To appear in *Proc. 18th Int. Workshop Computer Science Logic (CSL 2004), Karpacz, Karkonoski National Park, Poland, Sep. 2004 Lecture Notes in Computer Science. Springer, 2004.*

# Complete Lax Logical Relations for Cryptographic Lambda-Calculi $^{\star}$

Jean Goubault-Larrecq[1] Sławomir Lasota[2⋆⋆] David Nowak[1] Yu Zhang[1⋆⋆⋆]

[1] LSV/CNRS & INRIA Futurs & ENS Cachan, Cachan, France
{goubault,nowak,zhang}@lsv.ens-cachan.fr
[2] Institute of Informatics, Warsaw University, 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.
**Keywords:** Logical relations, Monads, Cryptographic lambda-calculus, Subscone

## 1 Introduction

There are nowadays many existing models for cryptographic protocol verification. The most well-known are perhaps the Dolev-Yao model (after [7], see [6] for a survey) and the spi-calculus of [1]. A lesser known model was introduced by Sumii and Pierce [18], 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 $\lambda$-calculus: see [12, Chapter 8], for numerous examples. Sumii and Pierce [18] in particular define three logical relations

$^{\star}$ 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".

$^{\star\star}$ Partially supported by the KBN grant 7 T11C 002 21 and by the Research Training Network *Games*. Part of this work was performed during the author's stay at LSV.

$^{\star\star\star}$ PhD student under an MENRT grant on ACI Cryptologie funding, École Doctorale Sciences Pratiques (Cachan).

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 [10] and more generally to lax logical relations [16]; 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. Proofs omitted in the sequel are to be found in the full version of this paper, available as a technical report [9].

**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 Section 4, 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 5, using Moggi's elegant computational $\lambda$-calculus framework with Stark's name creation monad.

## 2 Related Work

Logical relations have often been used to prove various properties of typed lambda calculi. 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, 18], a datum $M$ of type $\tau$ is *secret* in some term $t(M)$ of type $\tau'$ 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 $\tau'$. We are using $\lambda$-calculus notions here, following [18], but the idea of using contextual equivalence to define security properties was pioneered by Abadi and Gordon [1], where both secrecy and authentication are investigated.

We shall define precisely what we mean by contextual equivalence in a calculus without names (Section 3.2), then with names (Section 5.3). Both notions are standard, the latter being inspired by [15], only adapted to Moggi's computational $\lambda$-calculus [14]. In [15] 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 [12], where *models* are related, not *values* in the same model as here.

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 $\lambda$-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 [10], which we believe is both simpler and more general than Sumii and Pierce's proposal [18]; 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 [18] is inspired in this respect by Pitts and Stark [15], who proposed a $\lambda$-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.

In [8], Goubault-Larrecq, Lasota and Nowak define a Kripke logical relation for the dynamic name creation monad, which is extended by Zhang and Nowak in [19] so that it coincides with Pitts and Stark's operational logical relation up to first-order types. We continue this work here, relying on the elegance of Moggi's computational $\lambda$-calculus [14] to describe side effects, and in particular name creation, using Stark's insights [17].

Further comparisons will be made in the course of this paper, especially with bisimulations for spi-calculus [1, 4, 5]. This continues the observations pioneered in [8], where notions of logical relations for various monads were shown to be proper extensions of known notions of bisimulations. The precise relation with hedged and framed bisimulation [5] remains to be stated precisely.

## 3 Deconstructing Sumii and Pierce's approach

The starting point of this paper was the realization that the rather complex family of logical relations proposed by Sumii and Pierce [18] 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.

### 3.1 The Toy Cryptographic $\lambda$-Calculus

To show the idea in action, let us use a minimal extension of the simply-typed $\lambda$-calculus with encryption and decryption, and call it the *toy cryptographic $\lambda$-calculus*. We shall show how the idea works on this calculus, which is just a fragment of Sumii and Pierce's [18] cryptographic $\lambda$-calculus. The main thing that is missing here is nonce creation, i.e., fresh name creation.

For this moment, we 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 $\Sigma$ 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 $\tau$, $\mathtt{bits}[\tau]$ is the type of *ciphertexts* obtained by encrypting some value of type $\tau$—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 $\tau$.

The terms of the toy cryptographic $\lambda$-calculus are given by the grammar:

$$t, u, v, \ldots ::= \quad x \mid \lambda x \cdot t \mid tu \mid \{t\}_u \mid \texttt{let } \{x\}_t = u \texttt{ in } v_1 \texttt{ else } v_2$$

where $x$ ranges over a countable set of variables, $\{t\}_u$ denotes the ciphertext obtained by encrypting $t$ with key $u$ ($t$ is called the *plaintext*), and $\texttt{let } \{x\}_t = u \texttt{ in } v_1 \texttt{ else } v_2$ 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 $\alpha$-renaming are standard, hence omitted; $x$ is bound in $\lambda x \cdot t$, with scope $t$, as well in $\texttt{let } \{x\}_t = u \texttt{ in } v_1 \texttt{ else } v_2$, with scope $v_1$.

Typing is as one would expect. *Judgments* are of the form $\Gamma \vdash t : \tau$, where $\Gamma$ is a *context*, i.e., a finite mapping from variables to types. If $\Gamma$ maps $x_1$ to $\tau_1, \ldots, x_n$ to $\tau_n$, we write it $x_1 : \tau_1, \ldots, x_n : \tau_n$. Typing rules for encryption and decryption are

$$\frac{\Gamma \vdash t : \tau \qquad \Gamma \vdash u : \texttt{key}[\tau]}{\Gamma \vdash \{t\}_u : \texttt{bits}[\tau]} \; (Enc)$$

$$\frac{\Gamma \vdash t : \texttt{key}[\tau] \qquad \Gamma \vdash u : \texttt{bits}[\tau] \qquad \Gamma, x : \tau \vdash v_1 : \tau' \qquad \Gamma \vdash v_2 : \tau'}{\Gamma \vdash \texttt{let } \{x\}_t = u \texttt{ in } v_1 \texttt{ else } v_2 : \tau'} \; (Dec)$$

A simple denotational semantics for the typed toy cryptographic calculus is as follows. Let $[\![\_]\!]$ be any function mapping types $\tau$ 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 $\tau_1$ and $\tau_2$. Let $[\![b]\!]$ be arbitrary for every base type $b$, $[\![\texttt{key}[\tau]]\!]$ be arbitrary. For every $V \in [\![\tau]\!]$, $K \in [\![\texttt{key}[\tau]]\!]$, 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 [7], or in the spi-calculus [1].) Then, let $[\![\texttt{bits}[\tau]]\!]$ be the set of all pairs $\mathsf{E}(V, K)$, $V \in [\![\tau]\!]$, $K \in [\![\texttt{key}[\tau]]\!]$.

For any set $A$, let $A_\perp$ be the disjoint sum of $A$ with $\{\perp\}$, where $\perp$ is an element outside $A$, and let $\iota$ 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 $[\![\texttt{bits}[\tau]]\!] \times [\![\texttt{key}[\tau]]\!]$ 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 $\rho$ by structural induction on $t$,

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

More formally, for any context $\Gamma$, a $\Gamma$-*environment* $\rho$ is a map such that, for every $x : \tau$ in $\Gamma$, $\rho(x)$ 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 $V$.

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) $\lambda$-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 $\Gamma \vdash u : \tau_1$. We write $[\![t]\!]$ instead of $[\![t]\!] \rho$ when the environment $\rho$ is irrelevant, e.g., an empty environment.

## 3.2 What Are Logical Relations for Encryption?

We first fix a subset $\mathsf{Obs}$ of $\Sigma$, of so-called *observation types*. Typically, $\mathsf{Obs}$ will contain just the type `bool` of Booleans, one of the base types. We say that $a, a' \in [\![\tau]\!]$ are *contextually equivalent*, and we write $a \approx_\tau a'$, 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 ($o \in \mathsf{Obs}$), $[\![\mathcal{C}]\!] [x := a] = [\![\mathcal{C}]\!] [x := a']$.

In the $\lambda$-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 $\tau$, on $[\![\tau]\!]$, such that:

$$\textbf{(Log)} \quad \forall f, f' \in [\![\tau_1 \to \tau_2]\!], f \mathrel{\mathcal{R}_{\tau_1 \to \tau_2}} f' \Leftrightarrow (\forall a \mathrel{\mathcal{R}_{\tau_1}} a',\ f(a) \mathrel{\mathcal{R}_{\tau_2}} f'(a').$$

Here we write $a \mathrel{\mathcal{R}} a'$ 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 $\lambda$-calculus [12]. Note that there is no constraint on base types. In the typed $\lambda$-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 $\Gamma$-environments $\rho$, $\rho'$ are *related* by the logical relation, in notation $\rho \mathrel{\mathcal{R}_\Gamma} \rho'$, if and only if $\rho(x) \mathrel{\mathcal{R}_\tau} \rho'(x)$ for every $x : \tau$ in $\Gamma$. The basic lemma states that if $\Gamma \vdash t_0 : \tau$ is derivable, and $\rho$, $\rho'$ are two related $\Gamma$-environments, then $[\![t_0]\!] \rho \mathrel{\mathcal{R}_\tau} [\![t_0]\!] \rho'$. 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 [18], this implies that for any logical relation that coincides with equality on observation types, any two terms with logically related values are contextually equivalent.

In the toy cryptographic $\lambda$-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 $\lambda$-calculus. For any type $\tau$, let $\mathcal{R}_{\tau\ \mathtt{option}}$ be the binary relation on $[\![\tau]\!]_\perp$ defined by $V \mathrel{\mathcal{R}_{\tau\ \mathtt{option}}} V'$ if and only if $V = V' = \perp$, or $V = \iota(V_1)$, $V' = \iota(V_1')$ for some $V_1$, $V_1'$, and $V_1 \mathrel{\mathcal{R}_\tau} V_1'$.

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

Before we proceed, let us remark that we do not need *any* property of $\mathsf{E}$ or $\mathsf{D}$ in the proof of this lemma. The property that $\mathsf{D}(\mathsf{E}(V, K), K) = \iota(V)$ is only needed to show that `let` $\{x\}_t = \{u\}_t$ `in` $v_1$ `else` $v_2$ and $v_1[u/x]$ 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 $\mathsf{E}$ is even injective, is just never needed. This means that Lemma 1 also holds if we use encryption primitives that obey algebraic laws.

There is a kind of converse to Lemma 1. Assume that we have an additional type former $\tau\ \texttt{option}$, with constructors $\texttt{SOME}\ : \tau \rightarrow \tau\ \texttt{option}$ and $\texttt{NONE} : \tau\ \texttt{option}$. Assume their semantics is given by $[\![\tau\ \texttt{option}]\!] = [\![\tau]\!]_\perp$, $[\![\texttt{SOME}\ t]\!] = \iota([\![t]\!])$, $[\![\texttt{NONE}]\!] = \perp$. Finally, assume that $\mathcal{R}_{\tau\ \texttt{option}}$ 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 $dec = \lambda v \cdot \lambda k \cdot \texttt{let}\ \{x\}_k = v\ \texttt{in}\ \texttt{SOME}\ x\ \texttt{else}\ \texttt{NONE}$. If the basic lemma holds, then we must have $[\![enc]\!]\ \mathcal{R}_{\tau \rightarrow \texttt{key}[\tau] \rightarrow \texttt{bits}[\tau]}\ [\![enc]\!]$ and $[\![dec]\!]\ \mathcal{R}_{\texttt{bits}[\tau] \rightarrow \texttt{key}[\tau] \rightarrow \tau\ \texttt{option}}\ [\![dec]\!]$. 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.

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

In turn, $\mathcal{R}^\perp_{\texttt{bits}[\tau]} \subseteq \mathcal{R}^\top_{\texttt{bits}[\tau]}$ is equivalent to: for every $V\ \mathcal{R}_\tau\ V'$ and $K\ \mathcal{R}_{\texttt{key}[\tau]}\ K'$, for every $K_1\ \mathcal{R}_{\texttt{key}[\tau]}\ K'_1$, $\mathsf{D}(\mathsf{E}(V, K), K_1)\ \mathcal{R}_{\tau\ \texttt{option}}\ \mathsf{D}(\mathsf{E}(V', K'), K'_1)$ $(*)$. Let therefore $V\ \mathcal{R}_\tau\ V'$, and fix $K\ \mathcal{R}_{\texttt{key}[\tau]}\ K'$. By choosing $K_1 = K$, $(*)$ becomes $\iota(V)\ \mathcal{R}_{\tau\ \texttt{option}}\ \mathsf{D}(\mathsf{E}(V', K'), K'_1)$, 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, $\mathcal{R}_{\texttt{key}[\tau]}$ must be a *partial bijection* on $[\![\texttt{key}[\tau]]\!]$, i.e., the graph of a bijection between two subsets of $[\![\texttt{key}[\tau]]\!]$.

**Proposition 1** *Let $\mathcal{R}^0_b$ be given binary relations on $[\![b]\!]$ for every base type $b$. Let $\mathcal{R}^0_{\texttt{key}[\tau]}$ be any partial bijection on $[\![\texttt{key}[\tau]]\!]$ for every type $\tau$. 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 $\mathcal{R}_{\texttt{key}[\tau]} = \mathcal{R}^0_{\texttt{key}[\tau]}$ for every type $\tau$. We may define $\mathcal{R}_{\texttt{bits}[\tau]}$, for any type $\tau$, as any relation such that $\mathcal{R}^\perp_{\texttt{bits}[\tau]} \subseteq \mathcal{R}_{\texttt{bits}[\tau]} \subseteq \mathcal{R}^\top_{\texttt{bits}[\tau]}$.*

Proposition 1 shows that cryptographic logical relations exist that coincide with given relations on base types. But contrarily to logical relations in the $\lambda$-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 $\mathcal{R}_{\texttt{key}[\tau]}$, notably, we may use the intuition that some keys are observable by an intruder, and some others are not. Letting $fr_\tau$ 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 the identity on the subset $fr_\tau$ of $[\![\mathtt{key}[\tau]]\!]$. This is a popular choice: $fr_\tau$ 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 $\mathcal{R}^{\top}_{\mathtt{bits}[\tau]}$. 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 $K = K' \in fr_\tau$. 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 $fr_\tau$, plus a function $\psi_\tau$ 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 if $\mathsf{E}(V, K)\ \mathcal{R}_{\mathtt{bits}[\tau]}\ \mathsf{E}(V', K')$; on the other hand, given $\psi_\tau$, 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 $\psi$, 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 2** *Let $fr_\tau$ be some subset of $[\![\mathtt{key}[\tau]]\!]$, for each type $\tau$, and $\psi_\tau$ 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 $V\ \psi_\tau(K, K')\ V'$.*

*Whatever the choices of $fr_\tau$ and $\psi_\tau$, $(\mathcal{R}^{fr,\psi}_\tau)_{\tau \text{ type}}$ is a cryptographic logical relation.*

Clearly, Proposition 2 generalizes to the case where $fr_\tau$ and $\psi_\tau$ are not given *a priori*, but defined using the relations $\mathcal{R}^{fr,\psi}_{\tau'}$ for (not necessarily strict) subtypes $\tau'$ of $\tau$. That is, when not just $\mathcal{R}^{fr,\psi}_\tau$ but also $fr_\tau$ and $\psi_\tau$ are defined by mutual induction on types.

It is interesting, too, to relate the definition of $\mathcal{R}^{fr,\psi}_\tau$ to selected parts of the notion of framed bisimulation [2]. Slightly adapting [2] again, call a *theory* (on type $\mathtt{bits}[\tau]$) any *finite* binary relation $th_\tau$ on $[\![\mathtt{bits}[\tau]]\!]$. By finite, we mean that it should be finite as a set of pairs of values. A frame-theory pair $(fr_\tau, th_\tau)$ is *consistent* if and only if $th_\tau$ 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 $\psi_\tau$ 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 $fr_\tau$ and a function $\psi_\tau$.

## 4 A Uniform Cryptographic $\lambda$-Calculus, and Prelogical Relations

Reflecting on the developments above, we see that it would be more natural to use, instead of the toy cryptographic $\lambda$-calculus, a simply-typed $\lambda$-calculus with two constants `enc` and `dec`, with respective semantics given by $\mathsf{E}$ and $\mathsf{D}$. While we are at it, it is clear from the way we define $\mathcal{R}^0_{\mathtt{key}[\tau]}$ in Proposition 2 that the type $\mathtt{key}[\tau]$ behaves more like a base type than a type constructed from another type. It is therefore relevant to change the algebra of types to something like:

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

where $b$ ranges over $\Sigma$, $\Sigma$ 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 $\tau$ `option` type is used to give a typing to $\mathtt{dec} : \mathtt{bits}[\tau] \rightarrow \mathtt{key} \rightarrow \tau\ \mathtt{option}$; `enc` is assumed to have type $\tau \rightarrow \mathtt{key} \rightarrow \mathtt{bits}[\tau]$. 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 $\lambda$-calculus with constants... up to the fact that we need option types $\tau$ `option`. The constants to consider here are at least `dec`, `enc`, $\mathtt{SOME} : \tau \rightarrow \tau\ \mathtt{option}$, $\mathtt{NONE} : \tau\ \mathtt{option}$, and $\mathtt{case} : \tau\ \mathtt{option} \rightarrow (\tau \rightarrow \tau') \rightarrow \tau' \rightarrow \tau'$. (The `case` constant implements the elimination principle for $\tau$ `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, $\mathsf{D}$ and $\mathsf{E}$, related to themselves is reminiscent of *prelogical relations* [10]. These can be defined in a variety of ways. Following [10, Definition 3.1, Proposition 3.3], a *prelogical relation* is any family $(\mathcal{R}_\tau)_{\tau \text{ type}}$ of relations such that:
**1.** for every $f, f' \in [\![\tau_1 \rightarrow \tau_2]\!]$, if $f\ \mathcal{R}_{\tau_1 \rightarrow \tau_2}\ f'$ and $a\ \mathcal{R}_{\tau_1}\ a'$ then $f(a)\ \mathcal{R}_{\tau_2}\ f'(a')$;
**2.** $\mathsf{K}\ \mathcal{R}_{\tau_1 \rightarrow \tau_2 \rightarrow \tau_1}\ \mathsf{K}$, where $\mathsf{K}$ is the function mapping $x \in [\![\tau_1]\!]$, $y \in [\![\tau_2]\!]$ to $x$;
**3.** $\mathsf{S}\ \mathcal{R}_{(\tau_1 \rightarrow \tau_2 \rightarrow \tau_3) \rightarrow (\tau_1 \rightarrow \tau_2) \rightarrow \tau_1 \rightarrow \tau_3}\ \mathsf{S}$, where $\mathsf{S}$ is the function mapping $x \in [\![\tau_1 \rightarrow \tau_2 \rightarrow \tau_3]\!]$, $y \in [\![\tau_1 \rightarrow \tau_2]\!]$, $z \in [\![\tau_1]\!]$ to $x(z)(y(z))$;
**4.** and for every constant $a : \tau$, $[\![a]\!]\ \mathcal{R}_\tau\ [\![a]\!]$.
where $[\![a]\!]$ denotes $[\![a]\!]\,\rho$ for any environment $\rho$. Condition 1. is just one half of **(Log)**. The basic lemma for prelogical relations [10, 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 $a = [\![t]\!]$. The main point in our completeness argument is that there is a lax logical relation built by considering the trace of $\approx_\tau$ on definable elements. The relation is necessarily a partial equality on observation types $o \in \mathsf{Obs}$.

**Theorem 3 (Completeness)** *Prelogical relations are complete for contextual equivalence in the $\lambda$-calculus, in the strong sense that there is a prelogical relation $(\mathcal{R}_\tau)_{\tau \text{ type}}$ such that for every $t_1, t_2$ s.t. $\vdash t_1 : \tau, \vdash t_2' : \tau$, $[\![t_1]\!] \approx_\tau [\![t_2]\!]$ if and only if $[\![t_1]\!]\ \mathcal{R}_\tau\ [\![t_2]\!]$.*

The argument before Proposition 2 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 $fr \subseteq [\![\texttt{key}]\!]$ 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 3 (its proof is in the full version [9]), we get that $\mathcal{R}_{\texttt{key}}$ is exactly equality on the set $fr = \{[\![t]\!] \mid \vdash t : \texttt{key}\}$ of definable keys.

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

From this, we infer immediately the following combination of the analogue of Proposition 2 (soundness) with Theorem 3 (completeness):

**Proposition 4** *There is a prelogical relation $(\mathcal{R}^{fr,\psi}{}_\tau)_{\tau \text{ type}}$, parameterized by $fr$ and $\psi$, which is:*
- strict *at the* `key` *type: i.e., for every $K, K' \in [\![\texttt{key}]\!]$, $K \; \mathcal{R}^{fr,\psi}_{\texttt{key}} \; K'$ if and only if $K = K' \in fr$;*
- strict *at* $\texttt{bits}[\tau]$ *types: i.e., for every $V, V' \in [\![\tau]\!]$, for every $K, K' \in [\![\texttt{key}]\!]$, $\mathsf{E}(V, K) \; \mathcal{R}^{fr,\psi}_{\texttt{bits}[\tau]} \; \mathsf{E}(V', K')$ if and only if $V \; \mathcal{R}^{fr,\psi}_\tau \; V'$ and $K = K' \in fr$, or $K, K' \notin fr$ and $V \; \psi_\tau(K, K') \; V'$;*
- *and such that, for some $fr$ and $\psi$, for every closed terms $t, t'$ of type $\tau$, $[\![t]\!] \approx_\tau [\![t']\!]$ if and only if $[\![t]\!] \; \mathcal{R}^{fr,\psi}_\tau \; [\![t']\!]$.*

The idea of being *strict* at some type $\tau$ is, in all cases, that the (pre)logical relation at type $\tau$ should be defined uniquely as a function of the (pre)logical relations at all immediate subterms of $\tau$. The prelogical relation of Proposition 4 is strict at option types, too, provided there is a closed term of type $\tau$ or $[\![\tau]\!]$ has no junk.

While the point in prelogical relations in [10] is mainly of being not strict at arrow types, the point here is to argue that it is meaningful either not to be strict at $\texttt{bits}[\tau]$ types, as in Section 3.2 (in the sense that $\mathcal{R}_{\texttt{bits}[\tau]}$ was not determined uniquely from $\mathcal{R}_\tau$), or equivalently to be strict at $\texttt{bits}[\tau]$, given parameters $fr$ and $\tau$. We believe that just saying that we do not require strictness at $\texttt{bits}[\tau]$, thus omitting the $fr$ and $\tau$ parameters, leads to some simplification.

## 5 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 [17], who defined a categorical semantics for a calculus with fresh name creation based on Moggi's monadic $\lambda$-calculus [14]. We just take his language, adding all needed constants as in Section 4.

### 5.1 The Moggi-Stark Calculus

The *Moggi-Stark calculus* is obtained by adding a new type former $T$ (the *monad*), to the types of the $\lambda$-calculus of Section 4, so that $T\tau$ is a type as soon as $\tau$ is:

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

(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 $\lambda$-calculus of Section 4 is enriched with constructs $\texttt{val}\ t$ and $\texttt{let}\ x \Leftarrow t\ \texttt{in}\ u$ (not to be confused with the `let` construct of Section 3.1), with typing rules as following, and two constants $\texttt{new} : T\boldsymbol{\nu}$ (fresh name creation) and $\doteq : \boldsymbol{\nu} \to \boldsymbol{\nu} \to \texttt{bool}$ (equality of names).

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

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 $\rho$ *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 $\boldsymbol{Set}^{\mathcal{I}}$ [17], where $\mathcal{I}$ is the category whose objects are finite sets and whose morphisms $s \xrightarrow{i} s'$ are injections. Given any type $\tau$, $[\![\tau]\!]\, s$ is intuitively the set of all values of type $\tau$ 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 $\Gamma$ 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 [11], 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 $\Gamma, s$*-environment* $\rho$ (a map sending each $x_i$ such that $x_i : \tau_i$ is in $\Gamma$ 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 $\tau$, 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 in [17, 8, 19]. $\boldsymbol{T}A$ is defined by $\operatorname{colim}_{s'} A(\_ + s') : \mathcal{I} \to \boldsymbol{Set}$. On objects, this is given by $\boldsymbol{T}As = \operatorname{colim}_{s'} A(s + s')$, 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.

The semantics of `let` and `val` is standard [14]. Making it explicit on this particular monad, we obtain: $[\![\texttt{val}\ t]\!]\, s\rho = (\nu\emptyset)\, [\![t]\!]\, s\rho$ and $[\![\texttt{let}\ x \Leftarrow t\ \texttt{in}\ u]\!]\, s\rho = (\nu\ s' + s'')b$, 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 $[\![u]\!]\, (s + s')(([\![\Gamma]\!]\, (\mathsf{inl}_{s,s'})\rho)[x := a]) = (\nu s'')b$. (Concretely, if $\Gamma$ 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 $[\![\Gamma]\!]\, (\mathsf{inl}_{s,s'})\rho$ is $[x_1 := [\![\tau_1]\!]\, (\mathsf{inl}_{s,s'})a_1, \ldots, x_n := [\![\tau_n]\!]\, (\mathsf{inl}_{s,s'})a_n]$.)

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., $[\![\texttt{bool}]\!]\, s = \mathbb{B}$. 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 $s$. $\boldsymbol{Set}^{\mathcal{I}}$ is a presheaf category, hence cartesian-closed [11]. This provides a semantics for $\lambda$-abstraction and applications.

Finally, the semantics of `new` $: T\boldsymbol{\nu}$ is given by $[\![\texttt{new}]\!]\, s\rho = (\nu\{n\})n$, where $n$ is any element not in $s$, and $[\![\doteq]\!]$ is defined as the only morphism in $\boldsymbol{Set}^{\mathcal{I}}$ such that $[\![\doteq xy]\!]\, s[x := a, y := b]$ is `true` if $a = b$, and `false` otherwise.

## 5.2 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* [16]. We introduce this notion as gently as possible.

Let $\Sigma$ be the set of base types, seen as a discrete category. The simply-typed $\lambda$-calculus gives rise to the *free CCC* $\boldsymbol{\lambda}(\Sigma)$ over $\Sigma$ as follows: the objects of $\boldsymbol{\lambda}(\Sigma)$ are typing contexts $\Gamma$, a morphism from $\Gamma$ 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, $\Gamma$-environments are exactly morphisms from the terminal object, the empty context $\epsilon$, to $\Gamma$.) Composition is substitution. Being the free CCC means that, for any CCC $\boldsymbol{C}$, for any functor $[\![\_]\!]_0$ from $\Sigma$ to $\boldsymbol{C}$ (i.e., for any function $[\![\_]\!]_0$ mapping each base type in $\Sigma$ to some object in $\boldsymbol{C}$), there is a unique representation $[\![\_]\!]_1$ of CCCs from $\lambda(\Sigma)$ to $\boldsymbol{C}$ such that the right diagram commutes. 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 $\tau$) and $[\![t]\!]_1$ (denotations of typed $\lambda$-terms $t$) as used in Section 3.

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

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 [13] 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 $\Sigma$ 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 right diagram commutes. 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 $U \circ \mathcal{R} = [\![\_]\!]_1$, i.e., diagram (3) commutes. As observed in [13], 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

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

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 $S \hookrightarrow [\![\tau]\!]^2$, where $S$, up to isomorphism, is just a subset of the cartesian product of $[\![\tau]\!]$ 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 $\tau$, to a morphism in the subscone, i.e., a pair $\langle u, v\rangle$. The fact that diagram (3) commutes 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.

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

The important property to make $\mathcal{R}$ satisfy the basic lemma is just the equality in the right diagram. 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 [16, Section 6]. The difference is 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 meta-language follows the same pattern. The monadic $\lambda$-calculus gives rise to the *free let-CCC* $\boldsymbol{Comp}(\Sigma)$ over $\Sigma$, 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).

### 5.3 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 $\tau$, cannot observe anything. This is because the $(\texttt{let})$ typing rule 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 $x : \tau$. We must assume that $\mathcal{C}$ can also depend on an arbitrary set of public names. Given names $n_1, \ldots, n_m$, the only way $\mathcal{C}$ can be made to depend on them is to assume that $\mathcal{C}$ has $m$ free variables $z_1, \ldots, z_m$ of type $\boldsymbol{\nu}$, which are mapped to $n_1, \ldots, n_m$. (It is more standard [15, 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 $z_l$ mapped, in a one-to-one way, to names $n_l$.) Let $s_1$ be any set of names containing $n_1, \ldots, n_m$, let $w_1$ be $\underline{\{z_1, \ldots, z_m\}}$, and $w_1 \overset{i_1}{\rightarrow} s_1$ the injection mapping each $z_l$ to $n_l$, $1 \leq l \leq m$. Write $\overline{w_1 := i_1(w_1)}$ for $z_1 := n_1, \ldots, z_m := n_m$, and $\overline{w_1 : \boldsymbol{\nu}}$ for $z_1 : \boldsymbol{\nu}, \ldots, z_m : \boldsymbol{\nu}$. 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 requires $a$ and $a'$ to be in $[\![\tau]\!]\, s_1$, but they are in $[\![\tau]\!]\, s$ for some possibly different set $s$ of names. This is repaired by considering coercion $[\![\tau]\!]\, k_1$, where $s \xrightarrow{k_1} s_1$ is any injection.

To sum up, say that $a, a' \in [\![\tau]\!]\, s$ are *contextually equivalent at* $s$, and write $a \approx_\tau^s a'$, if and only if, for every finite set of variables $w_1$, for every injections $w_1 \xrightarrow{i_1} s_1$ and $s \xrightarrow{k_1} s_1$, for every term $\mathcal{C}$ such that $\overline{w_1 : \boldsymbol{\nu}}, x : \tau \vdash \mathcal{C} : T o$ 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 [15, 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 $a \approx_\tau^s a'$ if and only if, for every injection $s \xrightarrow{k_1} s_1$, for every term $\mathcal{C}$ such that $\overline{s_1 : \boldsymbol{\nu}}, x : \tau \vdash \mathcal{C} : T o$ 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 $\boldsymbol{Set}^{\mathcal{I}^{\to}}$, where $\mathcal{I}^{\to}$, the *arrow category* of $\mathcal{I}$, has as objects all morphisms $w \xrightarrow{i} s$ in $\mathcal{I}$, and as morphisms from $w \xrightarrow{i} s$ to $w' \xrightarrow{i'} s'$ all pairs $(j, k)$ of morphisms such that the right diagram commutes. This is in accordance with [19], where it is noticed that $\boldsymbol{Set}^{\mathcal{I}^{\to}}$ is the right category to define a Kripke logical relation (but not necessarily lax) 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 $\mathcal{I}^{\to}$). Objects $w \xrightarrow{i} s$ are then sets $w$ of variables denoting those public names in $s$, together with an injection $i$. So we shall work with lax logical relations in the subscone category $\mathrm{Subscone}_{\boldsymbol{C}}^{\mathbb{C}}$, where $\boldsymbol{C} = \boldsymbol{Set}^{\mathcal{I}} \times \boldsymbol{Set}^{\mathcal{I}}$, $\mathbb{C}$ is the presheaf category $\boldsymbol{Set}^{\mathcal{I}^{\to}}$, and $|\_| : \boldsymbol{C} \to \mathbb{C}$ is the composite of the binary product functor $\times : \boldsymbol{Set}^{\mathcal{I}} \times \boldsymbol{Set}^{\mathcal{I}} \to \boldsymbol{Set}^{\mathcal{I}}$ with the functor $\boldsymbol{Set}^{\mathfrak{u}} : \boldsymbol{Set}^{\mathcal{I}} \to \boldsymbol{Set}^{\mathcal{I}^{\to}}$. Here $\mathfrak{u} : \mathcal{I}^{\to} \to \mathcal{I}$ 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 $a = [\![t]\!]\, s[\overline{w := i(w)}]$.

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

**Definition 1** *Let* $w \xrightarrow{i} s$ *be any object of* $\mathcal{I}^{\to}$*. The value* $a, a' \in [\![\tau]\!]\, s$ *are said to be* contextually equivalent at $w \xrightarrow{i} s$*, written* $a \approx_\tau^{w \xrightarrow{i} s} a'$*, 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* $\mathcal{I}^{\to}$*, for every term* $\mathcal{C}$ *such that* $\overline{w_1 : \boldsymbol{\nu}}, x : \tau \vdash \mathcal{C} : T o$ *(*$o \in \mathsf{Obs}$*) 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* $\mathcal{R}_\tau^{w \xrightarrow{i} s}$ *by:* $a\ \mathcal{R}_\tau^{w \xrightarrow{i} s}\ a'$ *if and only if* $a$ *and* $a'$ *are definable at* $w \xrightarrow{i} s$ *and* $a \approx_\tau^{w \xrightarrow{i} s} a'$*.*

In particular, $a \approx_\tau^s a'$ iff $a \approx_\tau^{\emptyset \to s} a'$, where $\emptyset \to s$ denotes the unique empty injection.

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 for $t'$ obtained from t by renaming according to $j$,

$$[\![\tau]\!]\, k(a) = [\![t']\!]\, s'[\overline{w' := i'(w')}]. \tag{1}$$

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

**Theorem 5** *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, $[\![u]\!]\, s[\overline{w := i(w)}] \approx_\tau^{w \xrightarrow{i} s} [\![u']\!]\, s[\overline{w := i(w)}]$ iff $[\![u]\!]\, s[\overline{w := i(w)}]\ \mathcal{R}_\tau^{w \xrightarrow{i} s}\ [\![u']\!]\, s[\overline{w := i(w)}]$.*

The (non-lax) logical relation of [19] is defined on $\boldsymbol{\nu}$ by: $n\ \mathcal{R}_{\boldsymbol{\nu}}^{w \xrightarrow{i} s}\ n'$ iff $n = n' \in w$. This is exactly what the lax logical relation of Definition 1 is defined as on the $\boldsymbol{\nu}$ type:

**Lemma 2.** *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 $n = n' \in i(w)$.*

To finish this section, we observe:

**Lemma 3.** *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 $o$.*

We almost forgot to prove soundness! It is easy to see that any lax logical relation that coincides with partial equality on types $To$ is 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 $a \approx_\tau^{w \xrightarrow{i} s} a'$.

## 5.4 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 $\boldsymbol{Set}^{\mathcal{I}^{\rightarrow}}$, as used in Section 5.3… making sure that they relate each constant 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. The only requirement that the new constructs can be given a semantics in $\boldsymbol{Set}^{\mathcal{I}}$. In particular, a lax logical relation on $\boldsymbol{Set}^{\mathcal{I}^{\rightarrow}}$ is sound for observational equivalence in the presence of cryptographic primitives if each of the constants `enc`, `dec`, `SOME` , `NONE`, `case` is related to itself.

Then Theorem 5 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 remain complete. 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.

**Acknowledgements:** We would like to thank Michel Bidoit for having directed us 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)*, 1997.
2. M. Abadi and A. D. Gordon. A bisimulation method for cryptographic protocols. *Nordic Journal of Computing*, 5(4), 1998.
3. M. Alimohamed. A characterization of lambda definability in categorical models of implicit polymorphism. *Theoretical Computer Science*, 146(1–2), 1995.
4. M. Boreale, R. de Nicola, and R. Pugliese. Proof techniques for cryptographic processes. In *Proc. LICS'99*. IEEE Computer Society Press, 1999.
5. J. Borgström and U. Nestmann. On bisimulations for the spi calculus. In *Proc. AMAST'02*, volume 2422 of *LNCS*. Springer, 2002.
6. 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, 2002.
7. D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, IT-29(2), 1983.
8. J. Goubault-Larrecq, S. Lasota, and D. Nowak. Logical relations for monadic types. In *Proc. CSL'02*, volume 2471 of *LNCS*. Springer, 2002.
9. J. Goubault-Larrecq, S. Lasota, D. Nowak, and Y. Zhang. Complete lax logical relations for cryptographic lambda-calculi. Research Report, LSV, ENS de Cachan, 2004.
10. F. Honsell and D. Sannella. Pre-logical relations. In *Proc. CSL'99*, volume 1683 of *LNCS*, 1999.
11. J. Lambek and P. J. Scott. *Introduction to Higher Order Categorical Logic*, volume 7 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, 1986.
12. J. C. Mitchell. *Foundations for Programming Languages*. MIT Press, 1985.
13. J. C. Mitchell and A. Scedrov. Notes on sconing and relators. In *Proc. CSL'93*, volume 702 of *LNCS*. Springer, 1993.
14. E. Moggi. Notions of computation and monads. *Information and Computation*, 93, 1991.
15. 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 *LNCS*. Springer, 1993.
16. G. D. Plotkin, J. Power, D. Sannella, and R. D. Tennent. Lax logical relations. In *Proc. ICALP'00*, volume 1853 of *LNCS*. Springer, 2000.
17. I. Stark. Categorical models for local names. *Lisp and Symbolic Computation*, 9(1), 1996.
18. E. Sumii and B. C. Pierce. Logical relations for encryption. In *Proc. CSFW-14*. IEEE Computer Society Press, 2001.
19. Y. Zhang and D. Nowak. Logical relations for dynamic name creation. In *Proc. CSL/KGL'03*, volume 2803 of *LNCS*. Springer, 2003.