathsf{M}})} p = \sum_{j;\underline{\mathrm{m}}_j \in \underline{\mathsf{M}}} p_j \qquad \mathcal{Q}' \mapsto^{\mathsf{view}(\underline{\mathsf{M}})} q = \sum_{j;\underline{\mathrm{m}}_j \in \underline{\mathsf{M}}} q_j$$

So if we set $\bar{J} = \{j \in J \mid \underline{\mathrm{m}}_j \in \underline{\mathsf{M}}\}$we have:

$$|p - q| = |\sum_{j \in \bar{J}} p_j - q_j| \leqslant 1 \cdot d$$

by definition of d-relatedness

□

Now we can prove the result of non-expansiveness.

Theorem 5.1 (Non-expansiveness) *Given two term distributions such that $\delta^{\mathsf{T}}(\mathcal{T}, \mathcal{S}) = d$ then for all contexts C we have that $\delta^{\mathsf{T}}(C[\mathcal{T}], C[\mathcal{S}]) \leqslant d$.*

Proof: In order to get the thesis we have to prove that for all traces S we have that

$$\mid \Pr(C[\mathcal{T}], \mathsf{S}) - \Pr(C[\mathcal{S}], \mathsf{S}) \mid \leqslant d$$

- If S doesn't end with the $\mathsf{view}(\cdot)$ action then we have: $\mid \Pr(C[\mathcal{T}], \mathsf{S}) - \Pr(C[\mathcal{S}], \mathsf{S}) \mid = \mid 1 - 1 \mid = 0 \leqslant d$
- Otherwise we have that $(C, \mathcal{T}) \mapsto^{\mathsf{S}} p_1 = \Pr(C[\mathcal{T}], \mathsf{S})$ and similarly $(C, \mathcal{S}) \mapsto^{\mathsf{S}} p_2 = \Pr(C[\mathcal{S}], \mathsf{S})$. But it is clear that $\{(C, \mathcal{T})^1\}\nabla_d\{(C, \mathcal{S})^1\}$, and so by lemma 5.4 we have: $\mid p_1 - p_2 \mid \leqslant d$ and then the thesis.

□

Once proved that the trace metric δ^{T} is non-expansive w.r.t. linear contexts we can prove that the context metric δ^{C} is less or equal than δ^{T}, that is:

Theorem 5.2 *For all t, s, we have: $\delta^{\mathsf{C}}(t, s) \leqslant \delta^{\mathsf{T}}(t, s)$.*

Proof: By the previous theorem we know that, if $\delta^{\mathsf{T}}(t,s) = d$ then for all context C, we have $\delta^{\mathsf{T}}(C[t], C[s]) \leqslant d$; so:

$$\delta^{\mathsf{C}}(t,s) = \sup_C \mid [\![C[t]]\!](\underline{\epsilon}) - [\![C[s]]\!](\underline{\epsilon}) \mid = \sup_C \mid \Pr(C[t], \mathsf{view}(\underline{\epsilon})) - \Pr(C[s], \mathsf{view}(\underline{\epsilon})) \mid \leqslant$$
$$\leqslant \sup_C \delta^{\mathsf{T}}(C[t], C[s]) \leqslant d$$

□

As a corollary of non-expansiveness, one gets that:

Theorem 5.3 (Full Abstraction) *For all t, s, $\delta^{\mathsf{T}}(t,s) = \delta^{\mathsf{C}}(t,s)$.*

Proof: $\delta^{\mathsf{T}}(t,s) \leqslant \delta^{\mathsf{C}}(t,s)$ because by the full abstraction lemma for all traces T there exists a context C_{T} such that $[\![C_{\mathsf{T}}[t]]\!](\underline{\epsilon}) = \Pr(t, \mathsf{T})$ and so the quantification over contexts catches the quantification over traces.
The other inclusion, $\delta^{\mathsf{C}}(t,s) \leqslant \delta^{\mathsf{T}}(t,s)$, is a consequence of non-expansiveness.
□

Summing up, in this section we propose two different notions of distance between terms, based on context and traces, in order to evaluate how different two RSLR terms are; we proved that these two definitions of distance actually coincide but we had been forced to slightly modify the definition of the action $\mathsf{view}()$ on traces, and we showed why by using a concrete example.

The results of this chapter are the basis of the following one, where we will propose a characterization of Computational Indistinguishability by a parametrization of context and trace distance.

Chapter 6

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. This is the core point of my thesis, because this characterization is the first step towards the simplification of cryptographic proofs, by the use of adversary in the form of traces, rather than the more complex use of a PPT algorithm $\mathcal{A}$.

First of all we will give the formal definition of computational indistinguishability then, after having observed the similarities with context distance we will give a parametric definition of context equivalence that, in the case of terms of type Str, coincides with CI. At this point we will offer a parametric definition of trace equivalence that will be proved to be the same of parametric context equivalence.

6.1 Parametric Context Equivalence

Definition 26 *Two distribution ensembles* $\{\mathcal{D}_n\}_{n\in\mathbb{N}}$ *and* $\{\mathcal{E}_n\}_{n\in\mathbb{N}}$ *(where both* $\mathcal{D}_n$ *and* $\mathcal{E}_n$ *are distributions on binary strings) are said to be* computationally indistinguishable *iff for every PPT algorithm* $\mathcal{A}$ *the following quantity is a negligible*[1]

[1] A negligible function is a function which tends to 0 faster than any inverse polynomial (see [22] for more details).

function of $n \in \mathbb{N}$*:*

$$\left| \Pr_{x \leftarrow \mathcal{D}_n}(\mathcal{A}(x, 1^n) = \epsilon) - \Pr_{x \leftarrow \mathcal{E}_n}(\mathcal{A}(x, 1^n) = \epsilon) \right|$$

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 ([22], Theorem 3.2.6, page 108). This is in contrast to the case of arbitrary ensembles [23].

The careful reader should have already spotted the similarity between CI and the notion of context distance as given in Chapter 5. 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 27 (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*, we write* $t \simeq^{\mathsf{C}}_{\mathbf{n}} s$ *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 6.1 *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.*

6.2 Parametric Trace Equivalence

How could traces capture the peculiar way parametric context equivalence treats the security parameter? First of all, observe that, in Definition 27, 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 $\mathtt{D}$.

So now, given a term $t : \mathsf{aStr}$ we define the probability of t to satisfy the action $\mathsf{view}(\mathtt{D})$ as $t \mapsto^{\mathsf{view}(\mathtt{D})} \sum_{\underline{\mathrm{m}}} [\![t]\!](\underline{\mathrm{m}}) \cdot [\![\mathtt{D}\underline{\mathrm{m}}]\!](\underline{\epsilon})$. Roughly speaking, the term t evaluates to a string distribution and the observation of this distribution is performed by the distiguisher; the probability that it outputs the empty string $\underline{\epsilon}$ is the probability that the term t satisfies the action $\mathsf{view}(\mathtt{D})$. Furthermore, a trace T is said *parametrically compatible* with a type $\mathsf{A} = \mathsf{aStr} \to \mathsf{B}$ if $\mathsf{T} = \mathsf{pass}(\underline{\mathrm{m}}) \cdot \mathsf{S}$ with $\mathsf{S} : \mathsf{B}$.

Definition 28 *Two terms $t, s : \mathsf{A}$ are* parametrically trace equivalent*, we write $t \simeq_{\mathbf{n}}^{\mathsf{T}} s$, iff for every trace T which is parametrically compatible with A, there is a negligible function $\varepsilon : \mathbb{N} \to \mathbb{R}_{[0,1]}$ such that:*

$$|\Pr(t, \mathsf{pass}(1^{\mathbf{n}}) \cdot \mathsf{T}) - \Pr(s, \mathsf{pass}(1^{\mathbf{n}}) \cdot \mathsf{T})| \leqslant \varepsilon(\mathbf{n})$$

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 5.3). Indeed:

Theorem 6.2 *Parametric trace equivalence and parametric context equivalence coincide.*

The first 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 Chapter 4.

The first change comes from the fact that the behavior of the terms we want to analyze depends on the security parameter, so basically we will not test a term distribution as in the equivalence and metric, but a we will work with a *family* of term distribution. This means that, if we want to compare two term $t, s : \mathsf{aStr} \to \mathsf{A}$ what we did before was to consider the term distributions $\mathcal{T} = \{(t)^1\}, \mathcal{S} = \{(s)^1\}$, what we do now is to consider two families that we call *parametric term distributions* $\bar{\mathcal{T}} = \{\mathcal{T}^{\mathbf{n}}\}_{\mathbf{n}\in\mathbb{N}}, \bar{\mathcal{S}} = \{\mathcal{S}^{\mathbf{n}}\}_{\mathbf{n}\in\mathbb{N}}$, where $\mathcal{T}^{\mathbf{n}} = \{(t1^{\mathbf{n}})^1\}, \mathcal{S}^{\mathbf{n}} = \{(s1^{\mathbf{n}})^1\}$ for all $\mathbf{n}$. Notice that $\bar{\mathcal{T}}, \bar{\mathcal{S}}$ are families of distributions of type A; so, given a trace T, compatible with A we have that a family of term distributions $\bar{\mathcal{T}} \mapsto^{\mathsf{T}} \xi$, where $\xi : \mathbb{N} \to \mathbb{R}_{[0,1]}$ and $\xi(\mathbf{n}) = \Pr(\mathcal{T}^{\mathbf{n}}, \mathsf{T})$.

At this point, the definition of parametric trace equivalence can be lifted to parametric term distributions: we say that two parametric term distributions $\bar{\mathcal{T}}, \bar{\mathcal{S}}$ are parametrically trace equivalent, if for all trace T such that $\bar{\mathcal{T}} \mapsto^{\mathsf{T}} \xi, \bar{\mathcal{S}} \mapsto^{\mathsf{T}} \mu$ then there exists a negligible function $\varepsilon : \mathbb{N} \to \mathbb{R}_{[0,1]}$ such that for all $\mathbf{n} \in \mathbb{N}$ we have

$$|\xi(\mathbf{n}) - \mu(\mathbf{n})| \leqslant \varepsilon(\mathbf{n})$$

We write $\bar{\mathcal{T}} \simeq^{\mathsf{T}}_{\mathbf{n}} \bar{\mathcal{S}}$.

Lemma 6.1 *If* $\bar{\mathcal{T}} \simeq^{\mathsf{T}}_{\mathbf{n}} \bar{\mathcal{S}}$ *then:*

1. *If* $\bar{\mathcal{T}} \to \bar{\mathcal{T}}'$ *then* $\bar{\mathcal{T}}' \simeq^{\mathsf{T}}_{\mathbf{n}} \bar{\mathcal{S}}$.

2. *If* $\bar{\mathcal{T}} \Rightarrow^{\mathsf{pass}(v)} \bar{\mathcal{T}}', \bar{\mathcal{S}} \Rightarrow^{\mathsf{pass}(v)} \bar{\mathcal{S}}'$ *then* $\bar{\mathcal{T}} \simeq^{\mathsf{T}}_{\mathbf{n}} \bar{\mathcal{S}}$.

6.3 Full Abstraction

So now, in order to obtain the full abstraction, our goal is to prove that parametric trace equivalence is a congruence w.r.t. linear contexts, that is to prove that, for all contexts $C[\mathsf{A}] : \mathsf{B}$, if $t, s : \mathsf{aStr} \to \mathsf{A}$; are parametrically trace equivalent, than also $\lambda y.C[ty] \simeq^{\mathsf{T}}_{\mathbf{n}} \lambda y.C[sy]$. As we disclosed before the way we prove it is similar to the previous cases: we work with pairs of the form $(C, \bar{\mathcal{T}})$ and we want to prove that if $\bar{\mathcal{T}} \simeq^{\mathsf{T}}_{\mathbf{n}} \bar{\mathcal{S}}$ then also $(C, \bar{\mathcal{T}}) \simeq^{\mathsf{T}}_{\mathbf{n}} (C, \bar{\mathcal{S}})$. Notice that a pair $(C, \bar{\mathcal{T}})$ is actually a *parametric* pair, indeed $(C, \bar{\mathcal{T}}) = \{(C, \mathcal{T}^{\mathbf{n}})\}_{\mathbf{n} \in \mathbb{N}}$. In order to prove the property of congruence we will work with *parametric pair distributions* as $\bar{\mathcal{P}} = \{(C_i, \bar{\mathcal{T}}_i)^{p_i}\}$; given a trace T we have that $\bar{\mathcal{P}} \mapsto^{\mathsf{T}} \xi$ if $\{(C_i, \bar{\mathcal{T}}_i^{\mathbf{n}})^{p_i}\} \mapsto^{\mathsf{T}} \xi(\mathbf{n})$ for all $\mathbf{n} \in \mathbb{N}$. The definition of parametric trace equivalence is naturally extended to parametric pair distributions.

The first step towards the congruence proof is to show that if C is an evaluation context then, given $\bar{\mathcal{T}} \simeq^{\mathsf{T}}_{\mathbf{n}} \bar{\mathcal{S}}$, such that $\bar{\mathcal{T}}, \bar{\mathcal{S}} : \mathsf{Str}$, then we have $(C, \bar{\mathcal{T}}) \simeq^{\mathsf{T}}_{\mathbf{n}} (C, \bar{\mathcal{S}})$; we prove it by using the following lemmas.

Lemma 6.2 *Given a parametric term distribution* $\bar{\mathcal{T}} : \mathsf{Str} \to \mathsf{A}$ *and a context* $C[\vdash \mathsf{A}] : \mathsf{B}$ *then for all traces* $\mathsf{T} = \mathsf{pass}(v_1) \cdot \mathsf{pass}(v_2) \cdots \mathsf{pass}(v_n) \cdot \mathsf{view}(\mathtt{D})$ *we have:*

$$(C, \bar{\mathcal{T}}) \mapsto^{\mathsf{T}} \xi \iff (Cv_1v_2 \cdots v_n, \bar{\mathcal{T}}) \mapsto^{\mathsf{view}(\mathtt{D})} \xi$$

Proof: We work by induction on the length of T. If $\mathsf{T} = \mathsf{view}(\mathtt{D})$ then it is obvious. Suppose now $\mathsf{T} = \mathsf{pass}(v) \cdot \mathsf{S} \cdot \mathsf{view}(\mathtt{D})$, then we have two different cases:

1. If $C = \lambda x.C'$ then we have $(C, \bar{\mathcal{T}}) \to^{\mathsf{pass}(v)} \{(C'\{v/x\}, \bar{\mathcal{T}})^1\}$.
   Similarly $(Cv\mathsf{S}, \bar{\mathcal{T}}) \to \{(C'\{v/x\}\mathsf{S}, \bar{\mathcal{T}})^1\}$.
   But now, by the induction hypothesis we know

   $$(C'\{v/x\}, \bar{\mathcal{T}}) \mapsto^{\mathsf{S} \cdot \mathsf{view}(\mathtt{D})} \xi \iff (C'\{v/x\}\mathsf{S}, \bar{\mathcal{T}}) \mapsto^{\mathsf{view}(\mathtt{D})} \xi$$

   and so the thesis.

2. Suppose now $C = [\cdot]$ then we have $([\cdot], \bar{\mathcal{T}}) \rightarrow^{\mathsf{pass}(v)} \{([\cdot], \bar{\mathcal{T}}')^1\}$.
   Similarly $([\cdot]v\mathsf{S}, \bar{\mathcal{T}}) \rightarrow \{([\cdot]\mathsf{S}, \bar{\mathcal{T}}')^1\}$.
   Then by the induction hypothesis we have:

$$([\cdot], \bar{\mathcal{T}}') \mapsto^{\mathsf{S}\cdot\mathsf{view}(\mathtt{D})} \xi \iff ([\cdot]\mathsf{S}, \bar{\mathcal{T}}') \mapsto^{\mathsf{view}(\mathtt{D})}$$

   and so the thesis.

□

Lemma 6.3 *Given two parametric term distributions $\bar{\mathcal{T}}, \bar{\mathcal{S}}$ of type string with $\bar{\mathcal{T}} \simeq^{\mathsf{T}}_{\mathbf{n}} \bar{\mathcal{S}}$ then for all evaluation contexts C such that $\vdash C[\vdash \mathsf{Str}] : \mathsf{B}$ we have $(C, \bar{\mathcal{T}}) \simeq^{\mathsf{T}}_{\mathbf{n}} (C, \bar{\mathcal{S}})$.*

Proof: Given a trace $\mathsf{T} = \mathsf{pass}(v_1) \cdot \mathsf{pass}(v_2) \cdots \mathsf{pass}(v_n) \cdot \mathsf{view}(\mathtt{D})$ if we consider the distinguisher $\mathtt{D}' = \lambda x.C[x]v_1v_2 \cdots v_n$ then we have:

$$(C, \bar{\mathcal{T}}) \mapsto^{\mathsf{T}} \xi \iff \bar{\mathcal{T}} \mapsto^{\mathsf{view}(\mathtt{D}')} \xi$$

$$(C, \bar{\mathcal{S}}) \mapsto^{\mathsf{T}} \mu \iff \bar{\mathcal{S}} \mapsto^{\mathsf{view}(\mathtt{D}')} \mu$$

□

We can extend this result to parametric pair distributions.

Proposition 1 *Given $\bar{\mathcal{P}} = \{(C_i, \bar{\mathcal{T}}_i)^{p_i}\}$, $\bar{\mathcal{Q}} = \{(C_i, \bar{\mathcal{S}}_i)^{p_i}\}$, two parametric pair distributions such that, for all i, C_i are evaluation contexts, $\bar{\mathcal{T}}_i, \bar{\mathcal{S}}_i : \mathsf{Str}$ and $\bar{\mathcal{T}}_i \simeq^{\mathsf{T}}_{\mathbf{n}} \bar{\mathcal{S}}_i$ then we have $\bar{\mathcal{P}} \simeq^{\mathsf{T}}_{\mathbf{n}} \bar{\mathcal{Q}}$.*

Proof: For all i we know that $(C_i, \bar{\mathcal{T}}_i) \simeq^{\mathsf{T}}_{\mathbf{n}} (C_i, \bar{\mathcal{S}}_i)$; it means that, given a trace T, if $(C_i, \bar{\mathcal{T}}_i) \mapsto^{\mathsf{T}} \xi_i, (C_i, \bar{\mathcal{S}}_i) \mapsto^{\mathsf{T}} \mu_i$ there exists a negligible function ε_i such that $|\xi_i(\mathbf{n}) - \mu_i(\mathbf{n})| \leqslant \varepsilon_i(\mathbf{n})$.
So we have by definition $\bar{\mathcal{P}} \mapsto^{\mathsf{T}} \xi = \sum_i p_i \cdot \xi_i$, $\bar{\mathcal{Q}} \mapsto^{\mathsf{T}} \mu = \sum_i p_i \cdot \mu_i =$ and then for all $\mathbf{n}$:

$$|\xi(\mathbf{n}) - \mu(\mathbf{n})| = |\sum_i p_i \cdot \xi_i(\mathbf{n}) - \sum_i p_i \cdot \mu_i(\mathbf{n})| = |\sum_i p_i \cdot (\xi_i(\mathbf{n}) - \mu_i(\mathbf{n}))| \leqslant$$
$$\leqslant \sum_i p_i \cdot |\xi_i(\mathbf{n}) - \mu_i(\mathbf{n})| \leqslant \sum_i p_i \cdot \varepsilon_i(\mathbf{n}) \leqslant \varepsilon(\mathbf{n})$$

for a certain negligible function ε. □

The last step is proving that the previous proposition is valid for all context C, so, before giving the proof we we give, as in the previous cases, a definition of relatedness between pair distributions.

Definition 29 *Given* $\bar{\mathcal{P}} = \{(C_i, \bar{\mathcal{T}}_i)^{p_i}\}_{i\in I}, \bar{\mathcal{Q}} = \{(C_i, \bar{\mathcal{S}}_i)^{q_i}\}_{i\in I}$*, we say that they are* parametrically related*, and we write* $\bar{\mathcal{P}} \nabla_{\mathbf{n}} \bar{\mathcal{Q}}$ *if they can be written as:*

$$\bar{\mathcal{P}} = \sum_{j\in J} r_j \cdot \bar{\mathcal{P}}_j + \{(C_k, \bar{\mathcal{T}}_k)^{r_k}\}_{k\in K}$$

$$\bar{\mathcal{Q}} = \sum_{j\in J} r_j \cdot \bar{\mathcal{Q}}_j + \{(C_k, \bar{\mathcal{S}}_k)^{r_k}\}_{k\in K}$$

Where for all j *we have* $\bar{\mathcal{P}}_j \simeq^{\mathsf{T}}_{\mathbf{n}} \bar{\mathcal{Q}}$ *and for all* k *we have* $\bar{\mathcal{T}}_k \simeq^{\mathsf{T}}_{\mathbf{n}} \bar{\mathcal{S}}_k$*.*

The idea behind this definition is that we say that two pair distributions $\bar{\mathcal{P}}, \bar{\mathcal{Q}}$ are parametrically related if we can split each one into two subdistributions, $\bar{\mathcal{P}} = \bar{\mathcal{P}}_1 + \bar{\mathcal{P}}_2,\ \bar{\mathcal{Q}} = \bar{\mathcal{Q}}_1 + \bar{\mathcal{Q}}_2$ such that $\bar{\mathcal{P}}_1$ and $\bar{\mathcal{Q}}_1$ are parametrically trace equivalent and $\bar{\mathcal{P}}_2, \bar{\mathcal{Q}}_2$ are made by couples where the contexts are the same with the same probability and the term distributions are parametrically trace related. Notice that, by this definition, if $(C, \mathcal{T}), (C, \mathcal{S})$ are two couples where C is an evaluation context with the hole of type Str and $\mathcal{T} \simeq^{\mathsf{T}}_{\mathbf{n}} \mathcal{S}$, then by lemma 6.2 $(C, \mathcal{T})$ and $(C, \mathcal{S})$ are parametrically trace equivalent and so included in $\bar{\mathcal{P}}_1$ and $\bar{\mathcal{Q}}_1$ respectively. The same happens if C is a term; indeed we have that for every trace T, $\Pr((t, \mathcal{T}), \mathsf{T}) = \Pr(t, \mathsf{T}) = \Pr((t, \mathcal{S}), \mathsf{T})$, so $(t, \mathcal{T}) \simeq^{\mathsf{T}}_{\mathbf{n}} (t, \mathcal{S})$ and then $(t, \mathcal{T})$ and $(t, \mathcal{S})$ are included in $\bar{\mathcal{P}}_1, \bar{\mathcal{Q}}_1$ respectively.

Our goal is now to prove that parametric relatedness is preserved by internal and external reduction.

Lemma 6.4 (Internal Parametric Stability) *Given* $\bar{\mathcal{P}}, \bar{\mathcal{Q}}$ *such that* $\bar{\mathcal{P}} \nabla_{\mathbf{n}} \bar{\mathcal{Q}}$ *then if* $\bar{\mathcal{P}} \Rrightarrow \bar{\mathcal{P}}'$ *then* $\bar{\mathcal{P}}' \nabla_{\mathbf{n}} \bar{\mathcal{Q}}$ *or there exists* $\bar{\mathcal{Q}}'$ *such that* $\bar{\mathcal{Q}} \Rrightarrow \bar{\mathcal{Q}}'$ *and* $\bar{\mathcal{P}}' \nabla_{\mathbf{n}} \bar{\mathcal{Q}}'$*.*

Proof: As in the previous case we study all the possible reduction of $\bar{\mathcal{P}}$.
If the reduction is made by one of the $\bar{\mathcal{P}}_h \Rrightarrow \bar{\mathcal{P}}'_h$ with $h \in J$ then we have:

$$\bar{\mathcal{P}} \Rrightarrow \bar{\mathcal{P}}' = \sum_{j \in J \setminus \{h\}} r_j \cdot \bar{\mathcal{P}}_j + r_h \cdot \bar{\mathcal{P}}'_h + \{(C_k, \bar{\mathcal{T}}_k)^{r_k}\}_{k \in K}$$

$$\bar{\mathcal{Q}} = \sum_{j \in J \setminus \{h\}} r_j \cdot \bar{\mathcal{Q}}_j + r_h \cdot \bar{\mathcal{Q}}_h + \{(C_k, \bar{\mathcal{S}}_k)^{r_k}\}_{k \in K}$$

but we know $\bar{\mathcal{P}}'_h \simeq^{\mathsf{T}}_{\mathbf{n}} \bar{\mathcal{Q}}_h$ and so $\bar{\mathcal{P}}' \nabla_{\mathbf{n}} \bar{\mathcal{Q}}$.

Suppose now that the reduction is made by a pair $(C_h, \bar{\mathcal{T}}_h)$ with $h \in K$. As observed before C_h is not an evaluation context of the form $\vdash C[\vdash \mathsf{Str}] : \mathsf{B}$, so we can exclude reductions with observation from the possible cases. Then we have these possibilities:

- **Term distribution reduction.**
  Suppose $\bar{\mathcal{T}}_h \to \bar{\mathcal{T}}'_h$. Then we have:

  $$\bar{\mathcal{P}} \Rrightarrow \bar{\mathcal{P}}' = \sum_{j \in J} r_j \cdot \bar{\mathcal{P}}_j + \{(C_k, \bar{\mathcal{T}}_k)^{r_k}\}_{k \in K \setminus \{h\}} + \{(C_h, \bar{\mathcal{T}}'_h)^{r_h}\}$$

  $$\bar{\mathcal{Q}} = \sum_{j \in J} r_j \cdot \bar{\mathcal{Q}}_j + \{(C_k, \bar{\mathcal{S}}_k)^{r_k}\}_{k \in K \setminus \{h\}} + \{(C_h, \bar{\mathcal{S}}_h)^{r_h}\}$$

  We know $\bar{\mathcal{T}}'_h \simeq^{\mathsf{T}}_{\mathbf{n}} \bar{\mathcal{S}}_h$ and so $\bar{\mathcal{P}}' \nabla_{\mathbf{n}} \bar{\mathcal{Q}}$.

- **Mixed Reduction.**
  In this case we have $(C_h, \bar{\mathcal{T}}_h) \to \{(C'_h, \bar{\mathcal{T}}'_h)^1\}$, so, by the fact that C_h is the same for $\bar{\mathcal{P}}$ and $\bar{\mathcal{Q}}$ we can say that $(C_h, \bar{\mathcal{S}}_h) \to \{(C'_h, \bar{\mathcal{S}}'_h)^1\}$ and $\bar{\mathcal{T}}'_h \simeq^{\mathsf{T}}_{\mathbf{n}} \bar{\mathcal{S}}'_h$.
  So we have:

  - If C'_h is a term or an evaluation context and $\bar{\mathcal{T}}'_h, \bar{\mathcal{S}}'_h : \mathsf{Str}$, if we set $J' = J \cup \{h\}, \bar{\mathcal{P}}_h = \{(C'_h, \bar{\mathcal{T}}'_h)^1\}, \bar{\mathcal{Q}}_h = \{(C'_h, \bar{\mathcal{S}}'_h)\}$ then by lemma 6.3, $\bar{\mathcal{P}}_h \simeq^{\mathsf{T}}_{\mathbf{n}} \bar{\mathcal{Q}}_h$, so:

    $$\bar{\mathcal{P}} \Rrightarrow \bar{\mathcal{P}}' = \sum_{j \in J'} r_j \cdot \bar{\mathcal{P}}_j + \{(C_k, \bar{\mathcal{T}}_k)^{r_k}\}_{k \in K \setminus \{h\}}$$

$$\bar{\mathcal{Q}} \Rrightarrow \bar{\mathcal{Q}}' = \sum_{j \in J'} r_j \cdot \bar{\mathcal{Q}}_j + \{(C_k, \bar{\mathcal{S}}_k)^{r_k}\}_{k \in K \setminus \{h\}}$$

– Otherwise:

$$\bar{\mathcal{P}} \Rrightarrow \bar{\mathcal{P}}' = \sum_{j \in J} r_j \cdot \bar{\mathcal{P}}_j + \{(C_k, \bar{\mathcal{T}}_k)^{r_k}\}_{k \in K \setminus \{h\}} + \{(C'_h, \bar{\mathcal{T}}'_h)^{r_h}\}$$

$$\bar{\mathcal{Q}} \Rrightarrow \bar{\mathcal{Q}}' = \sum_{j \in J} r_j \cdot \bar{\mathcal{Q}}_j + \{(C_k, \bar{\mathcal{S}}_k)^{r_k}\}_{k \in K \setminus \{h\}} + \{(C'_h, \bar{\mathcal{S}}'_h)^{r_h}\}$$

and so the thesis.

- **Reduction without observation.**
  Suppose now $(C_h, \bar{\mathcal{T}}_h) \to \{(C'_l, \bar{\mathcal{T}}_l)^{r'_l}\}_{l \in L}$; by definition we have $(C_h, \bar{\mathcal{S}}_h) \to \{(C'_l, \bar{\mathcal{S}}_l)^{r'_l}\}_{l \in L}$ where $\bar{\mathcal{T}}_l = \bar{\mathcal{T}}_h, \bar{\mathcal{S}}_l = \bar{\mathcal{S}}_h$ for all $l \in L$.
  So now:

  – If $\bar{\mathcal{T}}_h, \bar{\mathcal{S}}_h : \mathsf{Str}$, then we set:

$$\begin{aligned} L_1 &= \{l \in L \mid C'_l \text{ is an evaluation context or a term}\} \\ L_2 &= L \setminus L_1 \\ \bar{\mathcal{P}}_l &= \{(C'_l, \bar{\mathcal{T}}_l)^1\} \text{ with } l \in L_1 \\ \bar{\mathcal{Q}}_l &= \{(C'_l, \bar{\mathcal{S}}_l)^1\} \text{ with } l \in L_1 \end{aligned}$$

  and we have:

$$\bar{\mathcal{P}} \Rrightarrow \bar{\mathcal{P}}' = \sum_{j \in J} r_j \cdot \bar{\mathcal{P}}_j + \sum_{l \in L_1} r_h \cdot r'_l \cdot \bar{\mathcal{P}}_l + \{(C_k, \bar{\mathcal{T}}_k)\}_{k \in K} + \{(C'_l, \bar{\mathcal{T}}_l)^{r_h \cdot r'_l}\}_{l \in L_2}$$

$$\bar{\mathcal{Q}} \Rrightarrow \bar{\mathcal{Q}}' = \sum_{j \in J} r_j \cdot \bar{\mathcal{Q}}_j + \sum_{l \in L_1} r_h \cdot r'_l \cdot \bar{\mathcal{Q}}_l + \{(C_k, \bar{\mathcal{S}}_k)\}_{k \in K} + \{(C'_l, \bar{\mathcal{S}}_l)^{r_h \cdot r'_l}\}_{l \in L_2}$$

  – Otherwise we make a similar procedure but we set $L_1 = \{l \in L \mid C'_l \text{ is a term}\}$.

  so by the fact that for all $l \in L_1, \bar{\mathcal{P}}_l \simeq^{\mathsf{T}}_{\mathbf{n}} \bar{\mathcal{Q}}_l$ and for all $l \in L_2, \bar{\mathcal{T}}_l \simeq^{\mathsf{T}}_{\mathbf{n}} \bar{\mathcal{S}}_l$, we have that $\bar{\mathcal{P}}' \nabla_{\mathbf{n}} \bar{\mathcal{Q}}'$.

□

Lemma 6.5 (Parametric relatedness, Internally) *Given $\bar{\mathcal{P}}, \bar{\mathcal{Q}}$ such that $\bar{\mathcal{P}} \nabla_{\mathbf{n}} \bar{\mathcal{Q}}$ then there exists $\bar{\mathcal{P}}', \bar{\mathcal{Q}}'$ normal, such that if $\bar{\mathcal{P}} \Rightarrow^{\epsilon} \bar{\mathcal{P}}'$ and $\bar{\mathcal{Q}} \Rightarrow^{\epsilon} \bar{\mathcal{Q}}'$ then $\bar{\mathcal{P}}' \nabla_{\mathbf{n}} \bar{\mathcal{Q}}'$.*

Lemma 6.6 (Parametric Relatedness, Externally) *Given $\bar{\mathcal{P}}, \bar{\mathcal{Q}} : \mathsf{A}$ with $\bar{\mathcal{P}} \nabla_{\mathbf{n}} \bar{\mathcal{Q}}$, then for all incomplete traces T, there exist $\bar{\mathcal{P}}', \bar{\mathcal{Q}}'$ normal, such that $\bar{\mathcal{P}} \Rightarrow^{\mathsf{T}} \bar{\mathcal{P}}', \bar{\mathcal{Q}} \Rightarrow^{\mathsf{T}} \bar{\mathcal{Q}}'$ and $\bar{\mathcal{P}}' \nabla_{\mathbf{n}} \bar{\mathcal{Q}}'$.*

Proof: We work by induction on the length of the trace.
If $\mathsf{T} = \epsilon$ then by lemma 6.5 we get the thesis.
If $\mathsf{T} = \mathsf{S} \cdot \mathsf{pass}(v)$ the we have:

$$\bar{\mathcal{P}} \Rightarrow^{\mathsf{S}} \sum_{j \in J} r_j \cdot \bar{\mathcal{P}}_j + \{(C_k, \bar{\mathcal{T}}_k)^{r_k}\}_{k \in K}$$

$$\bar{\mathcal{Q}} \Rightarrow^{\mathsf{S}} \sum_{j \in J} r_j \cdot \bar{\mathcal{Q}}_j + \{(C_k, \bar{\mathcal{S}}_k)^{r_k}\}_{k \in K}$$

with $\bar{\mathcal{P}}_j \simeq^{\mathsf{T}}_{\mathbf{n}} \bar{\mathcal{Q}}_j$ for all $j \in J$ and $\bar{\mathcal{T}}_k \simeq^{\mathsf{T}}_{\mathbf{n}} \bar{\mathcal{S}}_k$ for all $k \in K$.
Then we know $\bar{\mathcal{P}}_j \rightarrow^{\mathsf{pass}(v)} \bar{\mathcal{P}}'_j, \bar{\mathcal{Q}}_j \rightarrow^{\mathsf{pass}(v)} \bar{\mathcal{Q}}'_j$ but $\bar{\mathcal{P}}'_j \simeq^{\mathsf{T}} \bar{\mathcal{Q}}'_j$ by the properties of $\simeq^{\mathsf{T}}_{\mathbf{n}}$.
Similarly $(C_k, \bar{\mathcal{T}}_k) \rightarrow^{\mathsf{pass}(v)} \{(C'_k, \bar{\mathcal{T}}'_k)^1\}, (C_k, \bar{\mathcal{S}}_k) \rightarrow^{\mathsf{pass}(v)} \{(C'_k, \bar{\mathcal{S}}'_k)^1\}$ with $\bar{\mathcal{T}}'_k \simeq^{\mathsf{T}}_{\mathbf{n}} \bar{\mathcal{S}}'_k$ for all $k \in K$.
So we have:

$$\bar{\mathcal{P}} \Rightarrow^{\mathsf{S} \cdot \mathsf{pass}(v)} \bar{\mathcal{P}}' = \sum_{j \in J} r_j \cdot \bar{\mathcal{P}}'_j + \{(C'_k, \bar{\mathcal{T}}'_k)^{r_k}\}_{k \in K}$$

$$\bar{\mathcal{Q}} \Rightarrow^{\mathsf{S} \cdot \mathsf{pass}(v)} \bar{\mathcal{Q}}' = \sum_{j \in J} r_j \cdot \bar{\mathcal{Q}}'_j + \{(C'_k, \bar{\mathcal{S}}'_k)^{r_k}\}_{k \in K}$$

But $\bar{\mathcal{P}}' \nabla_{\mathbf{n}} \bar{\mathcal{Q}}'$ so by applying lemma 6.5 we get the thesis. □
Now we are able to prove this result.

Theorem 6.3 *Given $\bar{\mathcal{P}}, \bar{\mathcal{Q}}$ such that $\bar{\mathcal{P}} \nabla_{\mathbf{n}} \bar{\mathcal{Q}}$, then $\bar{\mathcal{P}} \simeq^{\mathsf{T}}_{\mathbf{n}} \bar{\mathcal{Q}}$.*

Proof: We want to show that for every trace $\mathsf{T} = \mathsf{S} \cdot \mathsf{view}(\mathtt{D})$ we have $\Pr(\bar{\mathcal{P}}, \mathsf{T}) = \Pr(\bar{\mathcal{Q}}, \mathsf{T})$.
By lemma 6.6 we know that there exist $\bar{\mathcal{P}}', \bar{\mathcal{Q}}'$ normal, such that $\bar{\mathcal{P}} \Rightarrow^{\mathsf{S}} \bar{\mathcal{Q}}'$, $\bar{\mathcal{Q}} \Rightarrow^{\mathsf{S}} \bar{\mathcal{Q}}'$ and $\bar{\mathcal{P}}' \triangledown_{\mathbf{n}} \bar{\mathcal{Q}}'$.
So we have $\bar{\mathcal{P}}' = \{(C_i, \bar{\mathcal{T}}_i)^{p_i}\}$, $\bar{\mathcal{Q}}' = \{(C_i, \bar{\mathcal{S}}_i)^{q_i}\}$, but both distributions are normal and of type Str, that means $C_i = [\vdash \mathsf{Str}]$ and $\bar{\mathcal{T}}_i, \bar{\mathcal{S}}_i$ normal or $C_i = \underline{\mathrm{m}_i}$.
If $\bar{\mathcal{P}}' \triangledown_{\mathbf{n}} \bar{\mathcal{Q}}'$ then we have $\bar{\mathcal{P}}' = \bar{\mathcal{P}}'_1 + \bar{\mathcal{P}}'_2$ and $\bar{\mathcal{Q}}' = \bar{\mathcal{Q}}'_1 + \bar{\mathcal{Q}}'_2$, but all the contexts are strings or evaluation contexts with the hole of type Str, so we can say that $\bar{\mathcal{P}}'_2 = \bar{\mathcal{Q}}'_2 = \varnothing$, that means $\bar{\mathcal{P}}' = \bar{\mathcal{P}}'_1 \simeq^{\mathsf{T}}_{\mathbf{n}} \bar{\mathcal{Q}}'_1 = \bar{\mathcal{Q}}'$. □

Corollary 6.1 *Parametrically trace equivalence is a congruence w.r.t. linear context.*

Proof: Given $t, s : \mathsf{aStr} \to \mathsf{A}$, such that $t \simeq^{\mathsf{T}}_{\mathbf{n}} s$, we want to show that for all context $C[\vdash \mathsf{A}] : \mathsf{B}$ then $\lambda y.C[ty] \simeq^{\mathsf{T}}_{\mathbf{n}} \lambda y.C[sy]$.
We set $\bar{\mathcal{T}} = \{\mathcal{T}^{\mathbf{n}}\}_{\mathbf{n}\in\mathbb{N}}, \bar{\mathcal{S}} = \{\mathcal{S}^{\mathbf{n}}\}_{\mathbf{n}\in\mathbb{N}}$, with $\mathcal{T}^{\mathbf{n}} = \{(t1^{\mathbf{n}})^1\}, \mathcal{S}^{\mathbf{n}} = \{(s1^{\mathbf{n}})^1\}$, and $\bar{\mathcal{P}} = \{(C, \bar{\mathcal{T}})^1\}, \bar{\mathcal{Q}} = \{(C, \bar{\mathcal{S}})^1\}$, with $\mathcal{P}^{\mathbf{n}} = \{(C, \mathcal{T}^{\mathbf{n}})^1\}, \mathcal{Q}^{\mathbf{n}} = \{(C, \mathcal{S}^{\mathbf{n}})^1\}$.
It is obvious $\bar{\mathcal{P}} \triangledown_{\mathbf{n}} \bar{\mathcal{Q}}$, so for every trace T such that $\bar{\mathcal{P}} \mapsto^{\mathsf{T}} \xi, \bar{\mathcal{Q}} \mapsto^{\mathsf{T}} \mu$ there exists a negligible function ε such that for all $\mathbf{n} \in \mathbb{N}$ we have $|\xi(\mathbf{n}) - \mu(\mathbf{n})| \leqslant \varepsilon(\mathbf{n})$.
So we get:

$$
\begin{aligned}
&|\Pr(\lambda y.C[ty], \mathsf{pass}(1^{\mathbf{n}}) \cdot \mathsf{T}) - \Pr(\lambda y.C[sy], \mathsf{pass}(1^{\mathbf{n}}) \cdot \mathsf{T})| = \\
&= |\Pr(C[t1^{\mathbf{n}}], \mathsf{T}) - \Pr(C[s1^{\mathbf{n}}], \mathsf{T})| = |\xi(\mathbf{n}) - \mu(\mathbf{n})| \leqslant \\
&\leqslant \varepsilon(\mathbf{n})
\end{aligned}
$$

and so the thesis. □

Chapter 7

Applications

In this chapter we will give some applications of our method to real cryptographic situations. As we talked in the introduction, in order to analyze a cryptographic primitive or a protocol, we need to use a standard approach based on four points: we need to define the security property we want to prove, we need to define a realistic model of a potential adversary, then we need to present a formalization of the cryptographic primitive or protocol under study and finally we need to prove that the security property reduces to a particular assumption.

7.1 A Simple Encryption Scheme

The first case we are going to analyze is a cryptographic primitive for the encryption of a message; first of all we need to define the security property we want to prove, that is *secrecy*. We formalize this property by saying that the encryptions of two different messages are indistinguishable.

This is a common assumption in cryptography and it is called *IND-CPA* (Indistinguishability against Chosen-Plaintext Attack), indeed if an adversary is not able to distinguish between the encryption of two different messages, that are chosen by himself, it means that he can't take any information of the plaintext from the

cyphertext and so secrecy is certain. The second step is the definition of a realistic model of the adversary, so we consider all the probabilistic polynomial time algorithm $\mathcal{A}$, that are encoded in our system by linear context C.

Now, we introduce the cryptographic system under study. Consider, as an example, the two terms

$$t = \lambda n.(\lambda k.\lambda x.\lambda y.\mathsf{ENC}(x\ n)\ k)(\mathsf{GEN}\ n) \quad s = \lambda n.(\lambda k.\lambda x.\lambda y.\mathsf{ENC}(y\ n)\ k)(\mathsf{GEN}\ n)$$

where ENC is meant to be an encryption function, GEN is a function generating a random key and x, y are two deterministic function that return a string whose length depends on the security parameter. We can prove that the two terms are parametrically context equivalent if ENC is the cryptoscheme induced by a pseudorandom generator. We propose:

$$\mathsf{ENC} = \lambda\underline{\mathrm{m}}.\lambda k.\underline{\mathrm{m}} \oplus (G\ k) : \mathsf{aStr} \to \mathsf{bStr} \to \mathsf{Str}$$

where $G : \mathsf{Str} \to \mathsf{Str}$ is a pseudorandom generator.

Now, in order to prove our security property, that is that t and s are indistinguishable, we need to reduce to a particular assumption. We will prove that the indistinguishability between t, s can be reduced to the indistinguishability between the pseudorandom generator G and a real random generator R. Going into detail we consider:

$$\mathsf{PRG} := \lambda n.(\lambda k.G\ k)(\mathsf{GEN}\ n) \qquad \mathsf{RND} := \lambda n.(\lambda k.R\ k)(\mathsf{GEN}\ n)$$

where PRG is a pseudorandom generator, RND a random generator.

We will show that if we are able to distinguish between t and s then we are able to distinguish between PRG and RND; but, by the fact that a pseudorandom generator and a random generator are indistinguishable by definition, we get a contradiction and so it can't be possible to distinguish between t and s.

Suppose t, s aren't indistinguishable; then there exists a trace $\mathsf{T} = \mathsf{pass(f)} \cdot \mathsf{pass(g)} \cdot \mathsf{view(D)}$, with $\mathsf{f}, \mathsf{g} : \mathsf{aStr} \to \mathsf{Str}$, such that:

$$|\Pr(t, \mathsf{pass}(1^{\mathbf{n}}) \cdot \mathsf{T}) - \Pr(s, \mathsf{pass}(1^{\mathbf{n}}) \cdot \mathsf{T})| > \varepsilon(\mathbf{n})$$

for all ε negligible function.
If $\mathsf{GEN}\ 1^{\mathbf{n}} \to \{(\mathsf{k})^{p_{\mathsf{k}}}\}_{\mathsf{k} \in \mathsf{S}(G\ 1^{\mathbf{n}})}$ we have:

$$t \Rightarrow^{\mathsf{pass}(1^{\mathbf{n}}) \cdot \mathsf{pass(f)} \cdot \mathsf{pass(g)}} \{(\mathsf{f}1^{\mathbf{n}} \oplus (G\ \mathsf{k})^{p_{\mathsf{k}}}\}_{\mathsf{k} \in \mathsf{S}(G\ 1^{\mathbf{n}})}$$

$$s \Rightarrow^{\mathsf{pass}(1^{\mathbf{n}}) \cdot \mathsf{pass(f)} \cdot \mathsf{pass(g)}} \{(\mathsf{g}1^{\mathbf{n}} \oplus (G\ \mathsf{k}))^{p_{\mathsf{k}}}\}_{\mathsf{k} \in \mathsf{S}(G\ 1^{\mathbf{n}})}$$

and then we have:

$$\begin{aligned}
\varepsilon(\mathbf{n}) \leqslant & |\ \Pr(t, \mathsf{pass}(1^{\mathbf{n}}) \cdot \mathsf{T}) - \Pr(s, \mathsf{pass}(1^{\mathbf{n}}) \cdot \mathsf{T})| = \\
= & |\sum_{\mathsf{k} \in \mathsf{S}(\mathsf{GEN}\ 1^{\mathbf{n}})} p_{\mathsf{k}} \cdot [\![\mathtt{D}\ (\mathsf{f}1^{\mathbf{n}} \oplus (G\ \mathsf{k}))]\!](\underline{\epsilon}) - \sum_{\mathsf{k} \in \mathsf{S}(\mathsf{GEN}\ 1^{\mathbf{n}})} p_{\mathsf{k}} \cdot [\![\mathtt{D}\ (\mathsf{g}1^{\mathbf{n}} \oplus (G\ \mathsf{k}))]\!](\underline{\epsilon})| = \\
= & |\xi(\mathbf{n}) - \mu(\mathbf{n})|
\end{aligned}$$

for all $\varepsilon : \mathbb{N} \to \mathbb{R}$ negligible functions.
Given a random generator R, we can say by the properties of one-time-pad that

$$[\![\mathtt{D}(\mathsf{f}1^{\mathbf{n}} \oplus (R\ \mathsf{k})]\!](\underline{\epsilon}) = [\![\mathtt{D}(\mathsf{g}1^{\mathbf{n}} \oplus (R\ \mathsf{k}))]\!](\underline{\epsilon}) = \nu(\mathbf{n})$$

So we have that for all $\mathbf{n}$:

$$\varepsilon(\mathbf{n}) < |\xi(\mathbf{n}) - \mu(\mathbf{n})| = |\xi(\mathbf{n}) - \nu(\mathbf{n}) + \nu(\mathbf{n}) - \mu(\mathbf{n})| \leqslant |\xi(\mathbf{n}) - \nu(\mathbf{n})| + |\nu(\mathbf{n}) - \mu(\mathbf{n})|$$

This means that at least one between $|\xi(\mathbf{n}) - \nu(\mathbf{n})|$ and $|\nu(\mathbf{n}) - \mu(\mathbf{n})|$ is more than negligible. Now, we call $\mathsf{GEN}^{-1} : \mathsf{aStr} \to \mathsf{Str}$ the inverse function of GEN, that given a string k returns the security parameter string $1^{\mathbf{n}}$; this way we can build

$$\mathtt{D}_{\mathsf{f}} = \lambda z.\mathtt{D}\ (\ (\mathsf{f}(\mathsf{GEN}^{-1} z)) \oplus z\ ) \qquad \mathtt{D}_{\mathsf{g}} = \lambda z.\mathtt{D}\ (\ (\mathsf{g}(\mathsf{GEN}^{-1} z)) \oplus z\ )$$

At this point, we consider:

$$\begin{aligned}
\Pr(\mathsf{PRG}, \mathsf{pass}(1^{\mathbf{n}}) \cdot \mathsf{view}(\mathtt{D}_{\mathsf{f}})) &= \sum_{\mathsf{k} \in \mathsf{S}(\mathsf{GEN}\ 1^{\mathbf{n}})} p_{\mathsf{k}} \cdot [\![\mathtt{D}_{\mathsf{f}}(G\ \mathsf{k})]\!](\underline{\epsilon}) = \\
&= \sum_{\mathsf{k} \in \mathsf{S}(\mathsf{GEN}\ 1^{\mathbf{n}})} p_{\mathsf{k}} \cdot [\![\mathtt{D}(\mathsf{f}1^{\mathbf{n}} \oplus (G\ \mathsf{k})]\!](\underline{\epsilon}) = \\
&= \xi(\mathbf{n}) \\
\Pr(\mathsf{RND}, \mathsf{pass}(1^{\mathbf{n}}) \cdot \mathsf{view}(\mathtt{D}_{\mathsf{f}})) &= \sum_{\mathsf{k} \in \mathsf{S}(\mathsf{GEN}\ 1^{\mathbf{n}})} p_{\mathsf{k}} \cdot [\![\mathtt{D}_{\mathsf{f}}(R\ \mathsf{k})]\!](\underline{\epsilon}) = \\
&= \sum_{\mathsf{k} \in \mathsf{S}(\mathsf{GEN}\ 1^{\mathbf{n}})} p_{\mathsf{k}} \cdot [\![\mathtt{D}(\mathsf{f}1^{\mathbf{n}} \oplus (R\ \mathsf{k})]\!](\underline{\epsilon}) = \\
&= \nu(\mathbf{n})
\end{aligned}$$

and we have:

$$\mid \Pr(\mathsf{PRG}, \mathsf{pass}(1^{\mathbf{n}}) \cdot \mathsf{view}(\mathtt{D}_{\mathsf{f}})) - \Pr(\mathsf{RND}, \mathsf{pass}(1^{\mathbf{n}}) \cdot \mathsf{view}(\mathtt{D}_{\mathsf{f}})) \mid = \mid \xi(\mathbf{n}) - \nu(\mathbf{n}) \mid$$

Funrthermore:

$$\begin{aligned}
\Pr(\mathsf{PRG}, \mathsf{pass}(1^{\mathbf{n}}) \cdot \mathsf{view}(\mathtt{D}_{\mathsf{g}})) &= \sum_{\mathsf{k} \in \mathsf{S}(\mathsf{GEN}\ 1^{\mathbf{n}})} p_{\mathsf{k}} \cdot [\![\mathtt{D}_{\mathsf{g}}(G\ \mathsf{k})]\!](\underline{\epsilon}) = \\
&= \sum_{\mathsf{k} \in \mathsf{S}(\mathsf{GEN}\ 1^{\mathbf{n}})} p_{\mathsf{k}} \cdot [\![\mathtt{D}(\mathsf{g}1^{\mathbf{n}} \oplus (G\ \mathsf{k})]\!](\underline{\epsilon}) = \\
&= \mu(\mathbf{n}) \\
\Pr(\mathsf{RND}, \mathsf{pass}(1^{\mathbf{n}}) \cdot \mathsf{view}(\mathtt{D}_{\mathsf{g}})) &= \sum_{\mathsf{k} \in \mathsf{S}(\mathsf{GEN}\ 1^{\mathbf{n}})} p_{\mathsf{k}} \cdot [\![\mathtt{D}_{\mathsf{g}}(R\ \mathsf{k})]\!](\underline{\epsilon}) = \\
&= \sum_{\mathsf{k} \in \mathsf{S}(\mathsf{GEN}\ 1^{\mathbf{n}})} p_{\mathsf{k}} \cdot [\![\mathtt{D}(\mathsf{g}1^{\mathbf{n}} \oplus (R\ \mathsf{k})]\!](\underline{\epsilon}) = \\
&= \nu(\mathbf{n})
\end{aligned}$$

and we have:

$$\mid \Pr(\mathsf{PRG}, \mathsf{pass}(1^{\mathbf{n}}) \cdot \mathsf{view}(\mathtt{D}_{\mathsf{g}})) - \Pr(\mathsf{RND}, \mathsf{pass}(1^{\mathbf{n}}) \cdot \mathsf{view}(\mathtt{D}_{\mathsf{g}})) \mid = \mid \mu(\mathbf{n}) - \nu(\mathbf{n}) \mid$$

so we have that at least one trace between $\mathsf{pass}(1^{\mathbf{n}}) \cdot \mathsf{view}(\mathtt{D}_{\mathsf{f}})$ and $\mathsf{pass}(1^{\mathbf{n}}) \cdot \mathsf{view}(\mathtt{D}_{\mathsf{g}})$ separates PRG and RND and this is a contradiction.

7.2 El-Gamal Encryption Scheme

We propose now another example to test our approach: the *El-Gamal Encryption Scheme*.

In order to describe the El-Gamal encryption scheme we have to introduce some different functions that are necessary to explain how the system works; we won't give in details the terms that represent these functions, but we know by the properties of RSLR that they can be easily implemented.

- The first function we introduce is $\mathbf{zr}(\mathbf{n})$; this function takes in input the security parameter $\mathbf{n}$ and returns a random string lesser or equal than $\mathbf{n}$ (every string can be seen as the binary representation of an integer). This function is necessary because we will work with a finite cyclic group whose cardinality can be different from a power of 2.

- The second function is the concatenation of bitstrings that we indicate with $\circ$; this function takes in input two strings and return a string that is the concatenation of the two received in input. We will use the notation $\underline{\mathrm{m}} \circ \underline{\mathrm{n}}$ for $\circ\ \underline{\mathrm{m}}\ \underline{\mathrm{n}}$.

- The last two function are π_1, π_2: these functions take in input a string $\underline{\mathrm{m}}$ and the security parameter $\mathbf{n}$ and returns the string received in input without the first (or the last in the case of π_2) $\lceil \log_2 \mathbf{n} \rceil$ bit of the string. For example we have $\pi_1(\underline{\mathrm{m}} \circ \underline{\mathrm{n}}) \rightarrow \underline{\mathrm{n}}$ and $\pi_2(\underline{\mathrm{m}} \circ \underline{\mathrm{n}}) \rightarrow \underline{\mathrm{m}}$

- Finally we have to suppose that we have the possibility to encode a function that evaluates the exponential, we will write g^{m} for g raise to the power m and a function that encodes multiplication ,we will write $\mathsf{g}_1 * \mathsf{g}_2$ for g_1 times g_2.

Now we are able to introduce the El-Gamal encryption scheme. Given a cyclic group G of order $\mathbf{n}$ and generator g the El-Gamal encryption scheme is defined by the triple

$(\mathsf{GEN}, \mathsf{ENC}, \mathsf{DEC})$, where:

$$\begin{aligned}\mathsf{GEN} =& \lambda n.(\lambda x.\mathsf{g}^x \circ x)(\mathbf{zr}\ n)\\ \mathsf{ENC} =& \lambda n.\lambda m.\lambda pk.(\lambda y.(\mathsf{g}^y \circ pk^y * mn))(\mathbf{zr}\ n)\\ \mathsf{DEC} =& \lambda n.\lambda c.\lambda sk.(\pi_1(c) * \pi_2(c)^{-sk})\end{aligned}$$

where $m : \mathsf{aStr} \to \mathsf{Str}$ is a varaible for a function that returns a bitstring distribution depending on the security parameter.

We can consider a cyphertext obtained by the El-Gamal cryptosystem as:

$$\mathsf{c} = \lambda n.\lambda m.(\lambda pk.\lambda y.(pk \circ \mathsf{g}^y \circ pk^y * mn))(\pi_2\mathsf{GEN}(n))(\mathbf{zr}(n))$$

that is the concatenation of the public key pk with g^y and with the encryption of the message $pk^y * m$. In order to ensure the security of the cryptosystem we want to prove that a cyphertext buildt by this system is indistinguishable from a string where the final part (i.e. the encryption of the message) is chosen randomly, that is:

$$\mathsf{rm} = \lambda n.\lambda m.(\lambda pk.\lambda y.\lambda z.(pk \circ \mathsf{g}^y \circ \mathsf{g}^z))(\pi_2\mathsf{GEN}(n))(\mathbf{zr}(n))(\mathbf{zr}(n))$$

The security proof of the El-Gamal encryption system is based on the DDH assumpion: it says that an adversary in unable to distinguish between $(\mathsf{g}^\mathsf{x}, \mathsf{g}^\mathsf{y}, \mathsf{g}^{\mathsf{xy}})$ and $(\mathsf{g}^\mathsf{x}, \mathsf{g}^\mathsf{y}, \mathsf{g}^\mathsf{z})$, with $\mathsf{x}, \mathsf{y}, \mathsf{z}$ chosen randomly. We convert this assumption in our language by saying that, if:

$$\begin{aligned}\mathsf{DDH}_1 =& \lambda n.(\lambda x.\lambda y.\mathsf{g}^x \circ \mathsf{g}^y \circ \mathsf{g}^{xy})(\mathbf{zr}(n))(\mathbf{zr}(n))\\ \mathsf{DDH}_2 =& \lambda n.(\lambda x.\lambda y.\lambda.z.\mathsf{g}^x \circ \mathsf{g}^y \circ \mathsf{g}^z)(\mathbf{zr}(n))(\mathbf{zr}(n))(\mathbf{zr}(n))\end{aligned}$$

then $\mathsf{DDH}_1 \equiv \mathsf{DDH}_2$.

Now, in order to prove that $\mathsf{c} \simeq^\mathsf{T}_\mathbf{n} \mathsf{rm}$ we will show that if this is not true then we will be able to distinguish between DDH_1 and DDH_2. So, suppose there exists T such that:

$$|\Pr(\mathsf{c}, \mathsf{pass}(\mathbf{n}) \cdot \mathsf{T}) - \Pr(\mathsf{rm}, \mathsf{pass}(\mathbf{n}) \cdot \mathsf{T})| > \varepsilon(\mathbf{n})$$

for all ε negligible functions.
We have $\mathsf{T} = \mathsf{pass}(\mathsf{f}) \cdot \mathsf{view}(\mathtt{D})$ and, if $\mathsf{f}\mathbf{n} \to \{(\mathsf{g}^{\mathsf{m}})^{p_{\mathsf{m}}}\}_{\mathsf{m} \in \mathsf{S}(\mathsf{f}\mathbf{n})}$ then:

$$\mathsf{c} \Rightarrow^{\mathsf{pass}(\mathbf{n}) \cdot \mathsf{pass}(\mathsf{f})} \{(\mathsf{g}^{\mathsf{x}} \circ \mathsf{g}^{\mathsf{y}} \circ \mathsf{g}^{\mathsf{xy}+\mathsf{m}})^{\frac{1}{\mathbf{n}^2} \cdot p_{\mathsf{m}}}\}_{\mathsf{x},\mathsf{y} \in \mathsf{G}, \mathsf{g}^{\mathsf{m}} \in \mathsf{S}(\mathsf{f}\mathbf{n})}$$

$$\mathsf{rm} \Rightarrow^{\mathsf{pass}(\mathbf{n}) \cdot \mathsf{pass}(\mathsf{f})} \{(\mathsf{g}^{\mathsf{x}} \circ \mathsf{g}^{\mathsf{y}} \circ \mathsf{g}^{\mathsf{z}})^{\frac{1}{\mathbf{n}^3}}\}_{\mathsf{x},\mathsf{y},\mathsf{z} \in \mathsf{G}}$$

that means:

$$\Pr(\mathsf{c}, \mathsf{pass}(\mathbf{n}) \cdot \mathsf{T}) = \sum_{\mathsf{x},\mathsf{y} \in \mathsf{G}, \mathsf{g}^{\mathsf{m}} \in \mathsf{S}(\mathsf{f}\mathbf{n})} \frac{1}{\mathbf{n}^2} \cdot p_{\mathsf{m}} \cdot [\![\mathtt{D}(\mathsf{g}^{\mathsf{x}} \circ \mathsf{g}^{\mathsf{y}} \circ \mathsf{g}^{\mathsf{xy}+\mathsf{m}})]\!](\underline{\epsilon}) = \xi(\mathbf{n})$$

$$\Pr(\mathsf{rm}, \mathsf{pass}(\mathbf{n}) \cdot \mathsf{T}) = \sum_{\mathsf{x},\mathsf{y},\mathsf{z} \in \mathsf{G}} \frac{1}{\mathbf{n}^3} \cdot [\![\mathtt{D}(\mathsf{g}^{\mathsf{x}} \circ \mathsf{g}^{\mathsf{y}} \circ \mathsf{g}^{\mathsf{z}})]\!](\underline{\epsilon}) = \mu(\mathbf{n})$$

So if we consider $\mathsf{T}' = \mathsf{view}(\mathtt{D}')$ with:

$$\mathtt{D}' = \lambda w.\mathtt{D}((\pi_2 w) \circ (\pi_1((\pi_1 w)) * \mathsf{f}(\mathsf{GEN}^{-1} w)))$$

we have:

$$\mathsf{DDH}_1 \Rightarrow^{\mathsf{pass}(\mathbf{n})} \{(\mathsf{g}^{\mathsf{x}} \circ \mathsf{g}^{\mathsf{y}} \circ \mathsf{g}^{\mathsf{xy}})^{\frac{1}{\mathbf{n}^2}}\}_{\mathsf{x},\mathsf{y} \in \mathsf{G}}$$

$$\mathsf{DDH}_2 \Rightarrow^{\mathsf{pass}(\mathbf{n})} \{(\mathsf{g}^{\mathsf{x}} \circ \mathsf{g}^{\mathsf{y}} \circ \mathsf{g}^{\mathsf{z}})^{\frac{1}{\mathbf{n}^3}}\}_{\mathsf{x},\mathsf{y},\mathsf{z} \in \mathsf{G}}$$

and so:

$$\begin{aligned}\Pr(\mathsf{DDH}_1, \mathsf{pass}(\mathbf{n}) \cdot \mathsf{T}') &= \sum_{\mathsf{x},\mathsf{y}\in\mathsf{G}} \frac{1}{\mathbf{n}^2} \cdot [\![\mathtt{D}'(\mathsf{g}^{\mathsf{x}} \circ \mathsf{g}^{\mathsf{y}} \circ \mathsf{g}^{\mathsf{xy}})]\!](\underline{\epsilon}) = \\ &= \sum_{\mathsf{x},\mathsf{y}\in\mathsf{G},\mathsf{g}^{\mathsf{m}}\in\mathsf{S}(\mathsf{f}\mathbf{n})} \frac{1}{\mathbf{n}^2} \cdot p_{\mathsf{m}} \cdot [\![\mathtt{D}(\mathsf{g}^{\mathsf{x}} \circ \mathsf{g}^{\mathsf{y}} \circ \mathsf{g}^{\mathsf{xy}+\mathsf{m}})]\!](\underline{\epsilon}) = \xi(\mathbf{n})\end{aligned}$$

$$\begin{aligned}\Pr(\mathsf{DDH}_2, \mathsf{pass}(\mathbf{n}) \cdot \mathsf{T}') &= \sum_{\mathsf{x},\mathsf{y},\mathsf{z}\in\mathsf{G}} \frac{1}{\mathbf{n}^3} \cdot [\![\mathtt{D}'(\mathsf{g}^{\mathsf{x}} \circ \mathsf{g}^{\mathsf{y}} \circ \mathsf{g}^{\mathsf{z}})]\!](\underline{\epsilon}) = \\ &= \sum_{\mathsf{x},\mathsf{y}\in\mathsf{G}} \frac{1}{\mathbf{n}^2} \cdot \sum_{\mathsf{z}\in\mathsf{G},\mathsf{g}^{\mathsf{m}}\in\mathsf{S}(\mathsf{f}\mathbf{n})} \frac{1}{\mathbf{n}} \cdot p_{\mathsf{m}} \cdot [\![\mathtt{D}(\mathsf{g}^{\mathsf{x}} \circ \mathsf{g}^{\mathsf{y}} \circ \mathsf{g}^{\mathsf{z}+\mathsf{m}})]\!](\underline{\epsilon}) = \\ &= \sum_{\mathsf{x},\mathsf{y},\mathsf{z}\in\mathsf{G}} \frac{1}{\mathbf{n}^3} \cdot \sum_{\mathsf{g}^{\mathsf{m}}\in\mathsf{S}(\mathsf{f}\mathbf{n})} p_{\mathsf{m}} \cdot [\![\mathtt{D}(\mathsf{g}^{\mathsf{x}} \circ \mathsf{g}^{\mathsf{y}} \circ \mathsf{g}^{\mathsf{z}})]\!](\underline{\epsilon}) = \\ &= \sum_{\mathsf{x},\mathsf{y},\mathsf{z}\in\mathsf{G}} \frac{1}{\mathbf{n}^3} \cdot [\![\mathtt{D}(\mathsf{g}^{\mathsf{x}} \circ \mathsf{g}^{\mathsf{y}} \circ \mathsf{g}^{\mathsf{z}})]\!](\underline{\epsilon}) = \mu(\mathbf{n})\end{aligned}$$

So we have :

$$|\Pr(\mathsf{DDH}_1, \mathsf{pass}(\mathbf{n}) \cdot \mathsf{T}') - \Pr(\mathsf{DDH}_2, \mathsf{pass}(\mathbf{n}) \cdot \mathsf{T})| = |\xi(\mathbf{n}) - \mu(\mathbf{n})| > \varepsilon(\mathbf{n})$$

for all ε negligible and so a contradiction.

Chapter 8

Conclusions

In this thesis we started from an overview of the actual scenario in the field of cryptographic proofs; as we said cryptographic proofs are becoming more and more complex so it is very difficult to give it by hand, furthermore, by this reason, it is necessary to give the possibility to analyze the proof itself.

In order to help the cryptographer community, there have been developed several approaches: we presented three of them based on formal methods, automated tools and process calculi. As we have seen, the game-based proof is one of the most used standard to give a cryptographic proof. A key point in such proofs are game transitions, that needs to be justified; these transitions are often justified by saying that the two games are computationally indistinguishable and this is the key point of this thesis. We focus our attention on computational indistinguishability, and we give a characterization of CI by using traces instead of arbitrary algorithms.

We started by studying notions of equivalences and metrics in a language for higher order probabilistic polynomial time computation, that is very useful to model primitives and adversaries. More specifically, we have shown that the discriminating power of linear contexts can be captured by traces, both when equivalences and metrics are considered. Moreover, we gave evidence on how applicative bisimilarity is a sound, but not fully abstract, methodology for context equivalence.

We believe, however, 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 [38, 37], 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.

Finally we gave concrete cryptographic examples in which we proved the secrecy of an encryption induced by a pseudorandom generator, by using the parametric trace equivalence we defined; the result we offer is a formal reduction proof in which each step is mathematically justified without external assumptions.

For what concerns real cryptographic applications, the careful reader should have spotted some limitations to the method we propose. One of the doubt that arises is about the linearity of the contexts; roughly speaking, this constraint doesn't permit to an adversary to copy the term he is testing. Endowing the observer with this capability would allow him to pass different arguments to the same term and observe multiple outputs; by this way it would be impossible for a simple trace to catch the differences seen by the observer. This faculty is a common standard in cryptography, where an adversary is allowed to do multiple (polynomially bounded) queries to an oracle that implements a cryptographic primitive.

Another limitation can be found in the calculus itself, that doesn't admit pairs and projections. As seen in the El-Gamal example, we can overcome this restriction in the case of terms of base type Str by postulating the existence of functions that concatenate and separate bitstrings, but we are not able to manage the case of pairs where terms are of higher order type.

A possible solution for both problems is the definition of a different framework, a framework in which we don't consider single terms but we rather work on tuples of indexed terms $[t_1^1, ..., t_n^n]$ [18]. In this framework traces are modeled so that they take into account the index of the term we want to perform the action, for instance

if we want to pass a value v to the j-th term $\lambda x.t^j$ the action will be in the form $\mathsf{pass}(j, v)$; the observation can be performed only of all the elements of the tuple are strings and it is made on the whole tuple, not on single values. Another feature of this framework is the management of pairs by giving the possibility to split them into new tuples; suppose to have a tuple of the form $[..., \langle s_1, s_2 \rangle^j, ..]$, then we insert a new action unfold that split the pair into two different terms such that:

$$[..., \langle s_1, s_2 \rangle^j, ..] \rightarrow^{\mathsf{unfold}^j} [..., s_1^j, s_2^k, ..]$$

so that we have a new tuple and we are able to pass argument to both terms of the pair.

We conclude with a final consideration: Theorem 6.1 and Theorem 6.2 state that two terms $t, s : \mathsf{aStr} \rightarrow \mathsf{Str}$ are computational indistinguishability if and only if they are parametrically trace equivalent, but there is a question that naturally becomes apparent: what happens if the type of the terms is a generic $\mathsf{aStr} \rightarrow \mathsf{A}$ with A of higher order type? Actually, we don't have a definition of *higher order computational indistinguishability* present in literature (at any rate, we don't have a *formal and precise* one), so a comparison with parametric context equivalence is difficult.

Furthermore, it seems that linear contexts do not capture equivalence as traditionally employed in cryptography already when A is a first-order type $\mathsf{bStr} \rightarrow \mathsf{Str}$. The simplest example can be found in the definition of *pseudo-randomness* which can be spelled out for functions, giving rise to the concept of pseudorandom functions [29]. Formally, a function $F : \{0,1\}^* \rightarrow \{0,1\}^* \rightarrow \{0,1\}^*$ 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 $F(s)$. 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, if s is a term that returns $\mathbf{n}$ bits drawn at random, the RSLR term $t_G s$ that implements the function G above can be proved *trace equivalent* to a term r that returns a truly random function from $\mathbf{n}-$bitstrings to $\mathbf{n}-$bitstrings.

However, investigating into higher-order computational indistinguishability is out of the scope of this thesis, but we believe that these argumentations could be a first step towards a formal analysis of this theme.

References

[1] Martín Abadi, Mathieu Baudet, and Bogdan Warinschi. Guessing attacks and the computational soundness of static equivalence. In *Foundations of Software Science and Computation Structures, 9th International Conference, FOSSACS 2006, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2006, Vienna, Austria, March 25-31, 2006, Proceedings*, pages 398–412, 2006.

[2] Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography (the computational soundnessof formal encryption). In *Theoretical Computer Science, Exploring New Frontiers of Theoretical Informatics, International Conference IFIP TCS 2000, Sendai, Japan, August 17-19, 2000, Proceedings*, pages 3–22, 2000.

[3] Pedro Adão, Gergei Bana, Jonathan Herzog, and Andre Scedrov. Soundness of formal encryption in the presence of key-cycles. In *Computer Security - ESORICS 2005, 10th European Symposium on Research in Computer Security, Milan, Italy, September 12-14, 2005, Proceedings*, pages 374–396, 2005.

[4] Pedro Adão, Gergei Bana, and Andre Scedrov. Computational and information-theoretic soundness and completeness of formal encryption. In *18th IEEE Computer Security Foundations Workshop, (CSFW-18 2005), 20-22 June 2005, Aix-en-Provence, France*, pages 170–184, 2005.

[5] Gergei Bana and Hubert Comon-Lundh. Towards unconditional soundness: Computationally complete symbolic attacker. In *Principles of Security and Trust - First International Conference, POST 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012, Proceedings*, pages 189–208, 2012.

[6] Gergei Bana and Hubert Comon-Lundh. A computationally complete symbolic attacker for equivalence properties. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, pages 609–620, 2014.

[7] Gergei Bana, Payman Mohassel, and Till Stegers. Computational soundness of formal indistinguishability and static equivalence. In *Advances in Computer Science - ASIAN 2006. Secure Software and Related Issues, 11th Asian Computing Science Conference, Tokyo, Japan, December 6-8, 2006, Revised Selected Papers*, pages 182–196, 2006.

[8] Gilles Barthe, Benjamin Grégoire, and Santiago Zanella Béguelin. Formal certification of code-based cryptographic proofs. In *Proceedings of the 36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2009, Savannah, GA, USA, January 21-23, 2009*, pages 90–101, 2009.

[9] Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella Béguelin. Computer-aided security proofs for the working cryptographer. pages 71–90, 2011.

[10] Mathieu Baudet, Véronique Cortier, and Steve Kremer. Computationally sound implementations of equational theories against passive adversaries. In *Automata, Languages and Programming, 32nd International Colloquium, ICALP 2005, Lisbon, Portugal, July 11-15, 2005, Proceedings*, pages 652–663, 2005.

[11] Mathieu Baudet, Véronique Cortier, and Steve Kremer. Computationally sound implementations of equational theories against passive adversaries. *Inf. Comput.*, 207(4):496–520, 2009.

[12] Stephen Bellantoni and Stephen A. Cook. A new recursion-theoretic characterization of the polytime functions (extended abstract). In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada*, pages 283–293, 1992.

[13] Mihir Bellare and Phillip Rogaway. Code-based game-playing proofs and the security of triple encryption. *IACR Cryptology ePrint Archive*, 2004:331, 2004.

[14] Bruno Blanchet. A computationally sound mechanized prover for security protocols. *IACR Cryptology ePrint Archive*, 2005:401, 2005.

[15] Emmanuel Bresson, Yassine Lakhnech, Laurent Mazaré, and Bogdan Warinschi. A generalization of DDH with applications to protocol analysis and computational soundness. In *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings*, pages 482–499, 2007.

[16] Alan Cobham. The intrinsic computational difficulty of functions. In *International Congress for Logic, Methodology and the Philosophy of Science*, pages 24–30, 1964.

[17] Raphaëlle Crubillé and Ugo Dal Lago. On probabilistic applicative bisimulation and call-by-value λ-calculi. In *Programming Languages and Systems - 23rd European Symposium on Programming, ESOP 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings*, pages 209–228, 2014.

[18] Raphaëlle Crubillé and Ugo Dal Lago. Metric reasoning about λ-terms: The affine case. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 633–644, 2015.

[19] Yuxin Deng and Yu Zhang. Program equivalence in linear contexts. *Theor. Comput. Sci.*, 585:71–90, 2015.

[20] Danny Dolev and Andrew Chi-Chih Yao. On the security of public key protocols (extended abstract). In *22nd Annual Symposium on Foundations of Computer Science, Nashville, Tennessee, USA, 28-30 October 1981*, pages 350–357, 1981.

[21] Flavio D. Garcia and Peter van Rossum. Sound and complete computational interpretation of symbolic hashes in the standard model. *Theor. Comput. Sci.*, 394(1-2):112–133, 2008.

[22] Oded Goldreich. *The Foundations of Cryptography - Volume 1, Basic Techniques*. Cambridge University Press, 2001.

[23] Oded Goldreich and Madhu Sudan. Computational indistinguishability: A sample hierarchy. In *Proceedings of the 13th Annual IEEE Conference on Computational Complexity, Buffalo, New York, USA, June 15-18, 1998*, pages 24–33, 1998.

[24] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.

[25] Jonathan Herzog. A computational interpretation of dolev-yao adversaries. *Theor. Comput. Sci.*, 340(1):57–81, 2005.

[26] Martin Hofmann. A mixed modal/linear lambda calculus with applications to bellantoni-cook safe recursion. In *Computer Science Logic, 11th International Workshop, CSL '97, Annual Conference of the EACSL, Aarhus, Denmark, August 23-29, 1997, Selected Papers*, pages 275–294, 1997.

[27] Omer Horvitz and Virgil D. Gligor. Weak key authenticity and the computational completeness of formal encryption. In *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, pages 530–547, 2003.

[28] Russell Impagliazzo and Bruce M. Kapron. Logics for reasoning about cryptographic constructions. In *44th Symposium on Foundations of Computer Science (FOCS 2003), 11-14 October 2003, Cambridge, MA, USA, Proceedings*, pages 372–383, 2003.

[29] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography.* Chapman and Hall/CRC Press, 2007.

[30] Ugo Dal Lago, Davide Sangiorgi, and Michele Alberti. On coinductive equivalences for higher-order probabilistic functional programs. In *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 297–308, 2014.

[31] Ugo Dal Lago and Paolo Parisen Toldin. A higher-order characterization of probabilistic polynomial time. In *Foundational and Practical Aspects of Resource Analysis - Second International Workshop, FOPARA 2011, Madrid, Spain, May 19, 2011, Revised Selected Papers*, pages 1–18, 2011.

[32] Peeter Laud. Encryption cycles and two views of cryptography. In *Nordic Workshop on Secure IT Systems - NORDSEC'02*, 2002.

[33] Peeter Laud and Ricardo Corin. Sound computational interpretation of formal encryption with composed keys. In *Information Security and Cryptology - ICISC 2003, 6th International Conference, Seoul, Korea, November 27-28, 2003, Revised Papers*, pages 55–66, 2003.

[34] Daniele Micciancio and Bogdan Warinschi. Completeness theorems for the abadi-rogaway language of encrypted expressions. *Journal of Computer Security*, 12(1):99–130, 2004.

[35] John C. Mitchell, Mark Mitchell, and Andre Scedrov. A linguistic characterization of bounded oracle computation and probabilistic polynomial time. In *39th Annual Symposium on Foundations of Computer Science, FOCS '98, November 8-11, 1998, Palo Alto, California, USA*, pages 725–733, 1998.

[36] John C. Mitchell, Ajith Ramanathan, Andre Scedrov, and Vanessa Teague. A probabilistic polynomial-time process calculus for the analysis of cryptographic protocols. *Theor. Comput. Sci.*, 353(1-3):118–164, 2006.

[37] David Nowak and Yu Zhang. A calculus for game-based security proofs. In *Provable Security - 4th International Conference, ProvSec 2010, Malacca, Malaysia, October 13-15, 2010. Proceedings*, pages 35–52, 2010.

[38] Yu Zhang. The computational SLR: a logic for reasoning about computational indistinguishability. *Mathematical Structures in Computer Science*, 20(5):951–975, 2010.