J. Goubault−Larrecq, S. Lasota, D. Nowak and Yu Zhang

Complete Lax Logical Relations for Cryptographic Lambda−Calculi

Research Report LSV−04−4, Feb. 2004

Complete Lax Logical Relations for Cryptographic Lambda-Calculi*

Jean Goubault-Larrecq[1] Sławomir Lasota[2] David Nowak[1]
Yu Zhang[1†]
[1] LSV/CNRS UMR 8643 & INRIA Futurs projet SECSI & ENS Cachan
61, av. du Président Wilson, 94235 Cachan Cedex, France
{goubault,nowak,zhang}@lsv.ens-cachan.fr
[2] Institute of Informatics, Warsaw University, ul. Banacha 2, 02-097 Warszawa, Poland
sl@mimuw.edu.pl

Abstract

Security properties are profitably expressed using notions of contextual equivalence, and logical relations are a powerful proof technique to establish contextual equivalence in typed lambda calculi, see e.g. Sumii and Pierce's logical relation for a cryptographic lambda-calculus. We clarify Sumii and Pierce's approach, showing that the right tool is prelogical relations, or lax logical relations in general: relations should be lax at encryption types, notably. To explore the difficult aspect of fresh name creation, we use Moggi's monadic lambda-calculus with constants for cryptographic primitives, and Stark's name creation monad. We define logical relations which are lax at encryption and function types but strict (non-lax) at various other types, and show that they are sound and complete for contextual equivalence at all types.

1. Introduction

There are nowadays many existing models for cryptographic protocol verification. The most well-known are perhaps the Dolev-Yao model (after [9], see [7] for a survey) and the spi-calculus of [1]. A lesser known model was introduced by Sumii and Pierce [21], the *cryptographic lambda-calculus*. This has certain advantages; notably, higher-order behaviors are naturally taken into account, which is ignored in other models (although, at the moment, higher order is not perceived as a needed feature in cryptographic protocols). Better, second-order terms naturally encode asymmetric encryption. It may also be appealing to consider that proving security properties in the cryptographic lambda-calculus can be achieved through the use of well-crafted *logical relations*, a tool that has been used many times with considerable success in the λ-calculus: see [13, Chapter 8], for numerous examples. Sumii and Pierce [21] in particular define three logical relations that can be used to establish contextual equivalence, hence prove security properties, but completeness remains open.

Our contributions are twofold: first, we clarify the import of Sumii and Pierce as far as the behavior of logical relations on encryption types is concerned, and simplify it to the point that we reduce it to prelogical relations [11] and more generally to lax logical relations [17]; while standard recourses to the latter were usually required because of arrow types, here we require the logical relations to be lax at *encryption types*. Second, we prove various completeness results: two terms are contextually equivalent if and only if they are related by some lax logical relation. This holds at all types, not just first-order types as in previous works. An added bonus of using lax logical relations is that they extend directly to more complex models of encryption, where cryptographic primitives may obey algebraic laws. This is touched upon briefly in Section 3.7.

Outline. We survey related work in Section 2. We focus on the approach of Sumii and Pierce, in which they define several rather complex logical relations as sound criteria of contextual equivalence. We take a new look at this approach in Section 3, and gradually deconstruct their work to the point where we show the power of prelogical relations in action. This is shown in the absence of fresh name creation, for added clarity. We tackle the difficult issue of names in Section 4, using Moggi's elegant computational λ-calculus framework with Stark's name creation monad.

*Partially supported by the RNTL project Prouvé, the ACI Sécurité Informatique Rossignol, the ACI jeunes chercheurs "Sécurité informatique, protocoles cryptographiques et détection d'intrusions", and the ACI Cryptologie "PSI-Robuste".
†PhD student under an MENRT grant on ACI Cryptologie funding, École Doctorale Sciences Pratiques (Cachan).

This requires passing from simple set-valued semantics to presheaf-valued semantics, and from prelogical relations to lax logical relations [17]. Again, we prove soundness and completeness of the latter with respect to contextual equivalence. In Section 4, following insights developed in previous sections, we shall consider a computational λ-calculus with an open set of types and constants (i.e., you may add the ones you wish); this handles cryptography, in particular.

2. Related Work

Logical relations have often been used to prove various properties of typed lambda calculi. The list would be too long here, see [13] for a sampler. We are interested here in using logical relations or variants thereof as sound criteria for establishing *contextual equivalence* of two programs. This is instrumental in defining security properties. As noticed in [1, 21], a datum M of type τ is *secret* in some term $t(M)$ of type τ' if and only if no intruder can say anything about M just by looking at $t(M)$, i.e., if and only if $t(M) \approx_{\tau'} t(M')$ for any two M and M', where $\approx_{\tau'}$ denotes contextual equivalence at type τ'; or equivalently, if and only if $\lambda x \cdot \lambda y \cdot t(x) \approx_{\tau \to \tau \to \tau'} \lambda x \cdot \lambda y \cdot t(y)$. We are using λ-calculus notions here, following [21], but the idea of using contextual equivalence to define security properties was pioneered by Abadi and Gordon [1], where not just secrecy, but also authentication, is investigated.

We shall define precisely what we mean by contextual equivalence in a calculus without names (Section 3.2), then with names (Section 4.3). Both notions are standard, the latter being inspired by [16], only adapted to Moggi's computational λ-calculus [15]. In [16] and some other places, this kind of equivalence, which states that two values (or terms) a and a' are equivalent provided every context of type `bool` must give identical results on a and on a', is called observational equivalence. We stress that this should not be confused with observational equivalence as it is defined for data refinement [13], where *models* are related, not *values* in the same model as here. For completeness of the data refinement kind, see [18].

The main point in passing from contextual equivalence to logical relations is to avoid the universal quantification over contexts in the former. But there are two kinds of technical difficulties one must face in defining logical relations for cryptographic λ-calculi. The first, and hardest one, is *fresh name creation*. The second is dealing with encryption and decryption. We shall see that the latter has an elegant solution in terms of *prelogical* relations [11], which we believe is both simpler and more general than Sumii and Pierce's proposal [21]; this is described in Section 3, although we ignore fresh name creation there, for clarity.

Dealing with fresh name creation is harder. The work of Sumii and Pierce [21] is inspired in this respect by Pitts and Stark [16], who proposed a λ-calculus devoted to the study of fresh name creation, the *nu-calculus*. They define a so-called operational logical relation to establish observational equivalence of nu-calculus expressions. They prove that this logical relation is complete up to first-order types. However, the extension to richer calculi (e.g., [21]) must be made on an ad hoc basis, probably because this operational logical relation is defined by syntactic means. We shall rest solely on semantic notions. That our notions can be extended at will is witnessed by the fact that we shall consider *open* calculi, in the sense that our calculi must include specific types and constants, but may contain any set of additional types and constants, too.

Contextual equivalence is defined using universal quantification over contexts, which is impractical. We were tempted to say that the logical relations of [21] and [16] did a poor job of dispensing with this, since they quantify over so-called canonical forms of various types, which is basically as complex. However, this would be unfair, since we do even worse. Indeed, one of our points in this paper is that completeness *at all types* can be obtained by replacing logical relations by *prelogical* [11] and even *lax* logical relations [17], and these are even harder to decide. We therefore examine on which types the lax logical relations we need can be taken to be strict (i.e., non-lax, see later). One nice point in our establishing completeness for lax logical relations is that it gives a precise meaning to the notion of *public* names, as used in the intuition behind the constructions of [16] for example: see Section 4.3.

In [10], Goubault-Larrecq, Lasota and Nowak define a Kripke logical relation for the dynamic name creation monad, which is extended by Zhang and Nowak in [23] so that it coincides with Pitts and Stark's operational logical relation up to first-order types. It rests on purely semantic constituents, and dispenses with the detours through operational semantics that Pitts and Stark use. We continue this work here, relying on the elegance of Moggi's [15] computational λ-calculus to describe side effects, and in particular fresh name creation, using Stark's insights [20].

Further comparisons will be made in the course of this paper, especially with bisimulations for spi-calculus [1, 5, 6]. This continues the observations pioneered in [10], where notions of logical relations for various monads (non-determinism, probabilistic non-determinism) were shown to be proper extensions of known notions of bisimulations. The precise relation with hedged and framed bisimulation [6] remains to be stated precisely.

3. Deconstructing Sumii and Pierce, Without Names

The starting point of this paper was the realization that the rather complex family of logical relations proposed by Sumii and Pierce [21] could be simplified in such a way that it could be described as merely *one* way of building logical relations that have all desired properties. It turned out that the only property we really need to be able to deal with encryption and decryption primitives is that the logical relations should relate the encryption function with itself, and the decryption function with itself. At this point, the reader familiar with *pre*logical relations will realize that this is exactly what prelogical relations are about, but let us not advance so fast. For now, we are content with explaining the above remark.

3.1. The Toy Cryptographic λ-Calculus

To show the idea in action, let us use a minimal extension of the simply-typed λ-calculus with encryption and decryption, and let's call it the *toy cryptographic λ-calculus*. We shall show how the idea works on this calculus, which is just a fragment of Sumii and Pierce's [21] cryptographic λ-calculus. The main thing that is missing here is nonce creation, i.e., fresh name creation. Dealing with nonces is the difficult point, see Section 4.

So, for now, restrict the types to:

$$\tau ::= b \mid \tau_1 \to \tau_2 \mid \mathtt{key}[\tau] \mid \mathtt{bits}[\tau]$$

where b ranges over a set Σ of so-called *base types*, e.g., integers, booleans, etc. Sumii and Pierce's calculus in addition has cartesian product and coproduct types. $\mathtt{key}[\tau]$ is the type of (symmetric) keys that can be used to encrypt values of type τ, $\mathtt{bits}[\tau]$ is the type of *ciphertexts* obtained by encrypting some value of type τ—necessarily with a key of type $\mathtt{key}[\tau]$. There is no special type for nonces, which are being thought as objects of type $\mathtt{key}[\tau]$ for some τ.

That the universe of values is split in so many types is, by the way, a flaw in this model. In particular, this model may conceal *type confusion attacks*, where a value of some type τ is expected, but a value of some other type τ' is received. Sumii and Pierce claim that such attacks can be prevented by standard dynamic type checking. We return to this point in Section 3.7, and show how type confusion attacks can be modeled at no cost.

The terms of the toy cryptographic λ-calculus are given by the grammar:

$$\begin{array}{lll} t, u, v, ... & ::= & x \mid \lambda x \cdot t \mid tu \mid \{t\}_u \\ & & \mid \mathtt{let}\ \{x\}_t = u\ \mathtt{in}\ v_1\ \mathtt{else}\ v_2 \end{array}$$

where x ranges over a countable set of variables, $\{t\}_u$ denotes the ciphertext obtained by encrypting t with key u (t is then called the *plaintext*), and $\mathtt{let}\ \{x\}_t = u\ \mathtt{in}\ v_1\ \mathtt{else}\ v_2$ is meant to evaluate u, attempt to decrypt it using key t, then proceed to evaluate v_1 with plaintext stored in x if decryption succeeded, or evaluate v_2 if decryption failed. Definitions of free and bound variables and α-renaming are standard, hence omitted; x is bound in $\lambda x \cdot t$, with scope t, and x is bound in $\mathtt{let}\ \{x\}_t = u\ \mathtt{in}\ v_1\ \mathtt{else}\ v_2$, with scope v_1.

Typing is as one would expect, see Figure 1. *Judgments* are of the form $\Gamma \vdash t : \tau$, where Γ is a *context*, i.e., a finite mapping from variables to types. If Γ maps x_1 to τ_1, ..., x_n to τ_n, we write it $x_1 : \tau_1, \ldots, x_n : \tau_n$. Γ, Δ denotes the union of the contexts Γ and Δ, provided their domains do not intersect.

A simple denotational semantics for the typed toy cryptographic calculus is as follows. Let $[\![\_]\!]$ be any function mapping types τ to sets so that $[\![\tau_1 \to \tau_2]\!]$ is the set $[\![\tau_1]\!] \to [\![\tau_2]\!]$ of all functions from $[\![\tau_1]\!]$ to $[\![\tau_2]\!]$, for all types τ_1 and τ_2. Let $[\![b]\!]$ be arbitrary for every base type b, $[\![\mathtt{key}[\tau]]\!]$ be arbitrary. For every $V \in [\![\tau]\!]$, $K \in [\![\mathtt{key}[\tau]]\!]$, write $\mathsf{E}(V, K)$ the pair (V, K), to suggest that this really denotes the encryption of V with key K. (That ciphertexts are just modeled as pairs is exactly as in modern versions of the Dolev-Yao model [9], or in the spi-calculus [1].) Then, let $[\![\mathtt{bits}[\tau]]\!]$ be the set of all pairs $\mathsf{E}(V, K)$, $V \in [\![\tau]\!]$, $K \in [\![\mathtt{key}[\tau]]\!]$.

For any set A, let $A_\perp$ be the disjoint sum of A with $\{\perp\}$, where $\perp$ is an element outside A, and let ι be the canonical injection of A into $A_\perp$. While we have defined $\mathsf{E}(V, K)$ as the pair (V, K), we define the inverse decryption function from $[\![\mathtt{bits}[\tau]]\!] \times [\![\mathtt{key}[\tau]]\!]$ to $[\![\tau]\!]_\perp$ by letting $\mathsf{D}(V', K')$ be $\iota(V)$ if V' is of the form (V, K) with $K = K'$, and $\perp$ otherwise.

We then describe the value $[\![t]\!]\,\rho$ of the term t in the environment ρ by structural induction on t, see Figure 2. More formally, for any context Γ, a Γ-*environment* ρ is a map such that, for every $x : \tau$ in Γ, $\rho(x)$ is an element of $[\![\tau]\!]$. Write $\rho[x := V]$ the environment mapping x to V and every other variable y to $\rho(y)$. Write $[x := V]$ the environment mapping just x to V. Then, given any typing derivation π of the judgment $\Gamma \vdash t : \tau$, given any Γ-environment ρ, we define $[\![\pi]\!]\,\rho$ in $[\![\tau]\!]$ by structural induction on π. As is standard, we allow ourselves to write $\Gamma \vdash t : \tau$ or even just t in place of π here, since typing derivations are (almost) isomorphic to the terms t themselves. We write $(V \in A \mapsto f(V))$ the (set-theoretic) function mapping V in A to $f(V)$ to distinguish it from the (syntactic) λ-abstraction $\lambda x \cdot f(x)$. In $[\![\Gamma \vdash tu : \tau_2]\!]\,\rho$, we assume that the premises of the last rule of the implicit typing derivation are $\Gamma \vdash t : \tau_1 \to \tau_2$ and $\Gamma \vdash u : \tau_1$.

$$\frac{}{\Gamma, x : \tau \vdash x : \tau}\,(Var) \qquad \frac{\Gamma, x : \tau_1 \vdash t : \tau_2}{\Gamma \vdash \lambda x \cdot t : \tau_1 \to \tau_2}\,(Lam) \qquad \frac{\Gamma \vdash t : \tau_1 \to \tau_2 \qquad \Gamma \vdash u : \tau_1}{\Gamma \vdash tu : \tau_2}\,(App)$$

$$\frac{\Gamma \vdash t : \tau \qquad \Gamma \vdash u : \mathtt{key}[\tau]}{\Gamma \vdash \{t\}_u : \mathtt{bits}[\tau]}\,(Enc) \qquad \frac{\begin{array}{c}\Gamma \vdash t : \mathtt{key}[\tau] \qquad \Gamma \vdash u : \mathtt{bits}[\tau] \\ \Gamma, x : \tau \vdash v_1 : \tau' \qquad \Gamma \vdash v_2 : \tau'\end{array}}{\Gamma \vdash \mathtt{let}\ \{x\}_t = u\ \mathtt{in}\ v_1\ \mathtt{else}\ v_2 : \tau'}\,(Dec)$$

Figure 1. Typing the toy cryptographic λ-calculus

$$\begin{array}{rcl}
[\![\Gamma, x : \tau \vdash x : \tau]\!]\,\rho & = & \rho(x) \\
[\![\Gamma \vdash \lambda x \cdot t : \tau_1 \to \tau_2]\!]\,\rho & = & (V \in [\![\tau_1]\!] \mapsto [\![\Gamma, x : \tau_1 \vdash t : \tau_2]\!]\,\rho[x := V]) \\
[\![\Gamma \vdash tu : \tau_2]\!]\,\rho & = & [\![\Gamma \vdash t : \tau_1 \to \tau_2]\!]\,\rho([\![\Gamma \vdash u : \tau_1]\!]\,\rho) \\
[\![\Gamma \vdash \{t\}_u : \mathtt{bits}[\tau]]\!]\,\rho & = & \mathsf{E}([\![\Gamma \vdash t : \tau]\!]\,\rho, [\![\Gamma \vdash u : \mathtt{key}[\tau]]\!]) \\
[\![\mathtt{let}\ \{x\}_t = u\ \mathtt{in}\ v_1\ \mathtt{else}\ v_2]\!]\,\rho & = & \left\{\begin{array}{ll}[\![v_1]\!]\,\rho[x := V_1] & \text{if } V = \iota(V_1) \\ [\![v_2]\!]\,\rho & \text{if } V = \bot\end{array}\right. \qquad \text{where } V = \mathsf{D}([\![u]\!]\,\rho, [\![t]\!]\,\rho)
\end{array}$$

Figure 2. A simple denotational semantics for the toy cryptographic λ-calculus

3.2. Contextual Equivalence

Now fix a subset Obs of Σ, of so-called *observation types*. Typically, Obs will contain just the type `bool` of Booleans, one of the base types. We say that two terms u and v such that $\vdash u : \tau$ and $\vdash v : \tau$ are derivable, for the same type τ, are *contextually equivalent*, and we write $u \approx_\tau v$, in the set-theoretic model above if and only if, whatever the term $\mathcal{C}$ such that $x : \tau \vdash \mathcal{C} : o$ is derivable ($o \in \mathsf{Obs}$), $[\![\mathcal{C}]\!]\,[x := [\![u]\!]] = [\![\mathcal{C}]\!]\,[x := [\![v]\!]]$. Intuitively, this (essentially standard) notion captures the fact that we would like to consider u and v as equivalent provided, whatever the question we ask about them, the answer is the same for u and v. Asking a question about t means executing t in a context (a.k.a., an operating system) $\mathcal{C}$ with type `bool`, i.e., running $\mathcal{C}$ when x takes the value of t and testing whether the result is true or false. If the answer differs for $t = u$ and for $t = v$ in the context $\mathcal{C}$, then there is an observable difference between u and v. Taking other observation types than `bool` is certainly possible, but we usually do not want function, key and bits types to be observation types.

3.3. What Are Logical Relations for Encryption?

In the λ-calculus setting, a (binary) *logical relation* is a family $(\mathcal{R}_\tau)_{\tau\ \text{type}}$ of binary relations $\mathcal{R}_\tau$, one for each type τ, on $[\![\tau]\!]$, such that:

(Log) for every $f, f' \in [\![\tau_1 \to \tau_2]\!]$, $f\ \mathcal{R}_{\tau_1 \to \tau_2}\ f'$ if and only if for every $a\ \mathcal{R}_{\tau_1}\ a'$, $f(a)\ \mathcal{R}_{\tau_2}\ f'(a')$.

Here we write $a\ \mathcal{R}\ a'$ to say that a and a' are related by the binary relation $\mathcal{R}$. In other words, logical relations relate exactly those functions that map related arguments to related results. This is the standard definition of logical relations in the λ-calculus [13]. Note that there is no constraint on base types. In the typed λ-calculus, i.e., without encryption and decryption, the condition above forces $(\mathcal{R}_\tau)_{\tau\ \text{type}}$ to be uniquely determined, by induction on types, from the relations $\mathcal{R}_b$, $b \in \Sigma$. More importantly, it entails the so-called *basic lemma*. To state it, first say that two Γ-environments ρ, ρ' are *related* by the logical relation, in notation $\rho\ \mathcal{R}_\Gamma\ \rho'$, if and only if $\rho(x)\ \mathcal{R}_\tau\ \rho'(x)$ for every $x : \tau$ in Γ. The basic lemma states that if $\Gamma \vdash t_0 : \tau$ is derivable, and ρ, ρ' are two related Γ-environments, then $[\![t_0]\!]\,\rho\ \mathcal{R}_\tau\ [\![t_0]\!]\,\rho'$. This is a simple induction on (the typing derivation of) t_0.

We are interested in the basic lemma because, as observed e.g. in [21], this implies that for any logical relation that coincides with equality on observation types, any two terms with logically related values are contextually equivalent. More precisely, assume that $\mathcal{R}_o$ is equality on $[\![o]\!]$ for every $o \in \mathsf{Obs}$. Then, if $\vdash u : \tau$ and $\vdash v : \tau$ are derivable, and $u\ \mathcal{R}_\tau\ v$, then $u \approx_\tau v$. Indeed, by the basic lemma, for every $\mathcal{C}$ such that $x : \tau \vdash \mathcal{C} : o$ is derivable ($o \in \mathsf{Obs}$), then $[\![\mathcal{C}]\!]\,[x := [\![u]\!]]\ \mathcal{R}_o\ [\![\mathcal{C}]\!]\,[x := [\![v]\!]]$, i.e., $[\![\mathcal{C}]\!]\,[x := [\![u]\!]] = [\![\mathcal{C}]\!]\,[x := [\![v]\!]]$ since $\mathcal{R}_o$ is equality.

In the toy cryptographic λ-calculus, we have left the definition of $\mathcal{R}_{\mathtt{key}[\tau]}$ and $\mathcal{R}_{\mathtt{bits}[\tau]}$ open. Here are conditions under which the basic lemma holds in the toy cryptographic λ-calculus. For any type τ, let $\mathcal{R}_{\tau\ \mathtt{option}}$ be the binary relation on $[\![\tau]\!]_\bot$ defined by $V\ \mathcal{R}_{\tau\ \mathtt{option}}\ V'$ if and only if

$V = V' = \bot$, or $V = \iota(V_1)$, $V' = \iota(V_1')$ for some V_1, V_1', and $V_1 \mathcal{R}_\tau V_1'$.

Lemma 1 *Assume that:*
1. for every $V \mathcal{R}_\tau V'$ and $K \mathcal{R}_{\mathtt{key}[\tau]} K'$, $\mathsf{E}(V, K) \mathcal{R}_{\mathtt{bits}[\tau]} \mathsf{E}(V', K')$;
2. for every $V \mathcal{R}_{\mathtt{bits}[\tau]} V'$ and $K \mathcal{R}_{\mathtt{key}[\tau]} K'$, $\mathsf{D}(V, K) \mathcal{R}_{\tau\ \mathtt{option}} \mathsf{D}(V', K')$.
Then the basic lemma holds: if $\Gamma \vdash t_0 : \tau$ is derivable, and ρ, ρ' are two related Γ-environments, then $[\![t_0]\!]\,\rho\,\mathcal{R}_\tau\,[\![t_0]\!]\,\rho'$.

Proof. By structural induction on t_0. The cases of variables, λ-abstractions and applications are standard [13]. The case of encryptions $\{t\}_u$ is by item 1. When $t_0 = \mathtt{let}\ \{x\}_t = u\ \mathtt{in}\ v_1\ \mathtt{else}\ v_2$, by induction hypothesis we have $[\![u]\!]\,\rho\,\mathcal{R}_{\mathtt{bits}[\tau]}\,[\![u]\!]\,\rho'$ and $[\![t]\!]\,\rho\,\mathcal{R}_{\mathtt{key}[\tau]}\,[\![t]\!]\,\rho'$, so, letting $V = \mathsf{D}([\![u]\!]\,\rho, [\![t]\!]\,\rho)$ and $V' = \mathsf{D}([\![u]\!]\,\rho', [\![t]\!]\,\rho')$, then $V \mathcal{R}_{\tau\ \mathtt{option}} V'$ by 2. Therefore, either $V = V' = \bot$, so $[\![t_0]\!]\,\rho = [\![v_2]\!]\,\rho = [\![v_2]\!]\,\rho'$ (by induction hypothesis) $= [\![t_0]\!]\,\rho'$; or $V = \iota(V_1)$, $V' = \iota(V_1')$, and $[\![t_0]\!]\,\rho = [\![v_1]\!]\,\rho[x := V_1] = [\![v_1]\!]\,\rho'[x := V_1']$ (by induction hypothesis) $= [\![t_0]\!]\,\rho'$. □

Before we proceed, let us remark that we have never used *any* property of E or D in the proof of this lemma. The property that $\mathsf{D}(\mathsf{E}(V, K), K) = \iota(V)$ is only needed to show that $\mathtt{let}\ \{x\}_t = \{u\}_t\ \mathtt{in}\ v_1\ \mathtt{else}\ v_2$ and $v_1[x := u]$ have the same semantics, which we do not care about here. The property that $\mathsf{E}(V, K)$ is the pair (V, K), or that E is even injective, is just never needed. This means that Lemma 1 also holds if we use encryption primitives that obey algebraic laws: we shall use this observation in Section 3.7.

There is a kind of converse to Lemma 1. Assume that we have an additional type former $\tau\ \mathtt{option}$, with constructors $\mathtt{SOME} : \tau \to \tau\ \mathtt{option}$ and $\mathtt{NONE} : \tau\ \mathtt{option}$. Assume their semantics is given by $[\![\tau\ \mathtt{option}]\!] = [\![\tau]\!]_\bot$, $[\![\mathtt{SOME}\ t]\!] = \iota([\![t]\!])$, $[\![\mathtt{NONE}]\!] = \bot$. Finally, assume that $\mathcal{R}_{\tau\ \mathtt{option}}$ is defined as above. Then we may define an encryption primitive $enc = \lambda v \cdot \lambda k \cdot \{v\}_k$ and a decryption primitive in the toy cryptographic lambda-calculus by $dec = \lambda v \cdot \lambda k \cdot \mathtt{let}\ \{x\}_k = v\ \mathtt{in}\ \mathtt{SOME}\ x\ \mathtt{else}\ \mathtt{NONE}$. If the basic lemma holds, then we must have $[\![enc]\!]\ \mathcal{R}_{\tau \to \mathtt{key}[\tau] \to \mathtt{bits}[\tau]}\ [\![enc]\!]$ and $[\![dec]\!]\ \mathcal{R}_{\mathtt{bits}[\tau] \to \mathtt{key}[\tau] \to \tau\ \mathtt{option}}\ [\![dec]\!]$. These are just conditions 1. and 2.

Call cryptographic logical relation any logical relation for which the basic lemma holds. Conditions 1. and 2. can therefore be rephrased as the following motto:

A cryptographic logical relation should relate encryption with itself, and decryption with itself.

3.4. Existence of Logical Relations for Encryption

How can we *build* a cryptographic logical relation inductively on types? We first need to address the question of *existence* of logical relations satisfying the basic lemma.

Let us fix a type τ, and assume that we have already constructed $\mathcal{R}_\tau$ and $\mathcal{R}_{\mathtt{key}[\tau]}$. Let $\mathcal{R}^\bot_{\mathtt{bits}[\tau]}$ be the smallest relation on $[\![\mathtt{bits}[\tau]]\!]$ satisfying condition 1., i.e., such that $\mathsf{E}(V, K)\ \mathcal{R}^\bot_{\mathtt{bits}[\tau]}\ \mathsf{E}(V', K)$ for all $V \mathcal{R}_\tau V'$ and $K \mathcal{R}_{\mathtt{key}[\tau]} K'$. Let $\mathcal{R}^\top_{\mathtt{bits}[\tau]}$ be the largest relation on $[\![\mathtt{bits}[\tau]]\!]$ satisfying condition 2., i.e., such that whenever $V \mathcal{R}^\top_{\mathtt{bits}[\tau]} V'$, then $\mathsf{D}(V, K)\ \mathcal{R}_{\tau\ \mathtt{option}}\ \mathsf{D}(V', K')$ for every $K \mathcal{R}_{\mathtt{key}[\tau]} K'$. These two relations clearly exist. Conditions 1. and 2. state that we should choose $\mathcal{R}_{\mathtt{bits}[\tau]}$ so that $\mathcal{R}^\bot_{\mathtt{bits}[\tau]} \subseteq \mathcal{R}_{\mathtt{bits}[\tau]} \subseteq \mathcal{R}^\top_{\mathtt{bits}[\tau]}$. This exists if and only if $\mathcal{R}^\bot_{\mathtt{bits}[\tau]} \subseteq \mathcal{R}^\top_{\mathtt{bits}[\tau]}$.

In turn, $\mathcal{R}^\bot_{\mathtt{bits}[\tau]} \subseteq \mathcal{R}^\top_{\mathtt{bits}[\tau]}$ is equivalent to: for every $V \mathcal{R}_\tau V'$ and $K \mathcal{R}_{\mathtt{key}[\tau]} K'$, for every $K_1 \mathcal{R}_{\mathtt{key}[\tau]} K_1'$, $\mathsf{D}(\mathsf{E}(V, K), K_1)\ \mathcal{R}_{\tau\ \mathtt{option}}\ \mathsf{D}(\mathsf{E}(V', K'), K_1')$ $(*)$. Let therefore $V \mathcal{R}_\tau V'$, and fix $K \mathcal{R}_{\mathtt{key}[\tau]} K'$. By choosing $K_1 = K$, $(*)$ becomes $\iota(V)\ \mathcal{R}_{\tau\ \mathtt{option}}\ \mathsf{D}(\mathsf{E}(V', K'), K_1')$, which is equivalent to $K' = K_1'$ and $V \mathcal{R}_\tau V'$. Similarly by choosing $K' = K_1'$, we get $K = K_1$ and $V \mathcal{R}_\tau V'$. In other words, as soon as $\mathcal{R}_\tau$ is not empty, $\mathcal{R}_{\mathtt{key}[\tau]}$ must be a *partial bijection* on $[\![\mathtt{key}[\tau]]\!]$, i.e., the graph of a bijection between two subsets of $[\![\mathtt{key}[\tau]]\!]$.

Proposition 2 *Let $\mathcal{R}^0_b$ be given binary relations on $[\![b]\!]$ for every base type b. Let $\mathcal{R}^0_{\mathtt{key}[\tau]}$ be any partial bijection on $[\![\mathtt{key}[\tau]]\!]$ for every type τ. There exists a cryptographic logical relation $(\mathcal{R}_\tau)_{\tau\ \mathrm{type}}$ such that $\mathcal{R}_b = \mathcal{R}^0_b$ for every base type b, and such that $\mathcal{R}_{\mathtt{key}[\tau]} = \mathcal{R}^0_{\mathtt{key}[\tau]}$ for every type τ. We may define $\mathcal{R}_{\mathtt{bits}[\tau]}$, for any type τ, as any relation such that $\mathcal{R}^\bot_{\mathtt{bits}[\tau]} \subseteq \mathcal{R}_{\mathtt{bits}[\tau]} \subseteq \mathcal{R}^\top_{\mathtt{bits}[\tau]}$.*

Proof. Build the logical relation by induction on types. When building $\mathcal{R}_{\mathtt{bits}[\tau]}$, we need to check that $\mathcal{R}^\bot_{\mathtt{bits}[\tau]} \subseteq \mathcal{R}^\top_{\mathtt{bits}[\tau]}$. Let $V \mathcal{R}_\tau V'$, $K \mathcal{R}_{\mathtt{key}[\tau]} K'$, and $K_1 \mathcal{R}_{\mathtt{key}[\tau]} K_1'$ be arbitrary, and let us check $(*)$ (see discussion above). If $K = K_1$ then $K' = K_1'$ since $\mathcal{R}_{\mathtt{key}[\tau]}$ is a partial bijection, so $\mathsf{D}(\mathsf{E}(V, K), K_1) = \iota(V)\ \mathcal{R}_{\tau\ \mathtt{option}}\ \iota(V') = \mathsf{D}(\mathsf{E}(V', K'), K_1')$, hence $(*)$ holds. Similarly, if $K' = K_1'$ then $K = K_1$, so $(*)$ holds again. Finally, if $K \neq K_1$ and $K' \neq K_1'$, $\mathsf{D}(\mathsf{E}(V, K), K_1) = \bot\ \mathcal{R}_{\tau\ \mathtt{option}}\ \bot = \mathsf{D}(\mathsf{E}(V', K'), K_1')$, hence again $(*)$ holds. □

Proposition 2 shows that cryptographic logical relations exist that coincide with given relations on base types. But contrarily to logical relations in the λ-calculus, they are far from being uniquely determined: we have considerable freedom as to the choice of the relations at key and bits types.

To define $\mathcal{R}_{\mathtt{key}[\tau]}$, notably, we may use the intuition that some keys are observable by an intruder, and some others are not. Letting fr_τ be the set of observable keys, define $\mathcal{R}_{\mathtt{key}[\tau]}$ as relating the key K with itself provided $K \in fr_\tau$, and not relating any non-observable key with any key. This is clearly a partial bijection, in fact one that coincides with the identity on the subset fr_τ of $[\![\mathtt{key}[\tau]]\!]$. This is a popular choice: fr_τ is what Abadi and Gordon [2] call a *frame*, up to the fact that frames are defined there as sets of names, not of keys.

To define $\mathcal{R}_{\mathtt{bits}[\tau]}$, we may choose any relation sandwiched between $\mathcal{R}^{\perp}_{\mathtt{bits}[\tau]}$ and $\mathcal{R}^{\top}_{\mathtt{bits}[\tau]}$. Let us make these two more explicit.

For every $V_0, V'_0 \in [\![\mathtt{bits}[\tau]]\!]$, $V_0\ \mathcal{R}^{\perp}_{\mathtt{bits}[\tau]}\ V'_0$ if and only if V_0 is of the form $\mathsf{E}(V, K)$, V'_0 is of the form $\mathsf{E}(V', K')$, $V\ \mathcal{R}_\tau\ V'$ and $K\ \mathcal{R}_{\mathtt{key}[\tau]}\ K'$, i.e., $K = K' \in fr_\tau$. In other words, V_0 and V'_0 are related by $\mathcal{R}^{\perp}_{\mathtt{bits}[\tau]}$ if and only if they are encryptions of related plaintexts by a unique key that the intruder may observe.

On the other hand, $V_0\ \mathcal{R}^{\top}_{\mathtt{bits}[\tau]}\ V'_0$ if and only if $V_0 = \mathsf{E}(V, K)$ and $V'_0 = \mathsf{E}(V', K')$ with either $V\ \mathcal{R}_\tau\ V'$ and $K = K' \in fr_\tau$, or $K, K' \notin fr_\tau$ (whatever V, V').

So, $\mathcal{R}_{\mathtt{bits}[\tau]}$ is completely characterized by the datum of fr_τ, plus a function ψ_τ mapping pairs of keys K, K' in $[\![\mathtt{key}[\tau]]\!] \setminus fr_\tau$ to a binary relation $\psi_\tau(K, K')$ on $[\![\tau]\!]$: if $\mathcal{R}_{\mathtt{bits}[\tau]}$ is given, then let $\psi_\tau(K, K')$ be defined as relating V with V' if and only $\mathsf{E}(V, K)\ \mathcal{R}_{\mathtt{bits}[\tau]}\ \mathsf{E}(V', K')$; on the other hand, given ψ_τ, the relation $\mathcal{R}_{\mathtt{bits}[\tau]}$ that relates $\mathsf{E}(V, K)$ with $\mathsf{E}(V', K')$ if and only if $V\ \mathcal{R}_\tau\ V'$ and $K = K' \in fr_\tau$, or $K, K' \notin fr_\tau$ and $V\ \psi_\tau(K, K')\ V'$, is such that $\mathcal{R}^{\perp}_{\mathtt{bits}[\tau]} \subseteq \mathcal{R}_{\mathtt{bits}[\tau]} \subseteq \mathcal{R}^{\top}_{\mathtt{bits}[\tau]}$.

Given parameters fr and ψ, we then get the following definition of a *unique* cryptographic logical relation by induction on types, so that it coincides with given relations on base types:

Proposition 3 *Let fr_τ be some subset of $[\![\mathtt{key}[\tau]]\!]$, for each type τ, and ψ_τ be any function from $([\![\mathtt{key}[\tau]]\!] \setminus fr_\tau)^2$ to the set $\mathbb{P}([\![\tau]\!] \times [\![\tau]\!])$ of binary relations on $[\![\tau]\!]$.*

For any family $\mathcal{R}^0_b$ of binary relations on $[\![b]\!]$, b a base type, let $(\mathcal{R}^{fr,\psi}_\tau)_{\tau\text{ type}}$ be the family of relations defined by:
- *$\mathcal{R}^{fr,\psi}_b = \mathcal{R}^0_b$ for each base type b;*
- *for every $f, f' \in [\![\tau_1 \to \tau_2]\!]$, $f\ \mathcal{R}^{fr,\psi}_{\tau_1\to\tau_2}\ f'$ if and only if for every $a\ \mathcal{R}^{fr,\psi}_{\tau_1}\ a'$, $f(a)\ \mathcal{R}^{fr,\psi}_{\tau_2}\ f'(a')$;*
- *for every $K, K' \in [\![\mathtt{key}[\tau]]\!]$, $K\ \mathcal{R}^{fr,\psi}_{\mathtt{key}[\tau]}\ K'$ if and only if $K = K' \in fr_\tau$;*
- *for every $V, V' \in [\![\tau]\!]$, for every $K, K' \in [\![\mathtt{key}[\tau]]\!]$, $\mathsf{E}(V, K)\ \mathcal{R}^{fr,\psi}_{\mathtt{bits}[\tau]}\ \mathsf{E}(V', K')$ if and only if $V\ \mathcal{R}^{fr,\psi}_\tau\ V'$ and $K = K' \in fr_\tau$, or $K, K' \notin fr_\tau$ and $V\ \psi_\tau(K, K')\ V'$.*

Then, whatever the choices of fr_τ and ψ_τ, $(\mathcal{R}^{fr,\psi}_\tau)_{\tau\text{ type}}$ is a cryptographic logical relation.

Clearly, Proposition 3 generalizes to the case where fr_τ and ψ_τ are not given *a priori*, but are defined using the relations $\mathcal{R}^{fr,\psi}_{\tau'}$ for (not necessarily strict) subtypes τ' of τ. That is, when not just $\mathcal{R}^{fr,\psi}_\tau$ but also fr_τ and ψ_τ are defined by mutual induction on types.

Ignoring the treatment of fresh names and the fact that their semantics is operational and ours is denotational, Sumii and Pierce [21, Section 6.3] use a unique function φ, instead of a family of functions ψ_τ, in defining their logical relation at bits types. This is inessential. They also define their logical relation on terms, while we define ours on their semantics. While we write $V_0\ \mathcal{R}^{fr,\psi}_\tau\ V'_0$, they use the notation $\varphi \vdash^{\mathrm{val}} V_0 \sim V'_0 : \tau$. They do not use a frame fr_τ. Again ignoring fresh names, they define $\varphi \vdash^{\mathrm{val}} K \sim K' : \mathtt{key}[\tau]$ by $K = K'$ and $(K, K'') \notin \mathrm{dom}\ \varphi$, $(K'', K') \notin \mathrm{dom}\ \varphi$ for any $K'' \in [\![\mathtt{key}[\tau]]\!]$, so that they get the same definition for key types $\mathtt{key}[\tau]$ as in Proposition 3, provided fr_τ is defined as $\{K \in [\![\mathtt{key}[\tau]]\!] \,|\, (K, K'') \notin \mathrm{dom}\ \varphi$ and $(K'', K) \notin \mathrm{dom}\ \varphi$ for any $K'' \in [\![\mathtt{key}[\tau]]\!]\}$. Their definition of the logical relation at bits types (again, ignoring names) is $\varphi \vdash^{\mathrm{val}} \mathsf{E}(V, K) \sim \mathsf{E}(V', K') : \mathtt{bits}[\tau]$ if and only if $(K, K') \notin \mathrm{dom}\ \varphi$ and $V\ \mathcal{R}^{fr,\psi}_\tau\ V'$, or $(K, K') \in \mathrm{dom}\ \varphi$ and $V\ \psi(K, K')\ V'$. This is exactly the definition we had in Proposition 3 if we take $\varphi = \psi$, provided the complement of $\mathrm{dom}\ \varphi$ is exactly fr^2_τ. Conversely, given φ we may define ψ_τ on $(\mathtt{key}[\tau] \setminus fr_\tau)^2$ by: $\psi_\tau(K, K') = \varphi(K, K')$ if $(K, K') \in \mathrm{dom}\ \varphi$, otherwise $\psi_\tau(K, K') = \mathcal{R}^{fr,\psi}_\tau$. (Note that $\mathrm{dom}\ \varphi \subseteq (\mathtt{key}[\tau] \setminus fr_\tau)^2$.) Then our definition of $\mathcal{R}^{fr,\psi}_{\mathtt{bits}[\tau]}$ coincides with their definition—provided fresh names are ignored in their definition, and provided we restrict ourselves to logical relations on key types that are restriction of the equality relation instead of being more general partial bijections.

It is interesting, too, to relate the definition of $\mathcal{R}^{fr,\psi}_\tau$ to selected parts of the notion of framed bisimulation [2]. We have already mentioned that the notion of frame originated in this paper, although frames related names and not keys. Slightly adapting [2] again, call a *theory* (on type $\mathtt{bits}[\tau]$) any *finite* binary relation th_τ on $[\![\mathtt{bits}[\tau]]\!]$. By finite, we mean that it should be finite as a set of pairs of values. A frame-theory pair (fr_τ, th_τ) is *consistent* if and only if th_τ is a partial bijection, and $\mathsf{E}(V, K)\ th_\tau\ \mathsf{E}(V', K')$ implies $K \notin fr_\tau$ and $K' \notin fr_\tau$. Any consistent frame-theory pair determines a ψ_τ function by $V\ \psi_\tau(K, K')\ V'$ if and only if $\mathsf{E}(V, K)\ th_\tau\ \mathsf{E}(V', K')$. It follows that frame-theory pairs, as explained here, are special cases of pairs of a frame fr_τ and a function ψ_τ.

3.5. A More Uniform Toy Cryptographic λ-Calculus, and Prelogical Relations

Reflecting on the developments above, we see that it would be much natural to use, instead of the toy cryptographic λ-calculus, a simply-typed λ-calculus with two constants `enc` and `dec`, with respective semantics given by E and D. While we are at it, it is clear from the way we define $\mathcal{R}^0_{\mathtt{key}[\tau]}$ in Proposition 3 that the type $\mathtt{key}[\tau]$ behaves more like a base type than a type constructed from smaller types. It is therefore relevant to change the algebra of types to something like:

$$\tau ::= b \mid \tau_1 \to \tau_2 \mid \mathtt{bits}[\tau] \mid \mathtt{key} \mid \tau\ \mathtt{option} \mid \ \ldots$$

where b ranges over Σ, Σ now contains a collection of *key types* $\mathtt{key}_1, \ldots, \mathtt{key}_n$ (wlog., we shall use just one, which we write `key`), and the $\tau\ \mathtt{option}$ type is used to give a typing to $\mathtt{dec} : \mathtt{bits}[\tau] \to \mathtt{key} \to \tau\ \mathtt{option}$; `enc` is assumed to have type $\tau \to \mathtt{key} \to \mathtt{bits}[\tau]$. The final ellipsis is meant to indicate that there may be other type formers (products, etc.): we do not wish to be too specific here.

The language we get is just the simply-typed λ-calculus with constants… up to the fact that we need option types $\tau\ \mathtt{option}$. The constants to consider here are at least `dec`, `enc`, $\mathtt{SOME} : \tau \to \tau\ \mathtt{option}$, $\mathtt{NONE} : \tau\ \mathtt{option}$, and $\mathtt{case} : \tau\ \mathtt{option} \to (\tau \to \tau') \to \tau' \to \tau'$. (The `case` constant implements the elimination principle for $\tau\ \mathtt{option}$; we write $\mathtt{case}\ s\ \mathtt{of}\ \mathtt{SOME}\ x \Rightarrow t \mid \mathtt{NONE} \Rightarrow t'$ instead of $\mathtt{case}\ s(\lambda x \cdot t)t'$, and leave the semantics of `case` as an exercise to the reader.)

The fact that the constants `dec`, `enc`, are required to have their denotations, D and E, related to themselves is reminiscent of *prelogical relations* [11]. These can be defined in a variety of ways. Following [11, Definition 3.1, Proposition 3.3], a *prelogical relation* is any family $(\mathcal{R}_\tau)_{\tau \text{ type}}$ of relations (between two values of $[\![\tau]\!]$ in our case) such that:
1. for every $f, f' \in [\![\tau_1 \to \tau_2]\!]$, if $f\ \mathcal{R}_{\tau_1 \to \tau_2}\ f'$ and $a\ \mathcal{R}_{\tau_1}\ a'$ then $f(a)\ \mathcal{R}_{\tau_2}\ f'(a')$;
2. $\mathsf{K}\ \mathcal{R}_{\tau_1 \to \tau_2 \to \tau_1}\ \mathsf{K}$, where K is the function mapping $x \in [\![\tau_1]\!]$, $y \in [\![\tau_2]\!]$ to x;
3. $\mathsf{S}\ \mathcal{R}_{(\tau_1 \to \tau_2 \to \tau_3) \to (\tau_1 \to \tau_2) \to \tau_1 \to \tau_3}\ \mathsf{S}$, where S is the function mapping $x \in [\![\tau_1 \to \tau_2 \to \tau_3]\!]$, $y \in [\![\tau_1 \to \tau_2]\!]$, $z \in [\![\tau_1]\!]$ to $x(z)(y(z))$;
4. and for every constant $a : \tau$, $[\![a]\!]\,\epsilon\ \mathcal{R}_\tau\ [\![a]\!]\,\epsilon$.
where $[\![a]\!]$ denotes $[\![a]\!]\,\rho$ for any environment ρ. Condition 1. is just one half of **(Log)**. The basic lemma for prelogical relations [11, Lemma 4.1] is stronger than for logical relations: prelogical relations are *exactly* those families of relations indexed by types such that the basic lemma holds.

Note that the use of prelogical relations also requires us to relate the semantics of `SOME` with itself, that of `NONE` with itself, and that of `case` with itself.

Then, we may observe that prelogical relations are not just sound for contextual equivalence, they are *complete*, at all types, even higher-order. Recall that a value $a \in [\![\tau]\!]$ is *definable* if and only if there exists a (necessarily closed) term t such that $\vdash t : \tau$ is derivable, and $a = [\![t]\!]\,\epsilon$, where ϵ is the empty environment. Note that it is fair to let Obs be $\{\mathtt{bool}\}$, where `bool` is a base type with two constants $\mathtt{true} : \mathtt{bool}$ and $\mathtt{false} : \mathtt{bool}$, such that $[\![\mathtt{true}]\!]\,\epsilon = \top$, $[\![\mathtt{false}]\!]\,\epsilon = \bot$, and $[\![\mathtt{bool}]\!]$ is $\mathbb{B} = \{\top, \bot\}$; then, every element of $\mathbb{B}$ is definable.

Theorem 4 (Completeness) *Assume that observation types have no junk, in the sense that every value of $[\![o]\!]$ ($o \in \mathsf{Obs}$) is definable.*

Then prelogical relations are complete for contextual equivalence in the λ-calculus, in the strong sense that there is a prelogical relation $(\mathcal{R}_\tau)_{\tau \text{ type}}$ that coincides with equality on observation types $o \in \mathsf{Obs}$ such that, for every closed terms u, u' of type τ, $u \approx_\tau u'$ if and only if $[\![u]\!]\ \mathcal{R}_\tau\ [\![u']\!]$.

Proof. Define $\mathcal{R}_\tau$ on $[\![\tau]\!]$ by $a\ \mathcal{R}_\tau\ a'$ if and only if a and a' are definable and $a \approx_\tau a'$.

First, $\approx_o$ coincides with equality on $[\![o]\!]$, for every $o \in \mathsf{Obs}$. Indeed, if $a = a'$ then clearly $a \approx_o a'$. Conversely, if $a \approx_o a'$ then $[\![\mathcal{C}]\!]\,[x := a] = [\![\mathcal{C}]\!]\,[x := a']$ for every context of observation type, in particular for $\mathcal{C} = x$; so $a = a'$.

Since observation types have no junk, $\mathcal{R}_o$ is also the equality relation on $[\![o]\!]$, for every $o \in \mathsf{Obs}$.

Next, we claim that $(\mathcal{R}_\tau)_{\tau \text{ type}}$ is a prelogical relation. First, check condition **1.** Assume $f\ \mathcal{R}_{\tau_1 \to \tau_2}\ f'$, i.e., f and f' are definable (say by closed terms t and t' respectively) and for every $\mathcal{C}_{fun}$ such that $z : \tau_1 \to \tau_2 \vdash \mathcal{C}_{fun} : o$ is derivable ($o \in \mathsf{Obs}$), $[\![\mathcal{C}_{fun}]\!]\,[z := f] = [\![\mathcal{C}_{fun}]\!]\,[z := f']$. Let $a\ \mathcal{R}_{\tau_1}\ a'$, i.e., a and a' are definable (say by closed terms u and u' respectively) and for every $\mathcal{C}_{arg}$ such that $y : \tau_1 \vdash \mathcal{C}_{arg} : o$ is derivable ($o \in \mathsf{Obs}$), $[\![\mathcal{C}_{arg}]\!]\,[y := a] = [\![\mathcal{C}_{arg}]\!]\,[y := a']$.

We have to show that $f(a)\ \mathcal{R}_{\tau_2}\ f'(a')$. Let $\mathcal{C}$ be any term such that $x : \tau_2 \vdash \mathcal{C} : o$, for any $o \in \mathsf{Obs}$. Then $f(a)$ and $f'(a')$ are definable (by tu and $t'u'$ respectively), and, letting z and z' be fresh variables:

$$\begin{aligned}
& [\![\mathcal{C}]\!]\,[x := f(a)] \\
= \ & [\![\mathcal{C}[x := zu]]\!]\,[z := f] \quad (\text{since } a = [\![u]\!]) \\
= \ & [\![\mathcal{C}[x := zu]]\!]\,[z := f'] \quad (\text{since } f\ \mathcal{R}_{\tau_1 \to \tau_2}\ f') \\
= \ & [\![\mathcal{C}]\!]\,[x := f'(a)] \\
= \ & [\![\mathcal{C}[x := t'z']]\!]\,[z' := a] \quad (\text{since } f' = [\![t']\!]) \\
= \ & [\![\mathcal{C}[x := t'z']]\!]\,[z' := a'] \quad (\text{since } a\ \mathcal{R}_{\tau_1}\ a') \\
= \ & [\![\mathcal{C}]\!]\,[x := f'(a')]
\end{aligned}$$

Now check **4.** For every constant $c : \tau$, $[\![c]\!]\,\epsilon \mathcal{R}_\tau\ [\![c]\!]\,\epsilon$; this is obvious, since clearly $[\![t]\!]\,\epsilon \mathcal{R}_\tau\ [\![t]\!]\,\epsilon$ for every closed term t of type τ. Similarly, $\mathsf{K}\ \mathcal{R}_{\tau_1 \to \tau_2 \to \tau_1}\ \mathsf{K}$ (take $t = \lambda x \cdot \lambda y \cdot x$), and S is related to itself (take $t = \lambda x \cdot \lambda y \cdot \lambda z \cdot xz(yz)$), establishing **2.** and **3.**

Finally, by definition $[\![u]\!]\ \mathcal{R}_\tau\ [\![u]\!]'$ just means that $[\![u]\!] \approx_\tau [\![u']\!]$, i.e., $u \approx_\tau u'$, since $[\![u]\!]$ and $[\![u']\!]$ are obviously definable. $\Box$

The argument before Proposition 3 applies here without further ado: every prelogical relation must be a partial bijection at the `key` type, and conversely, any prelogical relation that is the equality on $fr \subseteq [\![\mathtt{key}]\!]$ at the `key` type satisfies the basic lemma, hence can be used to establish contextual equivalence. Specializing the prelogical relation $(\mathcal{R}_\tau)_{\tau \text{ type}}$ of Theorem 4, we get that $\mathcal{R}_{\mathtt{key}}$ is exactly equality on the set $fr = \{[\![t]\!]\,\epsilon | \vdash t : \mathtt{key}\}$ of definable keys.

Similarly, we may define the binary relation $\psi_\tau(K, K')$, for every $K, K' \in [\![\mathtt{key}]\!] \setminus fr$, (i.e., i.e., for all non-definable keys) by $V \quad \psi_\tau(K, K') \quad V'$ if and only if $\mathsf{E}(V, K)\mathcal{R}_{\mathtt{bits}[\tau]}\mathsf{E}(V', K')$, i.e., if and only if $\mathsf{E}(V, K)$ and $\mathsf{E}(V', K')$ are definable at type $\mathtt{bits}[\tau]$, and $\mathsf{E}(V, K)\mathcal{R}_{\mathtt{bits}[\tau]}\mathsf{E}(V', K')$.

From this, we infer immediately the following combination of the analogue of Proposition 3 (soundness) with Theorem 4 (completeness):

Proposition 5 *Assume that observation types have no junk. There is a prelogical relation $(\mathcal{R}^{fr,\psi}{}_\tau)_{\tau \text{ type}}$, parameterized by fr and ψ, which is:*
- strict *at the* `key` *type: i.e., for every $K, K' \in [\![\mathtt{key}]\!]$, $K \ \mathcal{R}^{fr,\psi}_{\mathtt{key}} \ K'$ if and only if $K = K' \in fr$;*
- strict *at* $\mathtt{bits}[\tau]$ *types: i.e., for every $V, V' \in [\![\tau]\!]$, for every $K, K' \in [\![\mathtt{key}]\!]$, $\mathsf{E}(V, K) \ \mathcal{R}^{fr,\psi}_{\mathtt{bits}[\tau]} \ \mathsf{E}(V', K')$ if and only if $V \ \mathcal{R}^{fr,\psi}_\tau \ V'$ and $K = K' \in fr$, or $K, K' \notin fr$ and $V \ \psi_\tau(K, K') \ V'$;*
- *which coincides with equality on observation types;*
- *and such that, for some fr and ψ, for every closed terms t, t' of type τ, $[\![t]\!]\,\epsilon \approx_\tau [\![t']\!]\,\epsilon$ if and only if $[\![t]\!]\,\epsilon \ \mathcal{R}^{fr,\psi}_\tau \ [\![t']\!]\,\epsilon$.*

The idea of being *strict* at some type τ is, in all cases, that the (pre)logical relation at type τ should be defined uniquely as a function of the (pre)logical relations at all immediate subterms of τ. Analogously, we say that a prelogical relation $(\mathcal{R}^{fr,\psi}_\tau)_{\tau \text{ type}}$ is *strict* at $\tau_1 \to \tau_2$ if and only if $f \ \mathcal{R}^{fr,\psi}_{\tau_1 \to \tau_2} \ f'$ is equivalent to: for all $a \ \mathcal{R}^{fr,\psi}_{\tau_1} \ a'$, $f(a) \ \mathcal{R}^{fr,\psi}_{\tau_2} \ f'(a')$—i.e., if it is a (cryptographic) logical relation in the sense of Section 3.3.

Following the same idea, we say that $(\mathcal{R}_\tau)_{\tau \text{ type}}$ is *strict* at τ `option` if and only if $a \ \mathcal{R}^{fr,\psi}_{\tau \ \mathtt{option}} \ a'$ is equivalent to: either $a = a' = \bot$, or $a = \iota(a_1)$, $a' = \iota(a'_1)$, and $a_1 \ \mathcal{R}^{fr,\psi}_\tau \ a'_1$. The prelogical relation of Proposition 5 is then strict at option types, too, provided there is a closed term of type τ or $[\![\tau]\!]$ has no junk. Indeed, if $a \ \mathcal{R}^{fr,\psi}_{\tau \ \mathtt{option}} \ a'$ then, first, we cannot have $a = \bot$ but $a' = \iota(a'_1)$ (or conversely), since otherwise the context $\mathcal{C} = \mathtt{case} \ x \ \mathtt{of} \ \mathtt{SOME} \ z \Rightarrow \mathtt{true} \mid \mathtt{NONE} \Rightarrow \mathtt{false}$ would differentiate a from a'. So $a = a' = \bot$, or $a = \iota(a_1)$, $a' = \iota(a'_1)$; in the latter case, since $a \approx_\tau a'$, for every context $\mathcal{C}$ such that $x : \tau \ \mathtt{option} \vdash \mathcal{C} : o$ ($o \in \mathsf{Obs}$), $[\![\mathcal{C}]\!]\,[x := a] = [\![\mathcal{C}]\!]\,[x := a']$. If $[\![o]\!]$ is empty, then vacuously $[\![\mathcal{C}']\!]\,[x := a_1] = [\![\mathcal{C}']\!]\,[x := a'_1]$ for every $\mathcal{C}'$ such that $x : \tau \vdash \mathcal{C}' : o$. Otherwise, since o has no junk, let t_0 be any closed term of type o. $[\![\mathcal{C}]\!]\,[x := a] = [\![\mathcal{C}]\!]\,[x := a']$ holds in particular for $\mathcal{C} = \mathtt{case} \ x \ \mathtt{of} \ \mathtt{SOME} \ z \Rightarrow \mathcal{C}'[x := z] \mid \mathtt{NONE} \Rightarrow t_0$, where $\mathcal{C}'$ is arbitrary such that $x : \tau \vdash \mathcal{C}' : o$: by an easy computation, we obtain $[\![\mathcal{C}']\!]\,[x := a_1] = [\![\mathcal{C}']\!]\,[x := a'_1]$, and since $\mathcal{C}'$ is arbitrary $a_1 \approx_\tau a'_1$. We now claim that a_1 is definable. If there is a closed term t_1 of type τ, then a_1 is definable as $\mathtt{case} \ t \ \mathtt{of} \ \mathtt{SOME} \ z \Rightarrow z \mid \mathtt{NONE} \Rightarrow t_1$ (where $a = [\![t]\!]\,\epsilon$). Otherwise, if τ has no junk, then $[\![\tau]\!] =$, so $[\![\tau \ \mathtt{option}]\!] = \{\bot\}$, so a cannot be of the form $\iota(a_1)$. In any case a_1 is definable. Similarly, a'_1 is definable. So $a_1 \ \mathcal{R}^{fr,\psi}_\tau \ a'_1$.

While the point in prelogical relations in [11] is mainly of being not strict at arrow types, the point of Section 3 is to argue that it is meaningful either not to be strict either at $\mathtt{bits}[\tau]$ types, as in Section 3.3 (in the sense that $\mathcal{R}_{\mathtt{bits}[\tau]}$ was not determined uniquely from $\mathcal{R}_\tau$), or equivalently to be strict at $\mathtt{bits}[\tau]$, given parameters fr and τ. We believe that just saying that we do not require strictness at $\mathtt{bits}[\tau]$, thus omitting the fr and τ parameters, leads to some simplification.

3.6. Completeness on First-Order Terms

The prelogical relation $(\mathcal{R}^{fr,\psi}_\tau)_{\tau \text{ type}}$ of Proposition 5 is strict at bits, key, and most option types, but not at arrow types. If it were strict at all types, then, given fr, ψ, there would be a canonical way to decide whether $[\![t]\!]\,\epsilon \ \mathcal{R}^{fr,\psi}_\tau \ [\![t']\!]\,\epsilon$: apply the definition of $\mathcal{R}^{fr,\psi}_\tau$, recursing on types τ. We show that we may require the prelogical relation to be strict at arrow types, provided we restrict ourselves to first-order terms.

Instead of defining first-order types syntactically, we prefer a more semantical definition. Say that τ is *junkless* if and only if $[\![\tau]\!]$ contains no junk. This includes `bool`, $\mathtt{bits}[\tau]$ and τ `option` types provided τ is junkless; products and sums of junkless types if we decide to include products and sums; but not arrow types in general, and not the `key` type (which may contain junk, namely all those keys not in the frame fr of Proposition 5). *Zero-order* types are those types τ^0 which are junkless and not arrow types, and such that any subexpression of τ^0 is also zero-order. These include at least all types generated by the grammar:

$$\tau^0 \quad ::= \quad \mathtt{bool} \mid \mathtt{bits}[\tau^0] \mid \tau^0 \ \mathtt{option}$$

First-order types are defined inductively as `key`, or $\tau_1^0 \to \tau_2$ where τ_1 is zero-order and τ_2 is first-order. A closed term is *first-order* if and only if its type is first-order.

Proposition 6 *For every closed first-order terms t, t' of type τ_0, $[\![t]\!]\,\epsilon \approx_{\tau_0} [\![t']\!]\,\epsilon$ if and only if $[\![t]\!]\,\epsilon \ \mathcal{R}'_{\tau,}{}^{fr,\psi} \ [\![t']\!]\,\epsilon$,* *for some* logical *relation* $(\mathcal{R}'^{fr,\psi}_{\tau})_{\tau \text{ type}}$ *that is strict at arrow,* `key`, `bits`$[\tau]$ *and* τ `option` *(*τ *junkless) types and coincides with equality at observation types.*

Proof. The if direction is clear. Conversely, let $(\mathcal{R}'^{fr,\psi}_{\tau})_{\tau \text{ type}}$ be the unique logical relation (i.e., strict at all types) that coincides with $(\mathcal{R}^{fr,\psi}_{\tau})_{\tau \text{ type}}$ at base types. These relations coincide in particular at zero-order types, because $(\mathcal{R}^{fr,\psi}_{\tau})_{\tau \text{ type}}$ is strict at all zero-order types (see Proposition 6). They also coincide at the `key` type, because `key` is a base type. Finally, we claim that they coincide at all first-order types on definable values. This is by induction on first-order types $\tau_1^0 \rightarrow \tau_2$.

On the one hand, if $[\![t]\!]\,\epsilon \;\; \mathcal{R}'^{fr,\psi}_{\tau_1^0\rightarrow\tau_2} \;\; [\![t']\!]\,\epsilon$ then $[\![t]\!]\,\epsilon \approx_{\tau_1^0\rightarrow\tau_2} [\![t']\!]\,\epsilon$ since $\mathcal{R}'$ is sound for contextual equivalence (this is the basic lemma), so $[\![t]\!]\,\epsilon \; \mathcal{R}_{\tau_1^0\rightarrow\tau_2} \; [\![t']\!]\,\epsilon$ since $\mathcal{R}$ is complete on definable values.

On the other hand, if $[\![t]\!]\,\epsilon \mathcal{R}^{fr,\psi}_{\tau_1^0\rightarrow\tau_2} \; [\![t']\!]\,\epsilon$, let a and a' be arbitrary values such that $a \; \mathcal{R}'^{fr,\psi}_{\tau_1^0} \; a'$. Since τ_1^0 is zero-order, hence junkless, a and a' are definable. By induction hypothesis, it follows that $a \; \mathcal{R}^{fr,\psi}_{\tau_1^0} \; a'$. Since $\mathcal{R}$ is prelogical, $[\![t]\!]\,\epsilon(a) \; \mathcal{R}^{fr,\psi}_{\tau_2} \; [\![t']\!]\,\epsilon(a')$, so by induction hypothesis again, $[\![t]\!]\,\epsilon(a) \; \mathcal{R}'^{fr,\psi}_{\tau_2} \; [\![t']\!]\,\epsilon(a')$. Since a and a' are arbitrary such that $a \; \mathcal{R}'^{fr,\psi}_{\tau_1^0} \; a'$, by **(Log)** $[\![t]\!]\,\epsilon \; \mathcal{R}'^{fr,\psi}_{\tau_1^0\rightarrow\tau_2} \; [\![t']\!]\,\epsilon$. □

3.7. Extensions

We have already said that using a type `bits`$[\tau]$ for ciphertexts, distinct from the type τ of plaintexts, could lead to miss type confusion attacks: by typing, the intruder cannot submit an object of type τ where one of type `bits`$[\tau]$ is expected.

The obvious fix is to change the algebra of types so that both plaintexts and ciphertexts are of some universal message type `msg`, and require `enc` to be of type `msg` $\rightarrow$ `key` $\rightarrow$ `msg`, `dec` to be of type `msg` $\rightarrow$ `key` $\rightarrow$ `msg option`. It is in fact easier to *define* `msg` as an inductive type with constructors `b` : `bits`[`msg`] $\rightarrow$ `msg` (coercing ciphertexts to messages) and, say, `p` : `msg` $\rightarrow$ `msg` $\rightarrow$ `msg`, `nil` : `msg`, `raw` : `int` $\rightarrow$ `msg`, `k` : `key` $\rightarrow$ `msg` (with obvious meanings).

This simply amounts to considering that the λ-calculus of Section 3.5 contains the constants above, plus a recursor

$$\begin{aligned}\texttt{msg\_rec} \;\; : \;\; & ((\texttt{key} \rightarrow \tau \rightarrow \tau) \rightarrow \tau) \\ & \rightarrow (\tau \rightarrow \tau \rightarrow \tau) \rightarrow \tau \\ & \rightarrow \tau \rightarrow (\texttt{int} \rightarrow \tau) \rightarrow (\texttt{key} \rightarrow \tau) \\ & \rightarrow \texttt{msg} \rightarrow \tau\end{aligned}$$

so that, letting F be `msg_rec` $b\ p\ n\ i\ q$, the equations of Figure 3 obtain.

And the theory of Section 3.5 goes through. There are several variants here. We may split keys into symmetric, public and private keys to handle asymmetric encryption. We may also allow general terms of type `msg` as encryption keys: this is useful in actual protocols, as in the SSL handshake [22], where keys are computed from other messages. We let the interested reader do the necessary adaptations in each case.

Another extension is the handling of algebraic laws. E.g., in RSA encoding [19], encryption E is implemented as modular exponentiation, which obeys various associativity, commutativity and distributivity laws. To give an example that remains in the framework of symmetric encryption, DES [8] obeys the property that `not` $\mathsf{E}(V, K) = \mathsf{E}(\texttt{not}\ V, \texttt{not}\ K)$, where `not` is bitwise logical not. We may also try to be even more realistic and consider that `msg` is a finite type (e.g., 1024-bit integers). In this case, there is no injective pairing function from $[\![\tau]\!] \times [\![\tau]\!]$ to $[\![\tau]\!]$. This breaks Proposition 5, which relies on the fact that $\mathsf{E}(V, K)$ really behaves as the pair (V, K). Nonetheless, prelogical relations are still sound and complete for this semantics (or for any semantics whatsoever), i.e., Theorem 4 still holds. This opens the way to notions of (pre)logical relations that are sound and complete for more complex models of encryption.

4. Adding Name Creation, and Lax Logical Relations

No decent calculus for cryptographic protocols can dispense with fresh name creation. This is most easily done by following Stark [20], who defined a categorical semantics for a calculus with fresh name creation based on Moggi's monadic λ-calculus [15]. We just take his language, adding all needed constants as in Section 3.5.

4.1. The Moggi-Stark Calculus

The *Moggi-Stark calculus* is obtained by adding a new type former T (the *monad*), to the types of the λ-calculus of Section 3.5, so that $T\tau$ is a type as soon as τ is:

$$\tau \quad ::= \quad b \mid \tau_1 \rightarrow \tau_2 \mid \texttt{bits}[\tau] \mid \texttt{key} | \tau\ \texttt{option} \mid T\tau \mid \ldots$$

(We continue to leave the definition of our calculi open, as shown with the ellipsis ..., to facilitate the addition of new types and constants, if needed.) Following Stark, we also require the existence of a new base type $\boldsymbol{\nu} \in \Sigma$ of *names*. (This will take the place of the type `key` of keys, which we shall equate with names.) The λ-calculus of Section 3.5 is enriched with constructs `val` t and `let` $x \Leftarrow t$ `in` u (not to be confused with the `let` construct of Section 3.1), with typing rules as given in Figure 4, and two

$$\begin{array}{rcl}
[\![F(\mathtt{b}\ m)]\!]\,\rho & = & [\![b(\lambda k\cdot\lambda x\cdot\mathtt{case}\ \mathsf{D}\ m\ k\ \mathtt{of}\ \mathtt{SOME}\ m'\Rightarrow Fm' \mid \mathtt{NONE}\Rightarrow x)]\!]\,\rho \\
[\![F(\mathtt{p}\ m_1 m_2)]\!]\,\rho & = & [\![p(Fm_1)(Fm_2)]\!]\,\rho \\
[\![F(\mathtt{nil})]\!]\,\rho & = & n \qquad [\![F(\mathtt{raw}\ z)]\!]\,\rho\ =\ i(z) \qquad [\![F(\mathtt{k}\ k)]\!]\,\rho\ =\ q(k)
\end{array}$$

Figure 3. Recursor equations

$$\frac{\Gamma\vdash t:\tau}{\Gamma\vdash \mathtt{val}\ t:T\tau}\,(\mathtt{val}) \qquad \frac{\Gamma\vdash t:T\tau \quad \Gamma,x:\tau\vdash u:T\tau'}{\Gamma\vdash \mathtt{let}\ x\Leftarrow t\ \mathtt{in}\ u:T\tau'}\,(\mathtt{let})$$

Figure 4. Additional typing rules for the monadic λ-calculus

constants $\mathtt{new}:T\boldsymbol{\nu}$ (fresh name creation) and $\doteq: \boldsymbol{\nu}\to\boldsymbol{\nu}\to\mathtt{bool}$ (equality of names).

In Stark's semantics (notations are ours here), given any finite set s (of names), $[\![t]\!]\,s\rho$ is the value of t in environment ρ *assuming that all previously created names are in* s. This allows one to describe the creation of fresh names as returning any name outside s. This is most elegantly described by letting the values of terms be taken in the presheaf category $\boldsymbol{Set}^{\mathcal{I}}$ [20], where $\mathcal{I}$ is the category whose objects are finite sets and whose morphisms $s\xrightarrow{i}s'$ are injections. Given any type τ, $[\![\tau]\!]\,s$ is intuitively the set of all values of type τ in a world where all created names are in s. Since $[\![\tau]\!]$ is a functor, for every injection $s\xrightarrow{i}s'$ there is a conversion $[\![\tau]\!]\,i$ that sends any value a of $[\![\tau]\!]\,s$ to one in $[\![\tau]\!]\,s'$, intuitively by renaming the names in a using i. By extension, if Γ is any context $x_1:\tau_1,\ldots,x_n:\tau_n$, let $[\![\Gamma]\!]$ be $[\![\tau_1]\!]\times\ldots\times[\![\tau_n]\!]$, using the products in $\boldsymbol{Set}^{\mathcal{I}}$—i.e., products at each world s. Then, as usual in categorical semantics [12], given any term t such that $\Gamma\vdash t:\tau$ is derivable, $[\![t]\!]$ is a morphism from $[\![\Gamma]\!]$ to $[\![\tau]\!]$. This means that $[\![t]\!]$ is a natural transformation from $[\![\Gamma]\!]$ to $[\![\tau]\!]$, in particular that, for every finite set s, $[\![t]\!]\,s$ maps any Γ, *s-environment* ρ (a map sending each x_i such that $x_i:\tau_i$ is in Γ to some element of $[\![\tau_i]\!]\,s$) to some value $[\![t]\!]\,s\rho$ in $[\![\tau]\!]\,s$; and all this is natural in s, i.e., compatible with renaming of names.

Interestingly, $T\tau$, the type of computations that result in a value of type τ, possibly creating fresh names during the course of computation, is defined semantically by $[\![T\tau]\!]=\boldsymbol{T}\,[\![\tau]\!]$, where $(\boldsymbol{T},\boldsymbol{\eta},\boldsymbol{\mu},\boldsymbol{t})$ is the strong monad defined as:
• $\boldsymbol{T}A=\mathrm{colim}_{s'}\,A(\_+s'):\mathcal{I}\to\boldsymbol{Set}$. On objects, this is given by $\boldsymbol{T}As=\mathrm{colim}_{s'}\,A(s+s')$, i.e., $\boldsymbol{T}As$ is the set of all equivalence classes of pairs (s',a) with s' a finite set and $a\in A(s+s')$, modulo the smallest equivalence relation $\equiv$ such that $(s',a)\equiv(s'',A(\mathrm{id}_s+j)a)$ for every morphism $s'\xrightarrow{j}s''$ in $\mathcal{I}$. Intuitively, given a set of *names* s, elements of $\boldsymbol{T}As$ are formal expressions $(\nu s')a$ where all names in s' are bound and every name free in a is in $s+s'$—modulo the fact that $(\nu s',s'')a\equiv(\nu s')a$ for any additional set of new names s'' not free in a. We shall in fact write $(\nu s')a$ the equivalence class of (s',a), to aid intuition. (This was written $[s',a]$ in [10, 23].)
On morphisms $s_1\xrightarrow{i}s_2$, $\boldsymbol{T}Ai$ maps $(\nu s')a$ to $(\nu s')A(i+\mathrm{id}_{s'})a$.
• For any morphism $f:A\to B$ in $\boldsymbol{Set}^{\mathcal{I}}$, $\boldsymbol{T}fs:\boldsymbol{T}As\to\boldsymbol{T}Bs$ is defined by $\boldsymbol{T}fs((\nu s')a)=(\nu s')(f(s+s')a)$. This is compatible with $\equiv$ because f is natural.
• $\boldsymbol{\eta}_As:As\to\boldsymbol{T}As$ is defined by $\boldsymbol{\eta}_Asa=(\nu\emptyset)a$.
• $\boldsymbol{\mu}_As:\boldsymbol{T}^2As\to\boldsymbol{T}As$ is defined by $\boldsymbol{\mu}_As((\nu s')(\nu s'')a)=(\nu s'+s'')a$.
• $\boldsymbol{t}_{A,B}s:As\times\boldsymbol{T}Bs\to\boldsymbol{T}(A\times B)s$ is defined by $\boldsymbol{t}_{A,B}s(a,(\nu s')b)=(\nu s')(A(\mathsf{inl}ss')a,b))$ where $\mathsf{inl}ss':s\to s+s'$ is the canonical injection.
The semantics of `let` and `val` is standard [15]. Making it explicit on this particular monad, we obtain:

$$\begin{array}{rcl}
[\![\mathtt{val}\ t]\!]\,s\rho & = & (\nu\emptyset)\,[\![t]\!]\,s\rho \\
[\![\mathtt{let}\ x\Leftarrow t\ \mathtt{in}\ u]\!]\,s\rho & = & (\nu s'+s'')b
\end{array}$$

where $[\![t]\!]\,s\rho=(\nu s')a$, we assume that $\Gamma\vdash t:T\tau$ and $\Gamma,x:\tau\vdash u:T\tau'$, and where $[\![u]\!]\,(s+s')(([\![\Gamma]\!]\,(\mathsf{inl}ss')\rho)[x:=a])=(\nu s'')b$. (Concretely, if Γ is $x_1:\tau_1,\ldots,x_n:\tau_n$, $\rho=[x_1:=a_1,\ldots,x_n:=a_n]$ where $a_i\in[\![\tau_i]\!]\,s$ for every i, then $[\![\Gamma]\!]\,(\mathsf{inl}ss')\rho$ is $[x_1:=[\![\tau_1]\!]\,(\mathsf{inl}ss')a_1,\ldots,x_n:=[\![\tau_n]\!]\,(\mathsf{inl}ss')a_n]$.)

The semantics of base types $b\in\Sigma$, except $\boldsymbol{\nu}$, is given by constant functors: $[\![b]\!]\,s$ is a fixed set, independent of s; e.g., $[\![\mathtt{bool}]\!]\,s=\mathbb{B}$. The semantics of $\boldsymbol{\nu}$ is $[\![\boldsymbol{\nu}]\!]\,s=s$, $[\![\boldsymbol{\nu}]\!]\,i=i$; i.e., the names that exist at s are just the elements of s. Since $\boldsymbol{Set}^{\mathcal{I}}$ is a presheaf category, it is a topos, hence cartesian-closed [12]. This provides a semantics for λ-abstraction, variables and applications.

Finally, the semantics of $\mathtt{new}:T\boldsymbol{\nu}$ is given by $[\![\mathtt{new}]\!]\,s\rho=(\nu\{n\})n$, where n is any element not in s, and $[\![\doteq]\!]$ is defined as the only morphism in $\boldsymbol{Set}^{\mathcal{I}}$ such that

$[\![\dot{=}\, xy]\!]\, s[x := a, y := b]$ is `true` if $a = b$, and `false` otherwise.

4.2. Lax Logical Relations for Monads

Given that terms now take values in some category ($\boldsymbol{Set}^{\mathcal{I}}$), not in $\boldsymbol{Set}$ as in Section 3, the proper generalization of prelogical relations is given by *lax logical relations* [17]. We introduce this notion as gently as possible.

Let Σ be the set of base types, seen as a trivial category. The simply-typed λ-calculus gives rise to the *free CCC* $\boldsymbol{\lambda}(\Sigma)$ over Σ as follows: the objects of $\boldsymbol{\lambda}(\Sigma)$ are typing contexts Γ, a morphism from Γ to $\Delta = y_1 : \tau_1, \ldots, y_n : \tau_n$ is a substitution $[y_1 := t_1, \ldots, y_n := t_n]$, where $\Gamma \vdash t_i : \tau_i$ ($1 \leq i \leq n$), modulo $\beta\eta$-conversion. (In particular, Γ-environments are exactly morphisms from the terminal object, the empty context ϵ, to Γ.) Composition is substitution. Being the free CCC means that, for any CCC $\boldsymbol{C}$, for any functor $[\![\_]\!]_0$ from Σ to $\boldsymbol{C}$ (i.e., for any function $[\![\_]\!]_0$ mapping each base type in Σ to some object in $\boldsymbol{C}$), there is a unique representation $[\![\_]\!]_1$ of CCCs from $\lambda(\Sigma)$ to $\boldsymbol{C}$ such that the following diagram commutes:

$$\begin{tikzcd}
\Sigma \arrow[r, "\subseteq"] \arrow[dr, "{[\![\_]\!]_0}"'] & \boldsymbol{\lambda}(\Sigma) \arrow[d, "{[\![\_]\!]_1}"] \\
& \boldsymbol{C}
\end{tikzcd} \tag{1}$$

A representation of CCCs is any functor that preserves products and exponentials. When $\boldsymbol{C}$ is $\boldsymbol{Set}$, this describes all at once all the constructions $[\![\tau]\!]_1$ (denotation of types τ) and $[\![t]\!]_1$ (denotations of typed λ-terms t) as used in Section 3.

Let $\mathrm{Subscone}^{\mathbb{C}}_{\boldsymbol{C}}$ be the *subscone* category, defined as follows. Assume $\mathbb{C}$ is another CCC, such that $\mathbb{C}$ has pullbacks. Let $|\_|$ be a functor from $\boldsymbol{C}$ to $\mathbb{C}$ that preserves finite products. Then $\mathrm{Subscone}^{\mathbb{C}}_{\boldsymbol{C}}$ is the category whose objects are triples $\langle S, m, A\rangle$, where m is a mono $S \hookrightarrow |A|$ in $\mathbb{C}$, and whose morphisms from $\langle S, m, A\rangle$ to $\langle S', m', A'\rangle$ are pairs of morphisms $\langle u, v\rangle$ (u in $\mathbb{C}$, from S to S', and v in $\boldsymbol{C}$, from A to A'), making the obvious square commute. Noting that $\mathrm{Subscone}^{\mathbb{C}}_{\boldsymbol{C}}$ is again a CCC (Mitchell and Scedrov [14] make this remark when $\mathbb{C}$ is $\boldsymbol{Set}$, and $|\_|$ is the global section functor $\boldsymbol{C}(1, \_)$), the following purely diagrammatic argument obtains. Assume we are given a functor from Σ to $\mathrm{Subscone}^{\mathbb{C}}_{\boldsymbol{C}}$, i.e., a collection $\mathcal{R}_o$ of objects in $\mathrm{Subscone}^{\mathbb{C}}_{\boldsymbol{C}}$, one for each base type o. Then there is a unique representation $\mathcal{R}$ of CCCs from $\boldsymbol{\lambda}(\Sigma)$ such that the following diagram commutes:

$$\begin{tikzcd}
\Sigma \arrow[r, "\subseteq"] \arrow[d, "(\mathcal{R}_o)_{o\in\Sigma}"'] & \boldsymbol{\lambda}(\Sigma) \arrow[dl, "\mathcal{R}"] \\
\mathrm{Subscone}^{\mathbb{C}}_{\boldsymbol{C}} &
\end{tikzcd} \tag{2}$$

Now the crux of the argument is the following: the forgetful functor $U : \mathrm{Subscone}^{\mathbb{C}}_{\boldsymbol{C}} \to \boldsymbol{C}$ mapping the object $\langle S, m, A\rangle$ to A and the morphism $\langle u, v\rangle$ to v is also a representation of CCCs. It follows that $U \circ \mathcal{R}$ is a representation of CCCs again, from $\boldsymbol{\lambda}(\Sigma)$ to $\boldsymbol{C}$. If $U \circ (\mathcal{R}_o)_{o\in\Sigma} = [\![\_]\!]_0$, then by the uniqueness property of $[\![\_]\!]_1$, we must have $U \circ \mathcal{R} = [\![\_]\!]_1$. As observed in [14], and extended to CCCs in [3], when $\mathbb{C} = \boldsymbol{Set}$, $\boldsymbol{C}$ is the product of two CCCs $\boldsymbol{A}$ and $\boldsymbol{B}$, and $|\_|$ is the functor $\boldsymbol{A}(1, \_) \times \boldsymbol{B}(1, \_)$, $(\mathcal{R}(\tau))_{\tau \text{ type}}$ behaves like a logical relation. It is really a logical relation, as we have defined it earlier, when both $\boldsymbol{A}$ and $\boldsymbol{B}$ are $\boldsymbol{Set}$. (In this case, an object $\mathcal{R}(\tau)$ is of the form $\langle S, m, A\rangle$, where S, up to isomorphism, is just a subset of A, and A is the cartesian product of the set of values of type τ with itself.) In case $\boldsymbol{A}$ and $\boldsymbol{B}$ are the same presheaf category $\boldsymbol{Set}^{\mathcal{I}}$, $(R(\tau))_{\tau \text{ type}}$ is a Kripke logical relation with base category $\mathcal{I}$.

While the object part of functor $\mathcal{R}$, $(\mathcal{R}(\tau))_{\tau \text{ type}}$, yields logical relations (or extensions), the morphism part maps each morphism in $\boldsymbol{\lambda}(\Sigma)$, namely a typed term t modulo $\beta\eta$, of type τ, to a morphism in the subscone, i.e., a pair $\langle u, v\rangle$. The fact that $U \circ \mathcal{R} = [\![\_]\!]_1$ states that v is just the pair of the semantics of t in $\boldsymbol{A}$ and the semantics of t in $\boldsymbol{B}$, and the fact that $\langle u, v\rangle$ is a morphism (saying that a certain square commutes) states that these two semantics are related by $\mathcal{R}(\tau)$: this establishes the basic lemma.

The important property to make $\mathcal{R}$ satisfy the basic lemma is just the identity $U \circ \mathcal{R} = [\![\_]\!]_1$, i.e., the following commuting diagram:

$$\begin{tikzcd}
& \boldsymbol{\lambda}(\Sigma) \arrow[dl, "\mathcal{R}"'] \arrow[d, "{[\![\_]\!]_1}"] \\
\mathrm{Subscone}^{\mathbb{C}}_{\boldsymbol{C}} \arrow[r, "U"'] & \boldsymbol{C}
\end{tikzcd} \tag{3}$$

Logical relations are the case where $\mathcal{R}$ is a representation of CCCs, in which case, as we have seen, this diagram necessarily commutes. *Lax* logical relations are product preserving functors $\mathcal{R}$ such that Diagram (3) commutes [17, Section 6]. The difference if that, with lax logical relations, we do not require $\mathcal{R}$ to be representations of CCCs, just product preserving functors. We say that $\mathcal{R}$ is *strict at arrow types* if and only if $\mathcal{R}$ preserves exponentials, too.

Defining lax logical relations for Moggi's monadic metalanguage follows the same pattern. (We might use the formalization of algebraic theories of [17, Section 7], but it seems clearer to describe the construction explicitly.) The monadic λ-calculus gives rise to the *free let-CCC* $\boldsymbol{Comp}(\Sigma)$ over Σ, where a let-CCC is a CCC with a strong monad. We then get Diagram (1) again, only with $\boldsymbol{\lambda}(\Sigma)$ replaced by $\boldsymbol{Comp}(\Sigma)$, $\boldsymbol{C}$ is a let-CCC, and $[\![\_]\!]_1$ is a representation of let-CCCs, i.e., a functor that preserves products, exponentials, and the monad (functor, unit, multiplication, strength).

While we then needed $\mathrm{Subscone}^{\mathbb{C}}_{\boldsymbol{C}}$ to be a CCC to establish Diagram (2), we now need it to be a let-CCC. This can be established by following the general construction of [10]. In general, we need to consider not the subscone considered above, rather the subscone relative to a mono factorization system $(\mathcal{E}, \mathcal{M})$. In a presheaf category such as $\boldsymbol{Set}^{\mathcal{I}}$, the only reasonable mono factorization system has $\mathcal{E}$ be the class of all epis (i.e., morphisms e such that es is surjective at all worlds s), and $\mathcal{M}$ be the class of all monos (i.e., morphisms m such that ms is injective at all worlds s). The construction of [10] also requires $|\_|$ to be part of a strong monad morphim $(|\_|, \sigma)$ from the strong monad $(\boldsymbol{T}, \boldsymbol{\eta}, \boldsymbol{\mu}, \boldsymbol{t})$ to some strong monad $(\mathbb{T}, \mathbb{n}, \mathbb{p}, \mathbb{t})$ on $\mathbb{C}$ (σ was called a strong distributivity law in [10], but this led to some confusion with the notion of distributivity laws for monads [4]; while the latter are rare, strong monad morphisms abound). The construction of [10] then provides one with a strong monad on $\mathrm{Subscone}^{\mathbb{C}}_{\boldsymbol{C}}$, such that U is a representation of let-CCCs.

4.3. Contextual Equivalence

Defining contextual equivalence in a calculus with names is a bit tricky. First, we have to consider contexts $\mathcal{C}$ of type To ($o \in \mathsf{Obs}$), not of type o. Intuitively, contexts should be allowed to do some computations; were they of type o, they could only return values. In particular, note that contexts $\mathcal{C}$ such that $x : T\tau \vdash \mathcal{C} : o$, meant to observe computations at type τ (i.e., values of type $T\tau$), cannot observe anything. This is because the $(\mathtt{let})$ typing rule (Figure 4) only allows one to use computations to build other computations, never values.

Another tricky aspect is that we cannot take contexts $\mathcal{C}$ that only depend on one variable $x : \tau$ as before. We must indeed assume that $\mathcal{C}$ can also depend on an arbitrary set of public names. Given names $n_1, \ldots, n_k$, the only way $\mathcal{C}$ can be made to depend on them is to assume that $\mathcal{C}$ has k free variables $z_1, \ldots, z_k$ of type $\boldsymbol{\nu}$, which are mapped to $n_1, \ldots, n_k$. (It is more standard [16, 1] to consider expressions built on separate sets of variables and names, thus introducing the semantic notion of names in the syntax. It is more natural here to consider that there are variables z_i mapped, in a one-to-one way, to names n_i.) Let s_1 be any set of names containing $n_1, \ldots, n_k$, let w_1 be $\{z_1, \ldots, z_k\}$, and $w_1 \overset{i_1}{\rightarrow} s_1$ the injection mapping each n_i to z_i, $1 \leq i \leq k$. Write $\overline{w_1 := i_1(w_1)}$ for $z_1 := n_1, \ldots, z_k := n_k$, and $\overline{w_1 : \boldsymbol{\nu}}$ for $z_1 : \boldsymbol{\nu}, \ldots, z_k : \boldsymbol{\nu}$. We shall then consider contexts $\mathcal{C}$ such that $\overline{w_1 : \boldsymbol{\nu}}, x : \tau \vdash \mathcal{C} : To$ is derivable, and evaluate $[\![\mathcal{C}]\!]\, s_1[x := a, \overline{w_1 := i_1(w_1)}]$ and compare it with $[\![\mathcal{C}]\!]\, s_1[x := a', \overline{w_1 := i_1(w_1)}]$ to decide whether a and a' are contextually equivalent. This represents the fact that $\mathcal{C}$ is evaluated in a world where all names in s_1 have been created, and where $\mathcal{C}$ has access to all (public) names in $i(w_1)$.

This definition is not yet correct, as this would require a and a' to be in $[\![\tau]\!]\, s_1$, but they are in $[\![\tau]\!]\, s$ for some possibly different set s of names created during the evaluation of a and a'. This is repaired by considering some coercion $[\![\tau]\!]\, k_1$, where $s \overset{k_1}{\rightarrow} s_1$ is any injection.

To sum up, say that $a, a' \in [\![\tau]\!]\, s$ are *contextually equivalent at* s, and write $a \approx^s_\tau a'$, if and only if, for every finite set of variables w_1, for every injections $w_1 \overset{i_1}{\rightarrow} s_1$ and $s \overset{k_1}{\rightarrow} s_1$, for every term $\mathcal{C}$ such that $\overline{w_1 : \boldsymbol{\nu}}, x : \tau \vdash \mathcal{C} : To$ is derivable ($o \in \mathsf{Obs}$), $[\![\mathcal{C}]\!]\, s_1[x := [\![\tau]\!]\, k_1(a), \overline{w_1 := i_1(w_1)}] = [\![\mathcal{C}]\!]\, s_1[x := [\![\tau]\!]\, k_1(a'), \overline{w_1 := i_1(w_1)}]$.

The notion we use here is inspired by [16, Definition 4], although it may not look so at first sight. We may simplify it a bit by noting that we lose no generality in considering that $\mathcal{C}$ has access to *all* names in s_1. Without loss of generality, we equate w_1 with s_1, and notice that $a \approx^s_\tau a'$ if and only if, for every injection $s \overset{k_1}{\rightarrow} s_1$, for every term $\mathcal{C}$ such that $\overline{s_1 : \boldsymbol{\nu}}, x : \tau \vdash \mathcal{C} : To$ is derivable ($o \in \mathsf{Obs}$), $[\![\mathcal{C}]\!]\, s_1[x := [\![\tau]\!]\, k_1(a), \overline{s_1 := s_1}] = [\![\mathcal{C}]\!]\, s_1[x := [\![\tau]\!]\, k_1(a'), \overline{s_1 := s_1}]$. (Remember we see the *variables* in s_1 as denoting the *names* in s_1 here, equating names with variables.)

The use of injections between finite sets leads us naturally to switch from $\boldsymbol{Set}^{\mathcal{I}}$ to the category $\boldsymbol{Set}^{\mathcal{I}^{\rightarrow}}$, where $\mathcal{I}^{\rightarrow}$, the *arrow category* of $\mathcal{I}$, has as objects all morphisms $w \overset{i}{\rightarrow} s$ in $\mathcal{I}$, and as morphisms from $w \overset{i}{\rightarrow} s$ to $w' \overset{i'}{\rightarrow} s'$ all pairs (j, k) of morphisms such that the following diagram commutes:

$$\begin{tikzcd}
w \arrow[r, "i"] \arrow[d, "j"'] & s \arrow[d, "k"] \\
w' \arrow[r, "i'"'] & s'
\end{tikzcd} \qquad (4)$$

This is in accordance with [23], where it is noticed that $\boldsymbol{Set}^{\mathcal{I}^{\rightarrow}}$ is the right category to define a Kripke logical relation that coincides with Pitts and Stark's on first-order types.

We shall consider here the equivalent category where w is restricted to be a finite set of *variables* (and continue to call this category $\mathcal{I}^{\rightarrow}$). Objects $w \overset{i}{\rightarrow} s$ are then sets w of variables denoting those public names in s, together with an injective denotation function i.

So we shall work with lax logical relations in the subscone category $\mathrm{Subscone}^{\mathbb{C}}_{\boldsymbol{C}}$, where $\boldsymbol{C} = \boldsymbol{Set}^{\mathcal{I}} \times \boldsymbol{Set}^{\mathcal{I}}$, $\mathbb{C}$ is the presheaf category $\boldsymbol{Set}^{\mathcal{I}^{\rightarrow}}$, and $|\_| : \boldsymbol{C} \rightarrow \mathbb{C}$ is the composite of the binary product functor $\times : \boldsymbol{Set}^{\mathcal{I}} \times \boldsymbol{Set}^{\mathcal{I}} \rightarrow \boldsymbol{Set}^{\mathcal{I}}$ with the functor $\boldsymbol{Set}^{\mathrm{u}} : \boldsymbol{Set}^{\mathcal{I}} \rightarrow \boldsymbol{Set}^{\mathcal{I}^{\rightarrow}}$. Here

$\mathfrak{u} : \mathcal{I}^{\rightarrow} \rightarrow \mathcal{I}$ is the obvious forgetful functor that maps $w \xrightarrow{i} s$ to s.

Say that a value $a \in [\![\tau]\!]\, s$ is *definable at* $w \xrightarrow{i} s$ if and only if there is a term t such that $\overline{w : \boldsymbol{\nu}} \vdash t : \tau$ is derivable and $a = [\![t]\!]\, s[\overline{w := i(w)}]$.

The main point in our completeness argument to come is, imitating Theorem 4, that there is a lax logical relation built by considering the trace of $\approx^s_\tau$ on definable elements. More specifically:

Definition 1 *Let $w \xrightarrow{i} s$ be any object of $\mathcal{I}^{\rightarrow}$. The values $a, a' \in [\![\tau]\!]\, s$ are said to be* contextually equivalent *at $w \xrightarrow{i} s$, written $a \approx^{w \xrightarrow{i} s}_\tau a'$, if and only if, for every morphism (j_1, k_1) from $w \xrightarrow{i} s$ to any object $w_1 \xrightarrow{i_1} s_1$ in $\mathcal{I}^{\rightarrow}$, for every term $\mathcal{C}$ such that $\overline{w_1 : \boldsymbol{\nu}}, x : \tau \vdash \mathcal{C} : To$ ($o \in \mathsf{Obs}$) is derivable, $[\![\mathcal{C}]\!]\, s_1[x := [\![\tau]\!]\, k_1(a), \overline{w_1 := i_1(w_1)}] = [\![\mathcal{C}]\!]\, s_1[x := [\![\tau]\!]\, k_1(a'), \overline{w_1 := i_1(w_1)}]$.*

Define the relation $\mathcal{R}^{w \xrightarrow{i} s}_\tau$ by: $a\ \mathcal{R}^{w \xrightarrow{i} s}_\tau\ a'$ if and only if a and a' are definable at $w \xrightarrow{i} s$ and $a \approx^{w \xrightarrow{i} s}_\tau a'$.

In particular, $a \approx^s_\tau a'$ if and only if $a \approx^{\emptyset \rightarrow s}_\tau a'$, where $\emptyset \rightarrow s$ denotes the unique injection from $\emptyset$ to s.

Note that for every value $a \in [\![\tau]\!]\, s$ definable at $w \xrightarrow{i} s$, $[\![\tau]\!]\, k(a)$ is also definable at $w' \xrightarrow{i'} s'$, whenever there is a morphism (j, k) from the former to the latter. Indeed, let $a = [\![t]\!]\, s[\overline{w := i(w)}]$. Then

$$\begin{array}{rcl}
[\![\tau]\!]\, k(a) & = & [\![\tau]\!]\, k([\![t]\!]\, s[\overline{w := i(w)}]) \\
& = & [\![t]\!]\, s'[\overline{w := [\![\boldsymbol{\nu}]\!]\, k(i(w))}] \\
& & \text{(by naturality of } [\![t]\!]\text{)} \\
& = & [\![t]\!]\, s'[\overline{w := k(i(w))}] \\
& & \text{(because } [\![\boldsymbol{\nu}]\!]\, k = k \text{ by definition)} \\
& = & [\![t]\!]\, s'[\overline{w := i'(j(w))}] \\
& = & [\![t'j]\!]\, s'[\overline{w := i'(j(w))}] \\
& & \text{(where } t' = tj^{-1}\text{,} \\
& & \text{seeing } j \text{ and } j^{-1} \text{ as substitutions)} \\
& = & [\![t']\!]\, s'([\![j]\!]\, s'[\overline{w := i'(j(w))}]) \\
& & \text{(because } [\![\_]\!] \text{ is functorial)} \\
& = & [\![t']\!]\, s'[\overline{j(w) := i'(j(w))}] \\
& = & [\![t']\!]\, s'[\overline{w' := i'(w')}]
\end{array}$$

In particular, every value $a \in [\![\tau]\!]\, s$ definable at s, i.e., definable at $\emptyset \rightarrow s$, is definable at every $w \xrightarrow{i} s$.

Theorem 7 *Lax logical relations are complete for contextual equivalence in the Moggi-Stark calculus, in the strong sense that there is a lax logical relation $\mathcal{R}$ such that, for every terms u, u' such that $\overline{w : \boldsymbol{\nu}} \vdash u : \tau$ and $\overline{w : \boldsymbol{\nu}} \vdash u' : \tau$ are derivable, $[\![u]\!]\, s[\overline{w := i(w)}] \approx^{w \xrightarrow{i} s}_\tau [\![u']\!]\, s[\overline{w := i(w)}]$ if and only if $[\![u]\!]\, s[\overline{w := i(w)}]\ \mathcal{R}^{w \xrightarrow{i} s}_\tau\ [\![u']\!]\, s[\overline{w := i(w)}]$.*

Proof. Define $\mathcal{R}^{w \xrightarrow{i} s}_\tau$ as in Definition 1.

We first need to show that $\mathcal{R}_\tau$, mapping $w \xrightarrow{i} s$ to $\mathcal{R}^{w \xrightarrow{i} s}_\tau$, defines an object of $\boldsymbol{Set}^{\mathcal{I}^{\rightarrow}}$, i.e., a functor from $\mathcal{I}^{\rightarrow}$ to $\boldsymbol{Set}$. The action on morphisms (j, k) is given by our requirement that $U \circ \mathcal{R} = [\![\_]\!]_1$, where $\mathcal{R}$ maps τ to $\mathcal{R}_\tau$, and $[\![\tau]\!]_1\, (w \xrightarrow{i} s) = [\![\tau]\!]\, s \times [\![\tau]\!]\, s$ and $[\![t]\!]_1\, (w \xrightarrow{i} s) = [\![t]\!]\, s \times [\![t]\!]\, s$. Expand the equation $U \circ \mathcal{R} = [\![\_]\!]_1$: $\mathcal{R}_\tau(j, k)$ must map $(a, a') \in [\![\tau]\!]\, s \times [\![\tau]\!]\, s$ to $([\![\tau]\!]\, k(a), [\![\tau]\!]\, k(a')) \in [\![\tau]\!]\, s' \times [\![\tau]\!]\, s'$. To check that $\mathcal{R}_\tau$ is a functor, we must check that if $a\ \mathcal{R}^{w \xrightarrow{i} s}_\tau\ a'$, then $[\![\tau]\!]\, k(a)\ \mathcal{R}^{w' \xrightarrow{i'} s'}_\tau\ [\![\tau]\!]\, k(a')$, for every commutative square (4):

- First, a and a' are definable at $w \xrightarrow{i} s$; by the remark before the theorem, $[\![\tau]\!]\, k(a)$ and $[\![\tau]\!]\, k(a')$ are definable at $w' \xrightarrow{i'} s'$.
- Second, $a \approx^{w \xrightarrow{i} s}_\tau a'$. We must show that this implies $[\![\tau]\!]\, k(a) \approx^{w' \xrightarrow{i'} s'}_\tau [\![\tau]\!]\, k(a')$ ($\star$). (We shall use the ($\star$) property later again.) Let (j_1, k_1) be any morphism from $w' \xrightarrow{i'} s'$ to some object $w_1 \xrightarrow{i_1} s_1$, and $\mathcal{C}$ be any term such that $\overline{w_1 : \boldsymbol{\nu}}, x : \tau \vdash \mathcal{C} : To$ is derivable ($o \in \mathsf{Obs}$).

$$\begin{array}{rl}
& [\![\mathcal{C}]\!]\, s_1[x := [\![\tau]\!]\, k_1([\![\tau]\!]\, k(a)), \overline{w_1 := i_1(w_1)}] \\
= & [\![\mathcal{C}]\!]\, s_1[x := [\![\tau]\!]\, (k_1 \circ k)(a), \overline{w_1 := i_1(w_1)}] \\
= & [\![\mathcal{C}]\!]\, s_1[x := [\![\tau]\!]\, (k_1 \circ k)(a'), \overline{w_1 := i_1(w_1)}] \\
& \text{(since } a \approx^{w \xrightarrow{i} s}_\tau a'\text{, using } \mathcal{C} \\
& \text{and the morphism } (j_1 \circ j, k_1 \circ k)) \\
= & [\![\mathcal{C}]\!]\, s_1[x := [\![\tau]\!]\, k_1([\![\tau]\!]\, k(a')), \overline{w_1 := i_1(w_1)}]
\end{array}$$

Next, we need to show that $\mathcal{R}_\tau$ is the object part of a product-preserving functor $\mathcal{R}$ from $\boldsymbol{Comp}(\Sigma)$ to $\mathrm{Subscone}^{\mathbb{C}}_{\mathcal{C}}$ such that $U \circ \mathcal{R} = [\![\_]\!]_1$. This means showing that, for every typing context $\Gamma = x_1 : \tau_1, \ldots, x_n : \tau_n$, for every type τ such that $\Gamma \vdash t : \tau$ is derivable, for every object $w \xrightarrow{i} s$ of $\boldsymbol{Set}^{\mathcal{I}^{\rightarrow}}$, if $a_i\ \mathcal{R}^{w \xrightarrow{i} s}_{\tau_1}\ a'_i$ for every i, $1 \leq i \leq n$, then $[\![t]\!]\, s[x_1 := a_1, \ldots, x_n := a_n]\ \mathcal{R}^{w \xrightarrow{i} s}_\tau\ [\![t]\!]\, s[x_1 := a'_1, \ldots, x_n := a'_n]$. Since a_i and a'_i are definable at $w \xrightarrow{i} s$, write $a_i = [\![t_i]\!]\, s[\overline{w := i(w)}]$ for some t_i such that $\overline{w : \boldsymbol{\nu}} \vdash t_i : \tau_i$, and similarly $a'_i = [\![t'_i]\!]\, s[\overline{w := i(w)}]$. Then, first, $[\![t]\!]\, s[x_1 := a_1, \ldots, x_n := a_n]$ is definable at $w \xrightarrow{i} s$, by the term $t[x_1 := t_1, \ldots, x_n := t_n]$, and similarly for $[\![t]\!]\, s[x_1 := a'_1, \ldots, x_n := a'_n]$. Second, for every morphism (j_1, k_1) from $w \xrightarrow{i} s$ to $w_1 \xrightarrow{i_1} s_1$, for every $\mathcal{C}$ such that $\overline{w_1 : \boldsymbol{\nu}}, x : \tau \vdash \mathcal{C} : To$ is derivable, the derivation of Figure 5 obtains; here we notice that, since a_i and a'_i are definable at $w \xrightarrow{i} s$ by t_i and t'_i, respectively, then $[\![\tau_i]\!]\, k_1(a_i)$ and $[\![\tau_i]\!]\, k_1(a'_i)$ are definable at $w_1 \xrightarrow{i_1} s_1$ by $t_i j_1^{-1}$, resp. $t'_i j_1^{-1}$. So $[\![t]\!]\, s[x_1 := a_1, \ldots, x_n := a_n]\ \mathcal{R}^{w \xrightarrow{i} s}_\tau\ [\![t]\!]\, s[x_1 := a'_1, \ldots, x_n := a'_n]$.

$$
\begin{array}{rl}
& [\![\mathcal{C}]\!]\, s_1[x := [\![\tau]\!]\, k_1([\![t]\!]\, s[x_1 := a_1, \ldots, x_n := a_n]), \overline{w_1 := i_1(w_1)}] \\
= & [\![\mathcal{C}]\!]\, s_1[x := [\![t]\!]\, s_1[x_1 := [\![\tau_1]\!]\, k_1(a_1), \ldots, x_n := [\![\tau_n]\!]\, k_1(a_n)]), \overline{w_1 := i_1(w_1)}] \\
& \text{(since } [\![t]\!] \text{ is a natural transformation)} \\
= & [\![\mathcal{C}[x := t]]\!]\, s_1[x_1 := [\![\tau_1]\!]\, k_1(a_1), \ldots, x_n := [\![\tau_n]\!]\, k_1(a_n), \overline{w_1 := i_1(w_1)}] \\
= & \left[\!\!\left[\mathcal{C}[x := t][x_1 := t_1 j_1^{-1}, \ldots, x_{n-1} := t_{n-1} j_1^{-1}]\right]\!\!\right] s_1[x_n := [\![\tau_n]\!]\, k_1(a_n), \overline{w_1 := i_1(w_1)}] \\
= & \left[\!\!\left[\mathcal{C}[x := t][x_1 := t_1 j_1^{-1}, \ldots, x_{n-1} := t_{n-1} j_1^{-1}]\right]\!\!\right] s_1[x_n := [\![\tau_n]\!]\, k_1(a'_n), \overline{w_1 := i_1(w_1)}] \\
& \text{(since } [\![\tau_n]\!]\, k_1(a_n) \approx_{\tau_n}^{w_1 \xrightarrow{i_1} s_1} [\![\tau_n]\!]\, k_1(a'_n)\text{, because } a_n \approx_{\tau_n}^{w \xrightarrow{i} s} a'_n\text{, using } (\star)) \\
= & \left[\!\!\left[\mathcal{C}[x := t][x_1 := t_1 j_1^{-1}, \ldots, x_{n-2} := t_{n-2} j_1^{-1}, x_n := t'_n j_1^{-1}]\right]\!\!\right] s_1 \\
& \qquad [x_{n-1} := [\![\tau_{n-1}]\!]\, k_1(a_{n-1}), \overline{w_1 := i_1(w_1)}] \\
= & \left[\!\!\left[\mathcal{C}[x := t][x_1 := t_1 j_1^{-1}, \ldots, x_{n-2} := t_{n-2} j_1^{-1}, x_n := t'_n j_1^{-1}]\right]\!\!\right] s_1 \\
& \qquad [x_{n-1} := [\![\tau_{n-1}]\!]\, k_1(a'_{n-1}), \overline{w_1 := i_1(w_1)}] \\
& \text{(since } [\![\tau_{n-1}]\!]\, k_1(a_{n-1}) \approx_{\tau_{n-1}}^{w_1 \xrightarrow{i_1} s_1} [\![\tau_{n-1}]\!]\, k_1(a'_{n-1})\text{, because } a_{n-1} \approx_{\tau_{n-1}}^{w \xrightarrow{i} s} a'_{n-1}\text{, using } (\star)) \\
= & \ldots \\
= & \left[\!\!\left[\mathcal{C}[x := t][x_1 := t_1 j_1^{-1}, \ldots, x_{m-1} := t_{m-1} j_1^{-1}, x_{m+1} := t'_{m+1} j_1^{-1}, \ldots, x_n := t'_n j_1^{-1}]\right]\!\!\right] s_1 \\
& \qquad [x_m := [\![\tau_m]\!]\, k_1(a_m), \overline{w_1 := i_1(w_1)}] \\
= & \left[\!\!\left[\mathcal{C}[x := t][x_1 := t_1 j_1^{-1}, \ldots, x_{m-1} := t_{m-1} j_1^{-1}, x_{m+1} := t'_{m+1} j_1^{-1}, \ldots, x_n := t'_n j_1^{-1}]\right]\!\!\right] s_1 \\
& \qquad [x_m := [\![\tau_m]\!]\, k_1(a'_m), \overline{w_1 := i_1(w_1)}] \\
& \text{(since } [\![\tau_m]\!]\, k_1(a_m) \approx_{\tau_m}^{w_1 \xrightarrow{i_1} s_1} [\![\tau_m]\!]\, k_1(a'_m)\text{, because } a_m \approx_{\tau_m}^{w \xrightarrow{i} s} a'_m\text{, using } (\star)) \\
= & \ldots \\
= & \left[\!\!\left[\mathcal{C}[x := t][x_2 := t'_2 j_1^{-1}, \ldots, x_n := t'_n j_1^{-1}]\right]\!\!\right] s_1[x_1 := [\![\tau_1]\!]\, k_1(a'_1), \overline{w_1 := i_1(w_1)}] \\
= & [\![\mathcal{C}[x := t]]\!]\, s_1[x_1 := [\![\tau_1]\!]\, k_1(a'_1), \ldots, x_n := [\![\tau_n]\!]\, k_1(a'_n), \overline{w_1 := i_1(w_1)}] \\
= & [\![\mathcal{C}]\!]\, s_1[x := [\![t]\!]\, s_1[x_1 := [\![\tau_1]\!]\, k_1(a'_1), \ldots, x_n := [\![\tau_n]\!]\, k_1(a'_n)]), \overline{w_1 := i_1(w_1)}] \\
= & [\![\mathcal{C}]\!]\, s_1[x := [\![\tau]\!]\, k_1([\![t]\!]\, s[x_1 := a'_1, \ldots, x_n := a'_n]), \overline{w_1 := i_1(w_1)}]
\end{array}
$$

Figure 5. Showing that $[\![t]\!]\, s[x_1 := a_1, \ldots, x_n := a_n] \approx_{\tau}^{w \xrightarrow{i} s} [\![t]\!]\, s[x_1 := a'_1, \ldots, x_n := a'_n]$

$\mathcal{R}$ is a lax logical relation since $U \circ \mathcal{R} = [\![\_]\!]_1$ by construction. $\square$

The (non-lax) logical relation of [23] is defined on $\boldsymbol{\nu}$ by: $n \; \mathcal{R}_{\boldsymbol{\nu}}^{w \xrightarrow{i} s} \; n'$ if and only if $n = n'$ and $n \in w$ ($n = n' \in w$, for short). This is exactly what the lax logical relation of Definition 1 is defined as on the $\boldsymbol{\nu}$ type:

Lemma 8 *Let $\mathcal{R}_\tau^{w \xrightarrow{i} s}$ be the logical relation of Definition 1. Then $n \; \mathcal{R}_{\boldsymbol{\nu}}^{w \xrightarrow{i} s} \; n'$ if and only if $n = n' \in i(w)$.*

Proof. We first claim that $(\star_{\boldsymbol{\nu}})$: the only values n in $[\![\boldsymbol{\nu}]\!] \, s = s$ that are definable at $w \xrightarrow{i} s$ are the names in $i(w)$. One first observes, by applying any morphism j from s to s that is the identity on $i(w)$, that the only possible exceptions n to $(\star_{\boldsymbol{\nu}})$ must be fixpoints of j: letting $n = [\![t]\!] \, s[\overline{w := i(w)}]$, $j(n) = [\![\boldsymbol{\nu}]\!] \, j(n) = [\![\boldsymbol{\nu}]\!] \, j([\![t]\!] \, s[\overline{w := i(w)}]) = [\![t]\!] \, s[\overline{w := j(i(w))}]$ (since $[\![t]\!]$ is natural) $= [\![t]\!] \, s[\overline{w := i(w)}]$ (since j is the identity on $i(w)$) $= n$. Since j is arbitrary such that it restricts to the identity on $i(w)$, $(\star_{\boldsymbol{\nu}})$ can only fail when s consists of $i(w)$ plus just the one extra name n. Let then s' be s plus another name n'. There is an obvious morphism (j, k) from $w \xrightarrow{i} s$ to $w \xrightarrow{i} s'$, and we have seen that in this case $[\![\boldsymbol{\nu}]\!] \, k(n) = n$ is again definable at $w \xrightarrow{i} s'$. But this is impossible, since s' contains *two* names outside of $i(w)$.

If $n \; \mathcal{R}_{\boldsymbol{\nu}}^{w \xrightarrow{i} s} \; n'$, then both n and n' are definable at $w \xrightarrow{i} s$, so by $(\star_{\boldsymbol{\nu}})$, $n = i(z)$ for some $z \in w$, and $n' = i(z')$ for some $z' \in w$. Since $n \approx_{\boldsymbol{\nu}}^{w \xrightarrow{i} s} n'$, taking as context $\mathcal{C}$ the term $\texttt{val}(z \doteq x)$ (so that $\overline{w : \boldsymbol{\nu}}, x : \boldsymbol{\nu} \vdash \mathcal{C} : T\texttt{bool}$ is derivable), we obtain $n = n'$.

Conversely, if $n = n' \in i(w)$, i.e., there is a variable $z \in w$ such that $n = n' = i(z)$, then clearly $n \; \mathcal{R}_{\boldsymbol{\nu}}^{w \xrightarrow{i} s} \; n'$. $\square$

To finish this section, we observe:

Lemma 9 *Assume that observation types have no junk, in the sense that every value of $[\![o]\!] \, s$ ($o \in \mathsf{Obs}$) is definable at s, for every s, equivalently at every $w \xrightarrow{i} s$.*

Then $\mathcal{R}_o^{w \xrightarrow{i} s}$ is equality on $[\![o]\!] \, s$, and $\mathcal{R}_{To}^{w \xrightarrow{i} s}$ is equality on $[\![To]\!] \, s$ for any observation type o.

Proof. Clearly $\mathcal{R}_o^{w \xrightarrow{i} s}$ contains equality. Conversely, let $a, a' \in [\![o]\!] \, s$ such that $a \; \mathcal{R}_o^{w \xrightarrow{i} s} a'$. Take (j, k) the identity functor from $w \xrightarrow{i} s$ to itself, $\mathcal{C}$ the context $\texttt{val}(c \doteq x)$ (so that $\overline{w : \boldsymbol{\nu}}, x : o \vdash \mathcal{C} : T\texttt{bool}$ is derivable), where c is any term such that $\overline{w : \boldsymbol{\nu}} \vdash c : o$ is derivable, and expand the definition of $\approx_o^{w \xrightarrow{i} s}$: $a = [\![c]\!] \, s[\overline{w := i(w)}]$ if and only if $a' = [\![c]\!] \, s[\overline{w := i(w)}]$. Since o contains no junk, and c is arbitrary, $a = a'$.

The argument is similar for $\mathcal{R}_{To}^{w \xrightarrow{i} s}$, taking $\texttt{let } z \Leftarrow c \texttt{ in val } (z \doteq x)$ for $\mathcal{C}$ instead. We just have to prove that To has no junk. Remember that every observation type is a base type, and $[\![o]\!]$ is a constant functor for any base type o except $\boldsymbol{\nu}$. So the elements of $[\![To]\!] \, s$ are of the form $(\nu s')b = (\nu\emptyset)b$, where $b \in [\![o]\!] \, (s + s') = [\![o]\!] \, s$. Given any element $(\nu\emptyset)b$ of $[\![To]\!] \, s$, since o has no junk, we may write b as the value of some term c, hence $(\nu\emptyset)b$ is the value of $\texttt{val } c$. $\square$

We almost forgot to prove soundness! It is easy to see that any lax logical relation that coincides with equality on types To ($o \in \mathsf{Obs}$) are sound for contextual equivalence. Indeed, by the basic lemma $U \circ \mathcal{R} = [\![\_]\!]_1$, whenever $a \; \mathcal{R}_\tau^{w \xrightarrow{i} s} \; a'$, then for any $\mathcal{C}$ such that $\overline{w_1 : \boldsymbol{\nu}}, x : \tau \vdash \mathcal{C} : To$ ($o \in \mathsf{Obs}$) is derivable, for any morphism (j_1, k_1) from $w \xrightarrow{i} s$ to $w_1 \xrightarrow{i_1} s_1$, $[\![\mathcal{C}]\!] \, s_1[\overline{w_1 := i_1(w_1)}, x := [\![\tau]\!] \, k_1(a)] \; \mathcal{R}_{To}^{w_1 \xrightarrow{i_1} s_1} \; [\![\mathcal{C}]\!] \, s_1[\overline{w_1 := i_1(w_1)}, x := [\![\tau]\!] \, k_1(a')]$; so $a \approx_\tau^{w \xrightarrow{i} s} a'$.

4.4. Mixing Fresh Name Creation and Encryption

Let us get down to earth. What do we need now to get lax logical relations that are sound and complete for contextual equivalence when both fresh name creation and cryptographic primitives are involved? The answer is: just lax logical relations on $\boldsymbol{Set}^{\mathcal{I}^{\to}}$, as used in Section 4.3… making sure that they relate each constant to itself. We have indeed been careful in being sure that our calculi were open, i.e. they can be extended to arbitrarily many new types and constants. In particular, (this is as in Section 3.3 and 3.5,) a lax logical relation on $\boldsymbol{Set}^{\mathcal{I}^{\to}}$ is sound for observational equivalence in the presence of cryptographic primitives if and only if each of the constants `enc`, `dec`, `SOME` , `NONE`, `case` is related to itself.

Then Theorem 7 shows that lax logical relations are complete for the Moggi-Stark calculus, which uses a name creation monad. We have in fact proved more, again because we have been particularly keen on leaving the set of types and constants open: whatever new constants and types you allow, lax logical relations will remain complete. The only requirement is that the new constructs can be given a semantics in $\boldsymbol{Set}^{\mathcal{I}}$. In particular, taking `enc`, `dec`, `SOME` , `NONE`, `case` as new constants, we automatically get sound and complete lax logical relations for name creation *and* cryptographic primitives. As noticed in Section 3.7, this can even be used for encryption primitives obeying algebraic laws, or even other encryption primitives, at no cost.

Acknowledgments

We would like to thank Michel Bidoit for having directed them to the notion of prelogical relations in the first place.

References

[1] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Proc. 4th ACM Conference on Computer and Communications Security (CCS)*, pages 36–47, 1997.

[2] M. Abadi and A. D. Gordon. A bisimulation method for cryptographic protocols. *Nordic Journal of Computing*, 5(4):267–303, 1998.

[3] M. Alimohamed. A characterization of lambda definability in categorical models of implicit polymorphism. *Theoretical Computer Science*, 146(1–2):5–23, July 1995.

[4] J. Beck. Distributive laws. In *Seminar on Triples and Categorical Homology Theory*, volume 80 of *Lecture Notes in Mathematics*, pages 119–140. Springer, 1969.

[5] M. Boreale, R. de Nicola, and R. Pugliese. Proof techniques for cryptographic processes. In *Proceedings of the Symposium on Logics in Computer Science (LICS)*, pages 157–166. IEEE Computer Society Press, 1999.

[6] J. Borgström and U. Nestmann. On bisimulations for the spi calculus. In *Proc. 9th Int. Conf. Algebraic Methodology And Software Technology (AMAST)*, volume 2422 of *Lecture Notes in Computer Science*, pages 287–303. Springer, Sept. 2002.

[7] H. Comon and V. Shmatikov. Is it possible to decide whether a cryptographic protocol is secure or not? *J. of Telecommunications and Information Technology*, 4:3–13, 2002.

[8] The data encryption standard. FIPS PUB 46.

[9] D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, IT-29(2):198–208, Mar. 1983.

[10] J. Goubault-Larrecq, S. Lasota, and D. Nowak. Logical relations for monadic types. In *Proc. 16th Conference of the European Association for Computer Science Logic (CSL)*, volume 2471 of *Lecture Notes in Computer Science*. Springer, Sept. 2002.

[11] F. Honsell and D. Sannella. Pre-logical relations. In *Proc. 13rd Int. Workshop Computer Science Logic (CSL)*, volume 1683 of *Lecture Notes in Computer Science*, pages 546–561, 1999.

[12] J. Lambek and P. J. Scott. *Introduction to Higher Order Categorical Logic*, volume 7 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, 1986.

[13] J. C. Mitchell. *Foundations for Programming Languages*. MIT Press, 1985.

[14] J. C. Mitchell and A. Scedrov. Notes on sconing and relators. In *6th Int. Workshop on Computer Science Logic (CSL)*, volume 702 of *Lecture Notes in Computer Science*, pages 352–378. Springer, 1993.

[15] E. Moggi. Notions of computation and monads. *Information and Computation*, 93:55–92, 1991.

[16] A. Pitts and I. Stark. Observable properties of higher order functions that dynamically create local names, or: What's *new*? In *Proc. Int. Conf. Mathematical Foundations of Computer Science (MFCS)*, volume 711 of *Lecture Notes in Computer Science*, pages 122–141. Springer, 1993.

[17] G. D. Plotkin, J. Power, D. Sannella, and R. D. Tennent. Lax logical relations. In *Proc. 27th International Conference on Automata, Languages and Programming (ICALP)*, volume 1853 of *Lecture Notes in Computer Science*, pages 85–102. Springer, 2000.

[18] J. Power and E. Robinson. Logical relations, data abstraction, and structured fibrations. In *Proc. 2nd ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming (PPDP)*, pages 15–23. ACM Press, 2000.

[19] R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[20] I. Stark. Categorical models for local names. *Lisp and Symbolic Computation*, 9(1):77–107, 1996.

[21] E. Sumii and B. C. Pierce. Logical relations for encryption. In *14th IEEE Computer Security Foundations Workshop (CSFW-14)*, pages 82–96. IEEE Computer Society Press, 2001.

[22] S. A. Thomas. *SSL & TLS Essentials: Securing the Web*. Wiley, 2000. ISBN 0471383546.

[23] Y. Zhang and D. Nowak. Logical relations for dynamic name creation. In *Proc. 17th Int. Workshop Computer Science Logic (CSL) and 8th Kurt Gödel Coll. (KGL)*, volume 2803 of *Lecture Notes in Computer Science*, pages 575–588. Springer, Aug. 2003.