Logical Relations for Names

ZHANG Yu

Supervised by
Jean GOUBAULT-LARRECQ and David NOWAK

A report towards the degree for D.E.A. Programmation

September 2002, Paris

Abstract

The nu-calculus is an extension of the simply-typed lambda calculus with a ground type for names. It is a useful formal system to study dynamically generated *names*. In this report, we first present briefly the syntax and the operational semantics of the nu-calculus. We also introduce a metalanguage in the style of Moggi's computational lambda-calculus and give the interpretation of the nu-calculus in such a metalanguage.

The idea of *logical relation* is used to derive powerful reasoning methods that capture some of the distinction between private and public names. Pitts and Stark have shown that it is a powerful technique for studying contextual equivalence. They provide a logical relation for the nu-calculus in a syntactical and operational way. This report gives another logical relation for the interpretation of the nu-calculus in the metalanguage, based on the idea of *Kripke logical relation* and *renaming* in programming languages. We show that, due to some renaming strategy, this logical relation is equivalent to that of Pitts and Stark.

Contents

Chapter 1

Introduction

The technique of Internet and e-commerce developed very rapidly in recent years. More and more business is carried on through Internet. Motivated by this, a good security policy is extremely necessary to make the business run smoothly and this is the security problem of computer systems. In computer security, information hiding is a central concern, which means that your private data, or say, your secret data should not be observable by anyone not authorized. Encryption is a fundamental technique of hiding information from outside. However, cryptographic protocols are very difficult to be guaranteed. For example, Needham-Schroeder's public-key protocol is a well-known cryptographic protocol. which was put forward in 1978, and in a period as long as 17 years, it had been regarded as a perfect protocol until Lowe found a flaw in this protocol in 1995 [2].

The main difficulty of verifying cryptographic protocols is to find a formal language which is simple and complete enough to describe the protocols. Sumii and Pierce defined a cryptographic lambda-calculus, which is an extension of the ordinary simply-typed lambda-calculus with primitives of encryption, decryption and key generation [7]. They stated that information hiding is also a central concern in programming languages and it is natural to make some of the techniques from the programming language setting applied to security problems. Desired secrecy properties can be formalized in terms of *observational equivalence*. For instance, we have a program keeping some data secret against all possible attacks, then it is required that, whatever the secret data the program carries, the observational behavior is always the same. It can be formalized as $f(p_i) = f(p_j)$ for $i \neq j$, where i, j is the secret data, p is the program carrying secret data and f is the attacker.

The proof technique used by Sumii and Pierce is *logical relation*.

The nu-calculus is a small language designed to study dynamically generated names in programming languages [5, 6]. It is also an extension of the simply-typed lambda-calculus with a ground type for names and a primitive of dynamical name generation. It is easy to see that the nu-calculus is a syntactical subset of the cryptographic lambda-calculus when we regard the type for names and the type for keys as a same type. In this report, we will study some logical relations for the nu-calculus.

We first give a description of the syntax and the operational semantics of the nu-calculus. We also present a computational metalanguage in Moggi's style and interpret the nu-calculus in this metalanguage. Contextual equivalence is an important concept in both the fields of programming languages and cryptography and we give one notion for this equivalence in the nu-calculus. All of these are presented in *Chapter 2* and *Chapter 3* and there is nothing new in these two chapters.

After the introductory chapters, we study several logical relations for the nu-calculus, including Pitts and Stark's operational logical relation, which is defined syntactically and operationally on the nu-calculus terms. According to the logical relation given by Goubault-Larrecq, Lasota and Nowak in [1], we define a Kripke logical relation on the interpretation of the nu-calculus in the computational metalanguage. We demonstrate that this Kripke logical relation is not equivalent to the operational logical relation up to contextual equivalence, so we make some improvement on it and show that, with some *renaming* strategy, the improved Kripke logical relation is equivalent to the operational logical relation.

Chapter 2

The Nu-Calculus

The nu-calculus is a small language designed to show the interaction between dynamically generated names and higher-order functions [5, 6]. We give its syntax and describe an operational semantics in a 'big step' evaluation style. We go on to define a notion of contextual equivalence for expressions of the language and present some examples of expressions that are equivalent or inequivalent.

The nu-calculus is a typed call-by-value lambda-calculus extended with the notion of a *name*; names have their own type ν, they can be created fresh, passed around and tested for equality. Higher-order functions and booleans are also available, but to ensure termination functions cannot be defined recursively. New names are created in expressions of the form $\nu n.M$, which binds a fresh name to the identifier n and then evaluates M. The use of a call-by-value semantics means that although only the expression M can refer to n explicitly, the new name itself may escape from this scope. For example, $(\lambda x^\nu.x = x)(\nu n.n)$ evaluates to *true*, with x bound to a name rather than to $\nu n.n$ itself.

2.1 Syntax

The syntax of the nu-calculus is based on the simply-typed lambda-calculus. Types take the form

$$\sigma ::= o \mid \nu \mid \sigma \to \sigma$$

where o and ν are two ground types denoting *booleans* and *names* respectively. Terms take the form

$$
\begin{array}{rcll}
M & ::= & x & \text{variable} \\
 & | & n & \text{name} \\
 & | & \mathit{true} \mid \mathit{false} & \text{boolean values} \\
 & | & \mathit{if}\ M\ \mathit{then}\ M\ \mathit{else}\ M & \text{conditional} \\
 & | & M = M & \text{equality of names} \\
 & | & \nu n.M & \text{local name declaration} \\
 & | & \lambda x^{\sigma}.M & \text{function abstraction} \\
 & | & MM & \text{function application}
\end{array}
$$

where $x \in Var$, an infinite set whose elements are called *variables*, and $n \in Nme$, an infinite set (disjoint from Var) whose elements are called *names*. Function abstraction is a variable-binding construct (occurrences of x in M are bound in $\lambda x^{\sigma}.M$), whereas local name declaration is a name-binding construct (occurrences of n in M are bound in $\nu n.M$). We implicitly identify expressions which only differ in their choice of bound variables and names (α-conversion).

We write $\mathrm{FV}(M)$ and $\mathrm{FN}(M)$ for the finite subsets of Var and Nme consisting of the free variables and the free names in a term M. An expression is *closed* if it has no free variables; a closed expression may still have free names. A useful abbreviation is *new* for $\nu n.n$; this is the expression that generates a new name and then immediately returns it.

We denote by $M[M'/x]$ (respectively $M[M'/n]$) the result of substituting the expression M' for free occurrences of the variable x (respectively, the name n) in the expression M. The substitution is *capture avoiding.* This can always be arranged by α-converting M. Simultaneous substitution is also capture avoiding; we write this as $M[M_1/x_1, \cdots, M_n/x_n]$, often abbreviated to $M[\vec{M}/\vec{x}]$.

Expressions will be assigned types via typing assertions of the form

$$s, \Gamma \vdash M : \sigma$$

where s is a finite subset of Nme, Γ is a finite set of typed variables, σ is a type, and M is an expression (more precisely, an α-equivalence class of expressions) with free names in s and free variables in Γ. Expressions are given types according to the following rules:

$$\frac{}{s, \Gamma \vdash x : \sigma}(x : \sigma \in \Gamma) \qquad \frac{}{s, \Gamma \vdash b : o}(b = \mathit{true}, \mathit{false})$$

$$\frac{}{s, \Gamma \vdash n : \nu}(n \in s) \qquad \frac{s, \Gamma \vdash B : o \quad s, \Gamma \vdash M : \sigma \quad s, \Gamma \vdash M' : \sigma}{s, \Gamma \vdash \mathit{if}\ B\ \mathit{then}\ M\ \mathit{else}\ M' : \sigma}$$

$$\frac{s, \Gamma \vdash N : \nu \quad s, \Gamma \vdash N' : \nu}{s, \Gamma \vdash (N = N') : o} \qquad \frac{s \oplus \{n\}, \Gamma \vdash M : \sigma}{s, \Gamma \vdash \nu n.M : \sigma}$$

$$\frac{s, \Gamma \oplus \{x : \sigma\} \vdash M : \sigma'}{s, \Gamma \vdash \lambda x^\sigma.M : \sigma \to \sigma'} \qquad \frac{s, \Gamma \vdash F : \sigma \to \sigma' \quad s, \Gamma \vdash M : \sigma}{s, \Gamma \vdash FM : \sigma'}$$

The symbol $\oplus$ represents disjoint union, here in $s \oplus \{n\}$ and $\Gamma \oplus \{x : \sigma\}$. Also, we use $s_1 \pitchfork s_2$ to denote that s_1 and s_2 are two disjoint sets. Clearly, if $s, \Gamma \vdash M : \sigma$ holds, then σ is uniquely determined by s, Γ and M. We may omit Γ if it is empty. From now on we shall consider only well-typed expressions.

An expression is in *canonical form* if it is either a name, a variable, one of the boolean constants $true$ or $false$, or a function abstraction. These are to be the *values* of the nu-calculus, and correspond to weak head normal form in the lambda-calculus. We define the sets

$$\begin{aligned}
\mathrm{Exp}_\sigma(s, \Gamma) &= \{M \mid s, \Gamma \vdash M : \sigma\} \\
\mathrm{Can}_\sigma(s, \Gamma) &= \{C \mid C \in \mathrm{Exp}_\sigma(s, \Gamma), C \text{ canonical}\} \\
\mathrm{Exp}_\sigma(s) &= \mathrm{Exp}_\sigma(s, \emptyset) \\
\mathrm{Can}_\sigma(s) &= \mathrm{Can}_\sigma(s, \emptyset)
\end{aligned}$$

of expressions and canonical expressions at any type σ and for any finite sets s, Γ of names and typed variables.

2.2 Evaluation Semantics

The operational semantics of the nu-calculus is specified by the inductively defined *evaluation relation*. Elements of the relation take the form

$$s \vdash M \Downarrow^{s'}_\sigma C$$

where $s \pitchfork s'$, $M \in \mathrm{Exp}_\sigma(s)$ and $C \in \mathrm{Can}_\sigma(s \oplus s')$. This is intended to mean that in the presence of the names s, expression M of type σ evaluates to canonical form C and creates fresh names s'. The set s and $s \oplus s'$ can be seen as initial and final *states* of the computation. The arrangement of these states shows the left to right order of evaluation. We may omit s or s' when they are empty.

The rule for generating the relation are

$$\text{(CAN)} \qquad \frac{}{s \vdash C \Downarrow_\sigma C}$$

$$\text{(COND1)} \qquad \frac{s \vdash B \Downarrow_o^{s_1} true \quad s \oplus s_1 \vdash M \Downarrow_\sigma^{s_2} C}{s \vdash if\ B\ then\ M\ else\ M' \Downarrow_\sigma^{s_1 \oplus s_2} C}$$

$$\text{(COND2)} \qquad \frac{s \vdash B \Downarrow_o^{s_1} false \quad s \oplus s_1 \vdash M' \Downarrow_\sigma^{s_2} C'}{s \vdash if\ B\ then\ M\ else\ M' \Downarrow_\sigma^{s_1 \oplus s_2} C'}$$

$$\text{(EQ1)} \qquad \frac{s \vdash N \Downarrow_\nu^{s_1} n \quad s \oplus s_1 \vdash N' \Downarrow_\nu^{s_2} n}{s \vdash (N = N') \Downarrow_o^{s_1 \oplus s_2} true}, \qquad n \in s$$

$$\text{(EQ2)} \qquad \frac{s \vdash N \Downarrow_\nu^{s_1} n \quad s \oplus s_1 \vdash N' \Downarrow_\nu^{s_2} n'}{s \vdash (N = N') \Downarrow_o^{s_1 \oplus s_2} false}, \qquad n, n' \text{ distinct}$$

$$\text{(LOCAL)} \qquad \frac{s \oplus \{n\} \vdash M \Downarrow_\sigma^{s_1} C}{s \vdash \nu n.M \Downarrow_\sigma^{\{n\} \oplus s_1} C}$$

$$\text{(APP)} \qquad \frac{\begin{array}{c} s \vdash F \Downarrow_{\sigma \to \sigma'}^{s_1} \lambda x^\sigma.M' \quad s \oplus s_1 \vdash M \Downarrow_\sigma^{s_2} C \\ s \oplus s_1 \oplus s_2 \vdash M'[C/x] \Downarrow_{\sigma'}^{s_3} C' \end{array}}{s \vdash FM \Downarrow_{\sigma'}^{s_1 \oplus s_2 \oplus s_3} C'.}$$

Occasionally, it is necessary to relabel bound names when applying these evaluation rules. For example, to evaluate $\nu n.n = \nu n.n$ we do not use $\vdash \nu n.n = \nu n.n \Downarrow_o^{n,n} true$ because it is not well formed; new names have to be distinct, and this is enforced by the use of disjoint union $\oplus$ in the (LOCAL) rule. Instead we relabel one of the n's to obtain $\vdash \nu n.n = \nu n'.n' \Downarrow_o^{n,n'} false$.

The abbreviation *new* for $\nu n.n$ has the derived evaluation rule

$$\text{(NEW)} \qquad \frac{}{s \vdash new \Downarrow_\nu^{\{n\}} n}, \qquad n \notin s.$$

The side condition confirms that the name generated by *new* is fresh, and corresponds precisely to the disjoint union $s \oplus \{n\}$ in the premise of the (LOCAL) rule for evaluating $\nu n.M$. Indeed the rules (LOCAL) and (NEW) are entirely equivalent, and we could formulate the nu-calculus with *new* as primitive and $\nu n.M$ an abbreviation for $(\lambda x^\nu.M[x/n])new$, where x is a fresh variable. This then makes precise the connection between the relabelling of names bound by ν and of variables bound by λ. In a setting with *new* primitive, they are the same thing.

Here are some propositions on the evaluating semantics, which are already proved in [5, 6].

Proposition 2.1 *If $s \vdash M \Downarrow_\sigma^{s_1} C$ and $s \vdash M \Downarrow_\sigma^{s_2} C'$, then there is a bijection $R : s_1 \leftrightarrow s_2$ so that C' is α-convertible with the expression $C[n'/n \mid (n, n') \in R]$.*

Proposition 2.2 (Termination) *For all closed expressions M, of type σ and with free names in the set s say, there is some set of names s' (disjoint from s) and some canonical expression $C \in \mathrm{Can}_\sigma(s \oplus s')$ such that $s \vdash M \Downarrow_\sigma^{s'} C$. Moreover, these are unique up to relabelling the elements of s' and α-conversion of C.*

2.3 Contextual Equivalence

We construct a notion of equivalence for expressions of the nu-calculus, based on their behavior when used in larger expressions. Informally, two expressions are equivalent if they can be freely exchanged; there is no way in the language itself to distinguish between them.

A *program context* $P\langle - \rangle$ is a program with zero or more occurrences of a hole $\langle - \rangle$. If M is some expression then $P\langle M \rangle$ is the program obtained by substituting M for each occurrence of this hole.

Definition 2.1 (Contextual Equivalence) If $M_1, M_2 \in \mathrm{Exp}_\sigma(s, \Gamma)$ then the assertion

$$s, \Gamma \vdash M_1 \approx_\sigma M_2$$

means that for all suitable typed program contexts $P\langle - \rangle$ defined over s, and each boolean value $b \in \{true, false\}$,

$$\exists s_1 (s \vdash P\langle M_1 \rangle \Downarrow_o^{s_1} b) \iff \exists s_2 (s \vdash P\langle M_2 \rangle \Downarrow_o^{s_2} b).$$

In this case, we say M_1 and M_2 are contextually equivalent.

This is the most general notion of equivalence for nu-calculus expressions. Other equivalences are useful if they imply contextual equivalence. and are easier to compute. Here are some examples of the contextual equivalence in the nu-calculus.

Example 2.1 General β-equivalence fails to make the β-equivalent terms to be observationally equivalent because the nu-calculus is call-by-value:

$$(\lambda x^\nu . x = x) new \not\approx_o (new = new).$$

Example 2.2 Local name declaration and function abstraction do not commute in general:

$$\nu n.\lambda x^o.n \not\approx_{o\to\nu} \lambda x^o.\nu n.n.$$

The left expression generates a new name first and then for any argument of type o, it returns this new name, while the right one returns the expression *new*.

Example 2.3 Expressions may be contextually equivalent if they differ only in their use of 'private' names:

$$\emptyset \vdash \nu n.\lambda x^v.(x = n) \ \approx_{v\to o} \ \lambda x^v.false \ .$$

The name n is a private name of the function, and no external context can supply this name. In fact, any external context can tell nothing about the name n at all.

Example 2.4 Similarly, any externally produced function cannot distinguish the private names:

$$\emptyset \vdash \nu n.\nu n'.\lambda f^{v\to o}.(fn = fn') \ \approx_{(v\to o)\to o} \ \lambda f^{v\to o}.true \ .$$

In this example the boolean test $fn = fn'$ is an abbreviation for

$$if\ fn\ then\ fn'\ else\ (if\ fn'\ then\ false\ else\ true).$$

Pitts and Stark also provides several ways to prove these examples [5, 6]. But in this report, we are just interested in the way using the technique of *logical relation*, which will be discussed in detail in *Chapter 4*.

Chapter 3

The Computational Metalanguage

Consideration of the nu-calculus in *Chapter 2* is entirely operational. In [6], Stark has developed a denotational semantics for the language, using a categorical strong monad to encapsulate the notion of 'computation'.

This chapter will not be concerned with anything categorical. We just introduce the particular metalanguage used by Stark to construct his categorical model. The metalanguage is based on Moggi's computational lambda-calculus, which can be regarded as an extension of simply-typed lambda-calculus with the notion of a *computation* [4]. The most important feature of the metalanguage is that it distinguishes between values and computations. For the nu-calculus, a computation will create some names and then return a value. We also describe an interpretation of the nu-calculus in the metalanguage. It respects the operational semantics.

3.1 Syntax

The types of the metalanguage are given by :

$$\begin{array}{rcll} A & ::= & Bool & \text{booleans} \\ & | & Name & \text{names} \\ & | & A \to A & \text{functions} \\ & | & TA & \text{computations.} \end{array}$$

There is a unary type constructor T: if A is a type, then elements of TA are *computations* of type A. In this particular metalanguage the difference between a value and a computation is that a computation may generate new names before returning a value.

The term forming operations of the metalanguage are as follows:

$$
\begin{array}{rcll}
a & ::= & x & \text{variable} \\
 & | & tt \mid ff & \text{boolean values} \\
 & | & cond(a,a,a) & \text{conditional} \\
 & | & eq(a,a) & \text{compare names} \\
 & | & new & \text{generate new name} \\
 & | & \lambda x^A.a & \text{function abstraction} \\
 & | & aa & \text{function application} \\
 & | & [a] & \text{value as trivial computation} \\
 & | & let\ x \Leftarrow a\ in\ a & \text{sequential computation}
\end{array}
$$

Function abstraction $\lambda x^A.a$ binds the variable x in the term a, and sequential computation $let\ x \Leftarrow e\ in\ e'$ binds the variable x within the term e'. We implicitly identify terms up to α-conversion, which allows us to require substitution to be capture avoiding. A term is *closed* if it has no free variables; there is no possibility of free names.

There are three forms of term involving computation. If a is a value, then $[a]$ is the trivial computation which simply returns a. The sequential form $let\ x \Leftarrow e\ in\ e'$ carries out the computation e, binds the result to x and then computes e'. Both of these are standard constructions of the computational lambda-calculus. Besides them, there is a constant new of type $TName$ which denotes the computation that generates a new name.

Type judgments of the metalanguage are of the form

$$\Gamma \vdash a : A$$

which asserts that in the presence of Γ, term a has type A. Here Γ is a finite set of typed variables. Unlike the nu-calculus, there is no set of free names. The symbol $\vdash$ we used to construct type assertions is the same as in the nu-calculus, but it can easily be identified according to the context. The rules for forming type judgments are given by

$$\frac{}{\Gamma \vdash x : A}(x : A \in \Gamma) \qquad \frac{}{\Gamma \vdash new : TName}$$

$$\frac{}{\Gamma \vdash tt : Bool} \qquad \frac{}{\Gamma \vdash \mathit{ff} : Bool}$$

$$\frac{\Gamma \vdash n, n' : Name}{\Gamma \vdash eq(n, n') : Bool} \qquad \frac{\Gamma \vdash b : Bool \quad \Gamma \vdash a, a' : A}{\Gamma \vdash cond(b, a, a') : A}$$

$$\frac{\Gamma \vdash a : A}{\Gamma \vdash [a] : TA} \qquad \frac{\Gamma, x : A \vdash b : B}{\Gamma \vdash \lambda x^A.b : A \to B}$$

$$\frac{\Gamma \vdash e : TA \quad \Gamma, x : A \vdash e' : TA'}{\Gamma \vdash let\ x \Leftarrow e\ in\ e' : TA'} \qquad \frac{\Gamma \vdash f : A \to B \quad \Gamma \vdash a : A}{\Gamma \vdash fa : B}$$

We abbreviate $\Gamma \oplus \{x : A\}$ by $\Gamma, x : A$ and write $\Gamma \vdash a_1, \cdots, a_n : A$ to indicate that all of the judgments $\Gamma \vdash a_1 : A, \cdots, \Gamma \vdash a_n : A$ hold.

We reason about terms of the metalanguage with an equational logic of Horn clauses. If the type judgments $\Gamma \vdash a : A$ and $\Gamma \vdash a' : A$ are valid then

$$\Gamma \vdash a = a' : A$$

is an *equation in context* Γ; we usually omit the type and write $\Gamma \vdash a = a'$. A *sequent* is a judgment

$$\Gamma; \Phi \vdash \phi$$

where Γ is a finite set of typed variables, Φ is a finite set of equations in context Γ and ϕ is a single equation in context Γ. The detailed rules for deriving sequents are given in *Appendix A*.

3.2 Interpreting the Nu-Calculus

Types are translated from the nu-calculus to the metalanguage according to:

$$\begin{array}{rcl} [\![o]\!] & = & Bool \\ [\![\nu]\!] & = & Name \\ [\![\sigma \to \sigma']\!] & = & [\![\sigma]\!] \to T[\![\sigma']\!] \end{array}$$

Function types use the constructor T: the application of a function to a value may result in a computation, that is, the generation of new names.

There are two mutually defined schemes that translate nu-calculus expressions into terms of the metalanguage: $|-|$ for expressions in canonical form and $[\![-]\!]$ for general expressions, name and variable contexts. The interpretation is defined by:

Canonical forms:

$$\begin{array}{rcl}
|x| & = & x \\
|n| & = & n \\
|true| & = & tt \\
|false| & = & ff \\
|\lambda x^{\sigma}.M| & = & \lambda x^{[\![\sigma]\!]}.[\![M]\!]
\end{array}$$

Expressions:

$$\begin{array}{rcl}
[\![C]\!] & = & [|C|] \\
[\![if\ B\ then\ M\ else\ M']\!] & = & let\ b \Leftarrow [\![B]\!]\ in\ cond(b, [\![M]\!], [\![M']\!]) \\
[\![N = N']\!] & = & let\ n \Leftarrow [\![N]\!]\ in\ let\ n' \Leftarrow [\![N']\!]\ in\ [eq(n, n')] \\
[\![\nu n.M]\!] & = & let\ n \Leftarrow new\ in\ [\![M]\!] \\
[\![FM]\!] & = & let\ f \Leftarrow [\![F]\!]\ in\ let\ m \Leftarrow [\![M]\!]\ in\ fm
\end{array}$$

Contexts:

$$[\![s, \Gamma]\!] = n_1, \cdots, n_k : Name,\ x_1 : [\![\sigma_1]\!], \cdots, x_n : [\![\sigma_n]\!]$$

where $s = \{n_1, \cdots, n_k\}, \Gamma = \{x_1 : \sigma_1, \cdots, x_n : \sigma_n\}$.

The interpretation respects types and substitution of values; it also treats reduction expressions and contexts in a uniform way.

Lemma 3.1 *For any well-typed nu-calculus expression M, or expression C in canonical form:*

$$\begin{array}{rcl}
s, \Gamma \vdash M : \sigma & \Longleftrightarrow & [\![s, \Gamma]\!] \vdash [\![M]\!] : T[\![\sigma]\!] \\
s, \Gamma \vdash C : \sigma & \Longleftrightarrow & [\![s, \Gamma]\!] \vdash |C| : [\![\sigma]\!]
\end{array}$$

Proof See [6]. □

We define the sets of the interpretation of the well-typed expressions and canonical forms in the nu-calculus at any type σ and for any sets s, Γ of names and typed variables.

$$\begin{array}{rcl}
\mathrm{TExp}_{\sigma}(s, \Gamma) & = & \{[\![M]\!] \mid s, \Gamma \vdash M : \sigma\} \\
\mathrm{TCan}_{\sigma}(s, \Gamma) & = & \{|C| \mid s, \Gamma \vdash C : \sigma, C \text{ canonical}\} \\
\mathrm{TExp}_{\sigma}(s) & = & \mathrm{TExp}_{\sigma}(s, \emptyset) \\
\mathrm{TCan}_{\sigma}(s) & = & \mathrm{TCan}_{\sigma}(s, \emptyset)
\end{array}$$

Lemma 3.2 *If* $M \in \mathrm{Exp}_\sigma(s, \Gamma \oplus \{x : \sigma\})$ *and* $C \in \mathrm{Can}_\sigma(s, \Gamma)$ *then*

$$[\![s, \Gamma]\!] \vdash [\![M[C/x]]\!] = [\![M]\!][|C|/x]$$

in the metalanguage.

Proof See [6]. □

The interpretation of the nu-calculus in the metalanguage is *correct* with respect to its operational semantics: if $s \vdash M \Downarrow_\sigma^{s'} C$ then the terms $[\![M]\!]$ and $[\![\nu s'.C]\!]$ can be proved equal in the metalanguage, under the assumption that all the names in s are distinct. To make this formal we need two abbreviations:

$$(\neq s) = \{eq(n_i, n_j) = ff \mid 1 \leq i \leq j \leq k\}$$

for the assertion that all the names in $s = \{n_1, \cdots, n_k\}$ are distinct, and

$$(let\ s' \Leftarrow \overrightarrow{new}\ \ in\ e) \equiv (let\ n'_1 \Leftarrow new\ in \cdots let\ n'_{k'} \Leftarrow new\ in\ e)$$

which is the expression that assigns new names to all of $s' = \{n'_1, \cdots, n'_{k'}\}$ and then computes e.

Proposition 3.3 (Correctness of Translation) *If* $s \vdash M \Downarrow_\sigma^{s'} C$ *is a valid evaluation judgment in the nu-calculus then*

$$[\![s]\!] \vdash [\![M]\!] = let\ s' \Leftarrow \overrightarrow{new}\ \ in\ [\![C]\!]$$

is provable in the metalanguage.

Proof See [6]. □

Proposition 3.4 (Adequacy of Translation) *If* $M_1, M_2 \in \mathrm{Exp}_\sigma(s, \Gamma)$ *and we can derive*

$$[\![s, \Gamma]\!]; (\neq s) \vdash [\![M_1]\!] = [\![M_2]\!]$$

in the metalanguage, then the expressions are contextually equivalent, i.e., $s, \Gamma \vdash M_1 \approx_\sigma M_2$.

Proof See [6]. □

Chapter 4

Logical Relation for Nu-Calculus

The previous two chapters have provided a formal system for names and described its interpretation in a computational metalanguage. In this chapter, we shall use a technique called *logical relation* to study names. A logical relation is a family of binary relations which are constructed between expressions by induction on the structure of their type, with functions related if they take related arguments to related results [3]. Usually, how good a logical relation depends on a careful choice of the relations on the ground types.

In the first section, we shall introduce briefly the operational logical relation given by Pitts and Stark [5]. Then in *Section 4.2*, we give a formal description for *renaming* and based on that, we define a Kripke logical relation on the interpretation of the nu-calculus in the metalanguage. This Kripke logical relation comes from the logical relation given by Goubault-Larrecq, Lasota and Nowak, which is defined on the whole computational metalanguage [1]. But this Kripke logical relation is not equivalent to the operational logical relation up to observational equivalence. We demonstrate this inequivalence with a counterexample. Then we make some improvement on the Kripke logical relation and define an indexed Kripke logical relation in *Section 4.3*, which is proved to be equivalent to the operational logical relation due to some renaming strategy.

4.1 Operational Logical Relation

This section describes a system of binary logical relations between nu-calculus expressions, based on their operational behavior. We begin by identifying a certain kind of relation between sets of names.

Definition 4.1 (Spans) If s_1 and s_2 are sets of names, then a *span* $R : s_1 \rightleftharpoons s_2$ is an injective partial map from s_1 to s_2. That is, the graph $R \subseteq s_1 \times s_2$ satisfies

$$(n_1 \ R \ n_2) \ \& \ (n_1' \ R \ n_2') \Longrightarrow (n_1 = n_1' \Longleftrightarrow n_2 = n_2').$$

A span can also be represented as a pair of injections $s_1 \leftarrowtail R \rightarrowtail s_2$.

If $R' : s_1' \rightleftharpoons s_2'$ is another span, with s_1' and s_2' disjoint from s_1 and s_2 respectively, then the disjoint union of R and R' is also a span:

$$R \oplus R' : s_1 \oplus s_1' \rightleftharpoons s_2 \oplus s_2'.$$

The domain and codomain of definition for $R : s_1 \rightleftharpoons s_2$ are defined by :

$$\begin{array}{rcl} \mathrm{dom}(R) & = & \{n_1 \in s_1 \mid \exists n_2 \in s_2 \ (n_1 \ R \ n_2)\} \\ \mathrm{cod}(R) & = & \{n_2 \in s_2 \mid \exists n_1 \in s_1 \ (n_1 \ R \ n_2)\} \end{array}$$

The intuition behind spans is that they should capture the use that nu-calculus expressions make of public and private names. So if $R : s_1 \rightleftharpoons s_2$, then the bijection between $\mathrm{dom}(R) \subseteq s_1$ and $\mathrm{cod}(R) \subseteq s_2$ represents matching use of "visible" names, while the remaining elements not in the graph of R are "unseen" names.

Definition 4.2 (Operational logical relation) If $R : s_1 \rightleftharpoons s_2$ is a span then the relations

$$\begin{array}{rcl} R_\sigma^{can} & \subseteq & \mathrm{Can}_\sigma(s_1) \times \mathrm{Can}_\sigma(s_2) \\ R_\sigma^{exp} & \subseteq & \mathrm{Exp}_\sigma(s_1) \times \mathrm{Exp}_\sigma(s_2) \end{array}$$

are defined by induction over the structure of the type σ, according to:

$$b_1 \ R_o^{can} \ b_2 \iff b_1 = b_2$$

$$n_1 \ R_\nu^{can} \ n_2 \iff n_1 \ R \ n_2$$

$$
\begin{aligned}
&(\lambda x^\sigma.M_1)\ R^{can}_{\sigma\to\sigma'}\ (\lambda x^\sigma.M_2) \iff\\
&\qquad \forall R' : s'_1 \rightleftharpoons s'_2, C_1 \in Can_\sigma(s_1 \oplus s'_1), C_2 \in Can_\sigma(s_2 \oplus s'_2)\\
&\qquad\quad (C_1\ (R \oplus R')^{can}_\sigma\ C_2 \Longrightarrow M_1[C_1/x]\ (R \oplus R')^{exp}_{\sigma'}\ M_2[C_2/x])\\
&M_1\ R^{exp}_\sigma\ M_2 \iff\\
&\qquad \exists R' : s'_1 \rightleftharpoons s'_2, C_1 \in Can_\sigma(s_1 \oplus s'_1), C_2 \in Can_\sigma(s_2 \oplus s'_2)\\
&\qquad\quad (s_1 \vdash M_1 \Downarrow^{s'_1}_\sigma C_1\ \&\ s_2 \vdash M_2 \Downarrow^{s'_2}_\sigma C_2\ \&\ C_1\ (R \oplus R')^{can}_\sigma\ C_2)
\end{aligned}
$$

The relation R^{can}_σ and R^{exp}_σ coincide on canonical forms, and we may write them as R_σ indiscriminately.

Lemma 4.1 *If $R : s_1 \rightleftharpoons s_2$ and $M_1 \in \mathrm{Exp}_\sigma(s_1, \Gamma)$, $M_2 \in \mathrm{Exp}_\sigma(s_2, \Gamma)$ then*

$$\Gamma \vdash M_1\ R_\sigma\ M_2 \iff \Gamma \oplus \Gamma' \vdash M_1\ (R \oplus R')_\sigma\ M_2$$

for any Γ' and $R' : s'_1 \rightleftharpoons s'_2$.

Proof See [6]. □

With this logical relation we can deduce contextual equivalence, which means that if two terms are logically related, then they are contextually equivalent [5]. For example, the two contextually equivalent terms in *Example 2.3* can be related with the operational logical relation. The only possible span $R : \{n\} \rightleftharpoons \{\}$ is the empty relation, and with this we have that

$$(\lambda x^\nu.(x = n))\ R^{can}_{\nu\to o}\ (\lambda x^\nu.false)$$

because both functions take R-related names to $false$. Then

$$(\nu n.\lambda x^\nu.(x = n))\ R^{exp}_{\nu\to o}\ (\lambda x^\nu.false).$$

4.2 Kripke Logical Relation

A general Kripke logical relation has a relation for each type and a 'possible world' instead of just having a relation for each type [3]. The set of possible worlds is partially-ordered. Informally, the relationship between elements of type σ at different worlds is that every element of σ at w will continue to be an element of σ at every possible $w' \geq w$. In the case of names, we take the name sets as the possible worlds.

In this section, we shall introduce a particular Kripke logical relation for the nu-calculus. This logical relation is not defined directly on the nu-calculus. It comes from the logical relation defined on the computational metalanguage, which is given in [1]. But here what we are interested in is the nu-calculus, so our Kripke logical relation is just defined on the interpretation of the nu-calculus in the metalanguage. Before giving the definition, we shall first introduce some notions.

In *Section 3.2*, we have seen that in the metalanguage, a computation can be regarded as a procedure with some names generated, and a trivial computation is a procedure with no name generated. According to the interpretation of the nu-calculus, a canonical expression will be translated into a trivial computation

$$[\![C]\!] = [|C|],$$

while by the *Proposition 3.3*, the interpretation of a usual expression can be written as

$$[\![M]\!] = let\ s' \Leftarrow \overrightarrow{new}\ \ in\ [\![C]\!]$$

if $s \vdash M \Downarrow_{\sigma}^{s'} C$. In order to simplify and unify the notions, we introduce some syntactic sugar for these computations. If $s \vdash M \Downarrow_{\sigma}^{s'} C$, we write $[\![M]\!]$ as $\bar{\nu}s'.|C|$ and for a canonical form $s \vdash C : \sigma$, we write $[\![C]\!]$ as $\bar{\nu}\emptyset.|C|$. Then by the *Lemma 3.1* we have

$$\begin{array}{lcl} s, \Gamma \vdash C : \sigma & \Longleftrightarrow & [\![s, \Gamma]\!] \vdash |C| : [\![\sigma]\!] \\ s, \Gamma \vdash C : \sigma & \Longleftrightarrow & [\![s, \Gamma]\!] \vdash \bar{\nu}\emptyset.|C| : T[\![\sigma]\!] \end{array}$$

Intuitively, the idea of this particular Kripke logical relation comes from the *renaming* in programming languages. It is a natural and useful technique to manage name conflict. The key point is that distinct names must be renamed by distinct new names, and a name should not be renamed to two different names. An injection is the best choice for formalizing such an action. When we regard renaming as an injection, usually we need not specify its codomain, because in practice, the choice of the new names are at random.

Definition 4.3 (Renaming function) If s and s' are two sets of names with $|s| \leq |s'|$, a *renaming function* i on s is an injection from s to s', that is

$$\forall n_1, n_2 \in s(n_1 = n_2 \Longleftrightarrow i(n_1) = i(n_2)).$$

Note that here $|s|$ stands for the cardinal of the corresponding set, and so does $|s'|$.

When a renaming function i on s is applied to a name which is not in its domain, the result is just the name itself, i.e.,

$$i(n) = n, \text{ if } n \notin s.$$

A renaming function i on s can be applied to a random name set s' and even to a term in the nu-calculus. If $M \in \mathrm{Exp}_\sigma(s', \Gamma)$, we write

$$\begin{array}{rcl} i(s) & = & \{i(n) \mid n \in s'\} \\ i(M) & = & M[i(n_1)/n_1, \cdots, i(n_k)/n_k], \text{where } \mathrm{FN}(M) = \{n_1, \cdots, n_k\}. \end{array}$$

Usually, for a renamed term, it's natural to get back to the original term by the counter function of the renaming function. If i is a renaming function on s, we write i^- for the renaming function on $i(s)$ which satisfy

$$\forall n \in i(s)\ (i^-(n) = m, \text{ where } m \in s \ \&\ i(m) = n).$$

Moreover, i^- is a bijection between s and $i(s)$.

In particular, the identity function on a random name set s can be regarded as a renaming function. We write this special renaming function as id^s.

Lemma 4.2 *If i is a renaming function on a name set s and $s \vdash M : \sigma$, then $i(s) \vdash i(M) : \sigma$.*

Proof The result follows by induction on the structure of M. ☐

Lemma 4.3 *If i is a renaming function on a name set s and $M \in \mathrm{Exp}_\sigma(s)$, $C \in \mathrm{Can}_\sigma(s)$, $s \vdash M \Downarrow_\sigma^{s'} C$, then there exists some renaming function h on s' such that $i(s) \vdash i(M) \Downarrow_\sigma^{h(s')} h(i(C))$.*

Proof By induction on the structure of the derivation of the evaluation judgment in the nu-calculus. ☐

Lemma 4.4 *Suppose there are two terms $M_1 \in \mathrm{Exp}_\sigma(s_1)$, $M_2 \in \mathrm{Exp}_\sigma(s_2)$ and a span $R : s_1 \rightleftharpoons s_2$ such that $M_1\ R_\sigma\ M_2$. If i_1 and i_2 are two renaming functions on s_1 and s_2 respectively, there is a span $R' : i_1(s_1) \rightleftharpoons i_2(s_2)$ defined by*

$$\forall n_1 \in s_1, n_2 \in s_2 (n_1\ R\ n_2 \Longleftrightarrow i_1(n_1)\ R'\ i_2(n_2)) \tag{4.1}$$

such that $i_1(M_1)\ R'_\sigma\ i_2(M_2)$.

Proof By induction on the structure of the type σ. $\square$

Definition 4.4 (Kripke Logical Relation) For each type σ we define a family of binary relation

$$\begin{array}{rcl} \mathbb{R}^s_{[\![\sigma]\!]} & \subseteq & \mathrm{TCan}_\sigma(s) \times \mathrm{TCan}_\sigma(s) \\ \mathbb{R}^s_{T[\![\sigma]\!]} & \subseteq & \mathrm{TExp}_\sigma(s) \times \mathrm{TExp}_\sigma(s) \end{array}$$

by induction on the type structure in the metalanguage, according to

$$|b_1| \ \mathbb{R}^s_{Bool} \ |b_2| \iff b_1 = b_2,$$

$$|n_1| \ \mathbb{R}^s_{Name} \ |n_2| \iff n_1 = n_2,$$

$$\begin{array}{l} [\![M_1]\!] \ \mathbb{R}^s_{T[\![\sigma]\!]} \ [\![M_2]\!] \iff \\ \qquad \exists s', i_1 : s_1 \to s', i_2 : s_2 \to s' \ (|i_1(C_1)| \ \mathbb{R}^{s \oplus s'}_{[\![\sigma]\!]} \ |i_2(C_2)|) \\ \qquad\quad \text{where } [\![M_1]\!] \equiv \bar{\nu} s_1.|C_1| \text{ and } [\![M_2]\!] \equiv \bar{\nu} s_2.|C_2|, \end{array}$$

$$\begin{array}{l} |\lambda x^\sigma.M_1| \ \mathbb{R}^s_{[\![\sigma]\!] \to T[\![\sigma']\!]} \ |\lambda x^\sigma.M_2| \iff \\ \qquad \forall s', i : s \to s' \ (|C_1| \ \mathbb{R}^{s'}_{[\![\sigma]\!]} \ |C_2| \implies [\![i(M_1)[C_1/x]]\!] \ \mathbb{R}^{s'}_{T[\![\sigma']\!]} \ [\![i(M_2)[C_2/x]]\!]), \end{array}$$

where i, i_1, i_2 are renaming functions.

The definition for function types just defines a condition called *comprehension*, and the following proposition shows another condition *monotonicity*. They are two general conditions which should be satisfied by any Kripke logical relation.

Proposition 4.5 (Monotonicity) *Suppose that $M_1, M_2 \in \mathrm{Exp}_\sigma(s)$ and s' is a name set where $s \subseteq s'$. Then*

$$[\![M_1]\!] \ \mathbb{R}^s_{T[\![\sigma]\!]} \ [\![M_2]\!] \implies [\![M_1]\!] \ \mathbb{R}^{s'}_{T[\![\sigma]\!]} \ [\![M_2]\!].$$

Proof From the definition of $\mathbb{R}^s_{T[\![\sigma]\!]}$, we can easily show that for any two canonical forms $C_1, C_2 \in \mathrm{Can}_\sigma(s)$,

$$[\![C_1]\!] \ \mathbb{R}^s_{T[\![\sigma]\!]} \ [\![C_2]\!] \iff |C_1| \ \mathbb{R}^s_{[\![\sigma]\!]} \ |C_2|,$$

so we can make the proof by induction on the type structure in the interpretation of the nu-calculus.

For type *Bool* and *Name*, the result is obvious.

For type $T[\![\sigma]\!]$, if $[\![M_1]\!]\ \mathbb{R}^s_{T[\![\sigma]\!]}\ [\![M_2]\!]$, by the definition and renaming, we can find a name set $s'' \pitchfork s'$ and two renaming functions $i_1 : s_1 \to s''$, $i_2 : s_2 \to s''$ such that $|i_1(C_1)|\ \mathbb{R}^{s \oplus s''}_{[\![\sigma]\!]}\ |i_2(C_2)|$. Now, by the induction we have $|i_1(C_1)|\ \mathbb{R}^{s' \oplus s''}_{[\![\sigma]\!]}\ |i_2(C_2)|$ and then, $[\![M_1]\!]\ \mathbb{R}^{s'}_{T[\![\sigma]\!]}\ [\![M_2]\!]$.

In the case of type $[\![\sigma]\!] \to T[\![\sigma']\!]$, for any name set s'' and renaming function $i : s' \to s''$, since $s \subseteq s'$, i can also be regarded as a renaming function from s to s''. Because $|\lambda x^\sigma.M_1|\ \mathbb{R}^s_{[\![\sigma]\!] \to T[\![\sigma']\!]}\ |\lambda x^\sigma.M_2|$, we have

$$|C_1|\ \mathbb{R}^{s''}_{[\![\sigma]\!]}\ |C_2| \Longrightarrow [\![i(M_1)[C_1/x]]\!]\ \mathbb{R}^{s''}_{T[\![\sigma']\!]}\ [\![i(M_2)[C_2/x]]\!],$$

then by the definition, $|\lambda x^\sigma.M_1|\ \mathbb{R}^{s'}_{[\![\sigma]\!] \to T[\![\sigma']\!]}\ |\lambda x^\sigma.M_2|$. $\square$

4.3 Indexed Kripke Logical Relation

The Kripke logical relation defined in last section is not so strong as the operational logical relation up to contextual equivalence. There are some contextually equivalent terms in the nu-calculus which can be deduced by the operational logical relation but cannot by the Kripke logical relation.

Consider the two terms in *Example 2.3*. We can write the interpretations of the two terms as

$$\begin{aligned}[\![\nu n.\lambda x^\nu.(x = n)]\!] &\equiv \bar{\nu}\{n\}.\lambda x^{Name}.\bar{\nu}\emptyset.eq(x, n)\\ [\![\lambda x^\nu.false]\!] &\equiv \bar{\nu}\emptyset.\lambda x^{Name}.\bar{\nu}\emptyset.ff,\end{aligned}$$

which are both of type $T(Name \to TBool)$. If these two terms can be related by the Kripke logical relation, say

$$\bar{\nu}\{n\}.\lambda x^{Name}.\bar{\nu}\emptyset.eq(x, n)\ \mathbb{R}^{\emptyset}_{T(Name \to TBool)}\ \bar{\nu}\emptyset.\lambda x^{Name}.\bar{\nu}\emptyset.ff,$$

by *Definition 4.4*, we first have a set $s \supseteq \{n\}$ such that

$$\lambda x^{Name}.\bar{\nu}\emptyset.eq(x, n)\ \mathbb{R}^s_{Name \to TBool}\ \lambda x^{Name}.\bar{\nu}\emptyset.ff,$$

then because $|n|\ \mathbb{R}^s_{Name}\ |n|$, we get

$$\bar{\nu}\emptyset.eq(n, n)\ \mathbb{R}^s_{TBool}\ \bar{\nu}\emptyset.ff$$

when the two functions are both applied to $|n|$, which immediately leads to $tt\ \mathbb{R}^s_{Bool}\ ff$. This is a contradiction to the definition.

Comparing this relation with the operational logical relation, the spirit of identifying 'visible' names and 'unseen' names, or say, public names and private names, which can be shown by the span, doesn't exist in Kripke logical relation any more. In Kripke logical relation, the relation for the name type is an identity relation and every name can be related to itself, however, not every name is a 'visible' name. In order to distinguish between 'visible' names and 'unseen' names as in the operational logical relation, a possible way is to add some index to the relation.

Definition 4.5 (Indexed Kripke Logical Relation) For each type σ we define a family of binary relation

$$\begin{aligned} \mathcal{R}^{s,w}_{[\![\sigma]\!]} &\subseteq \mathrm{TCan}_\sigma(s) \times \mathrm{TCan}_\sigma(s) \\ \mathcal{R}^{s,w}_{T[\![\sigma]\!]} &\subseteq \mathrm{TExp}_\sigma(s) \times \mathrm{TExp}_\sigma(s), \end{aligned}$$

where s and w are sets of names and $w \subseteq s$, by induction on the type structure in the metalanguage, according to

$$|b_1| \ \mathcal{R}^{s,w}_{Bool} \ |b_2| \iff b_1 = b_2$$

$$|n_1| \ \mathcal{R}^{s,w}_{Name} \ |n_2| \iff n_1 = n_2 \ \& \ n_1, n_2 \in w$$

$$\begin{aligned} &[\![M_1]\!] \ \mathcal{R}^{s,w}_{T[\![\sigma]\!]} \ [\![M_2]\!] \iff \\ &\qquad \exists s', i_1 : s_1 \to s', i_2 : s_2 \to s', w' \subseteq s' \ (|i_1(C_1)| \ \mathcal{R}^{s \oplus s', w \oplus w'}_{[\![\sigma]\!]} \ |i_2(C_2)|) \\ &\qquad\quad \text{where } [\![M_1]\!] \equiv \bar{\nu} s_1.|C_1| \text{ and } [\![M_2]\!] \equiv \bar{\nu} s_2.|C_2|, \end{aligned}$$

$$\begin{aligned} &\lambda x^{[\![\sigma]\!]}.[\![M_1]\!] \ \mathcal{R}^{s,w}_{[\![\sigma]\!] \to T[\![\sigma']\!]} \ \lambda x^{[\![\sigma]\!]}.[\![M_2]\!] \iff \\ &\qquad \forall s', i : s \to s', w' \subseteq s' - i(s) \\ &\qquad (|C_1| \ \mathcal{R}^{s', i(w) \oplus w'}_{[\![\sigma]\!]} \ |C_2| \Longrightarrow [\![i(M_1)[C_1/x]]\!] \ \mathcal{R}^{s', i(w) \oplus w'}_{T[\![\sigma']\!]} \ [\![i(M_2)[C_2/x]]\!]) \end{aligned}$$

where i, i_1, i_2 are renaming functions.

In this definition, s still represents the 'possible world'. Intuitively, the index w just stands for the public names or 'visible' names. Up to contextual equivalence, if two terms are contextually equivalent, all of their observational actions and fetures should be same, including the public names, which means those names known to other terms.

With this logical relation, we can relate the two terms in *Example 2.3* again,

$$\bar{\nu}\{n\}.\lambda x^{Name}.\bar{\nu}\emptyset.eq(x, n) \ \mathcal{R}^{\emptyset,\emptyset}_{T(Name \to TBool)} \ \bar{\nu}\emptyset.\lambda x^{Name}.\bar{\nu}\emptyset.ff.$$

By the definition of the indexed Kripke logical relation, to prove the above relation we need

$$\lambda x^{Name}.\bar{\nu}\emptyset.eq(x,n)\ \mathcal{R}^{s,\emptyset}_{Name\to TBool}\ \lambda x^{Name}.\bar{\nu}\emptyset.\mathit{ff}$$

with $n \in s$. While according to the definition for functions types, for any name set s', renaming function $i : s \to s'$ and index $w' \subseteq s' - i(s)$, there is no name in $i(s)$ that can be related to itself with the index w', i.e.,

$$\forall m \in s'\ (|m|\ \mathcal{R}^{s',w'}_{Name}\ |m| \Longrightarrow m \notin i(s)),$$

which means that, when a name m that can be related to itself with the index w' is applied to the function $i(\lambda x^{Name}.\bar{\nu}\emptyset.eq(x,n))$, the result term $\bar{\nu}\emptyset.eq(m,i(n))$ will continue to result in $\bar{\nu}\emptyset.\mathit{ff}$. That is

$$\forall m \in s'\ (|m|\mathcal{R}^{s',w'}_{Name}\ |m| \Longrightarrow i(\bar{\nu}\emptyset.eq(x,n))[|m|/x]\ \mathcal{R}^{s',w'}_{TBool}\ i(\bar{\nu}\emptyset.\mathit{ff})[|m|/x],$$

and then, we get the result.

Proposition 4.6 (Monotonicity) *Suppose that $M_1, M_2 \in \mathrm{Exp}_\sigma(s)$ and s' is a name set where $s \subseteq s'$. Then*

$$[\![M_1]\!]\ \mathcal{R}^{s,w}_{T[\![\sigma]\!]}\ [\![M_2]\!] \Longrightarrow [\![M_1]\!]\ \mathcal{R}^{s',w}_{T[\![\sigma]\!]}\ [\![M_2]\!].$$

Proof Also by induction on the type structure in the interpretation of the nu-calculus. □

The following theorem shows that, with some renaming strategy, such an indexed Kripke logical relation is equivalent to Pitts and Stark's operational logical relation.

Theorem 4.7 (i) *Suppose there is an index w and two expressions $M_1, M_2 \in \mathrm{TExp}_\sigma(s)$, if $[\![M_1]\!]\ \mathcal{R}^{s,w}_{T[\![\sigma]\!]}\ [\![M_2]\!]$, then there is a span $R : \mathrm{FN}(M_1) \rightleftharpoons \mathrm{FN}(M_2)$ such that $M_1\ R^{exp}_\sigma\ M_2$. More precisely, the span R is defined by*

$$n_1\ R\ n_2 \Longleftrightarrow n_1 = n_2\ \&\ n_1, n_2 \in w. \tag{4.2}$$

(ii) *Suppose there is a span $R : s_1 \rightleftharpoons s_2$ and two expressions $M_1 \in \mathrm{Exp}_\sigma(s_1)$, $M_2 \in \mathrm{Exp}_\sigma(s_2)$,if $M_1\ R_\sigma\ M_2$, then for any two renaming functions i_1 on s_1 and i_2 on s_2 which satisfy*

$$\forall n_1 \in s_1, n_2 \in s_2\ (n_1\ R\ n_2 \Rightarrow i_1(n_1) = i_2(n_2)), \tag{4.3}$$

$[\![i_1(M_1)]\!]\ \mathcal{R}^{s,w}_{T[\![\sigma]\!]}\ [\![i_2(M_2)]\!]$, *where $s = i_1(s_1) \cup i_2(s_2)$ and*

$$w = \{i_1(n) \mid n \in s_1\ \&\ \exists n' \in s_2(n\ R\ n')\}. \tag{4.4}$$

Proof This theorem does not mention the canonical forms, but we have

$$[\![C_1]\!]\ \mathcal{R}^{s,w}_{T[\![\sigma]\!]}\ [\![C_2]\!] \Longleftrightarrow |C_1|\ \mathcal{R}^{s,w}_{[\![\sigma]\!]}\ |C_2|.$$

for any $C_1, C_2 \in \mathrm{Can}_\sigma(s)$, and also the operational logical relation R^{can}_σ and R^{exp}_σ coincide on canonical forms, so we can make the proof by induction on the type structure in the interpretation of nu-calculus.

• Type *Bool*, corresponding to the canonical forms of type o in the nu-calculus. The results of both (i) and (ii) are obvious.

• Type *Name*, corresponding to the canonical forms of type ν in the nu-calculus.

(i) If $|n_1|\ \mathcal{R}^{s,w}_{Name}\ |n_2|$, by the definition of $\mathbb{R}$ we have $n_1 = n_2$ and $n_1, n_2 \in w$. With the span $R : \{n_1\} \rightleftharpoons \{n_2\}$ defined by (4.2) there holds $n_1\ R^{can}_\nu\ n_2$.

(ii) If $n_1\ R^{can}_\nu\ n_2$ with $n_1\ R\ n_2$, for any two renaming function i_1 on $\{n_1\}$ and i_2 on $\{n_2\}$ satisfying (4.3), we have $i_1(n_1) = i_2(n_2)$, so that $i_1(\{n_1\}) = i_2(\{n_2\})$. Let $s = w = i_1(\{n_1\})$, then $|i_1(n_1)|\ \mathcal{R}^{s,w}_{Name}\ |i_2(n_2)|$.

• Type $T[\![\sigma]\!]$, corresponding to the expressions of type σ in the nu-calculus.

(i) If $[\![M_1]\!]\ \mathcal{R}^{s,w}_{T[\![\sigma]\!]}\ [\![M_2]\!]$ and $s \vdash M_1 \Downarrow^{s_1}_\sigma C_1, M_2 \Downarrow^{s_2}_\sigma C_2$, by definition we can find two name sets s', w' with $w' \subseteq s'$ and two renaming functions $i_1 : s_1 \to s', i_2 : s_2 \to s'$ such that $|i_1(C_1)|\ \mathcal{R}^{s\oplus s', w\oplus w'}_{[\![\sigma]\!]}\ |i_2(C_2)|$. Then, by induction hypothesis, we have a span $S : \mathrm{FN}(i_1(C_1)) \rightleftharpoons \mathrm{FN}(i_2(C_2))$ defined by

$$n_1\ S\ n_2 \Longleftrightarrow n_1 = n_2\ \&\ n_1, n_2 \in w \oplus w'$$

such that $i_1(C_1)\ S^{can}_\sigma\ i_2(C_2)$. Define two spans $R' : i_1(s_1) \rightleftharpoons i_2(s_2)$ as

$$n_1\ R'\ n_2 \Longleftrightarrow n_1 = n_2\ \&\ n_1, n_2 \in w'$$

and $R : \mathrm{FN}(M_1) \rightleftharpoons \mathrm{FN}(M_2)$ as

$$n_1\ R\ n_2 \Longleftrightarrow n_1 = n_2\ \&\ n_1, n_2 \in w,$$

and then $i_1(C_1)\ (R \oplus R')^{can}_\sigma\ i_2(C_2)$. Moreover, by *Lemma 4.3* we have $\mathrm{FN}(M_1) \vdash M_1 \Downarrow^{i_1(s_1)}_\sigma i_1(C_1)$, $\mathrm{FN}(M_2) \vdash M_2 \Downarrow^{i_2(s_2)}_\sigma i_2(C_2)$, and by the definition of operational logical relation, $M_1\ R^{exp}_\sigma\ M_2$.

(ii) There is a span $R : s_1 \rightleftharpoons s_2$ such that $M_1\ R^{exp}_\sigma\ M_2$, and by the definition of R^{exp}_σ we have a span $R' : s'_1 \rightleftharpoons s'_2$ and two canonical forms C_1, C_2 such that

$$s_1 \vdash M_1 \Downarrow^{s'_1}_\sigma C_1\ \&\ s_2 \vdash M_2 \Downarrow^{s'_2}_\sigma C_2\ \&\ C_1\ (R \oplus R')^{can}_\sigma\ C_2.$$

For any two renaming functions i_1 on s_1 and i_2 on s_2 satisfying (4.3) with the span $R : s_1 \rightleftharpoons s_2$ and any two renaming functions i'_1 on s'_1 and i'_2 on s'_2 satisfying (4.3) with the span $R' : s'_1 \rightleftharpoons s'_2$, we can define two renaming functions j_1 on $s_1 \oplus s'_1$ and j_2 on $s_2 \oplus s'_2$ with

$$j_k(n) = \begin{cases} i_k(n), & n \in s_k \\ i'_k(n), & n \in s'_k \end{cases} \qquad (k = 1, 2).$$

It's easy to show that j_1 and j_2 satisfy (4.3) with the span $R \oplus R'$. Then, by the induction we have $|j_1(C_1)|\ \mathcal{R}^{s,w}_{[\![\sigma]\!]}\ |j_2(C_2)|$ with

$$s = j_1(s_1 \oplus s'_1) \cup j_2(s_2 \oplus s'_2) = s' \oplus (i'_1(s'_1) \cup i'_2(s'_2))$$

and

$$\begin{aligned} w &= \{j_1(n) \mid n \in s_1 \oplus s'_1\ \&\ \exists m \in s_2 \oplus s'_2(n(R \oplus R')m)\} \\ &= w' \oplus \{i'_1(n) \mid n \in s'_1\ \&\ \exists m \in s'_2(n\ R'\ m)\} \end{aligned}$$

$$s' = i_1(s_1) \cup i_2(s_2), \qquad w' = \{i_1(n) \mid n \in s_1\ \&\ \exists m \in s_2(n\ R\ m)\}.$$

According to *Lemma 4.3*, we have

$$[\![i_1(M_1)]\!] = \bar{\nu}(i'_1(s'_1)).|j_1(C_1)|, \qquad [\![i_2(M_2)]\!] = \bar{\nu}(i'_2(s'_2)).|j_2(C_2)|,$$

then by the definition of $\mathcal{R}^{s,w}_{T[\![\sigma]\!]}$, we have $[\![i_1(M_1)]\!]\ \mathcal{R}^{s',w'}_{T[\![\sigma]\!]}\ [\![i_2(M_2)]\!]$.

• Type $[\![\sigma]\!] \to T[\![\sigma']\!]$, corresponding to the canonical forms of type $\sigma \to \sigma'$ in the nu-calculus.

(i) For a canonical form $\lambda x^\sigma.M$, because in the nu calculus, all the names are chosen distinctly from those variables, it's obvious that $\mathrm{FN}(\lambda x^\sigma.M) = \mathrm{FN}(M)$. For two canonical forms $|\lambda x^\sigma.M_1|\ \mathcal{R}^{s,w}_{[\![\sigma]\!] \to T[\![\sigma']\!]}\ |\lambda x^\sigma.M_2|$, let $s_1 = \mathrm{FN}(M_1)$, $s_2 = \mathrm{FN}(M_2)$, and define a span $R : s_1 \rightleftharpoons s_2$ by (4.2).Then for any span $R' : s'_1 \rightleftharpoons s'_2$ and any two canonical forms $C_1\ (R \oplus R')_\sigma\ C_2$, we can define two renaming functions i_1 on s'_1 and i_2 on s'_2 by (4.3) with span R'. Let $j_1 = id^{s_1} \oplus i_1$ be a renaming function on $s_1 \oplus s'_1$ and $j_2 = id^{s_2} \oplus i_2$ on $s_2 \oplus s'_2$. It's obvious that j_1 and j_2 satisfy (4.3) with span $R \oplus R'$. Let $s' = i_1(s'_1) \cup i_2(s'_2)$ and w' be defined by (4.4) with span R', then by the induction of (ii) we have $|j_1(C_1)|\ \mathcal{R}^{s'',w''}_{[\![\sigma]\!]}\ |j_2(C_2)|$ with

$$s'' = (\mathrm{FN}(M_1) \cup \mathrm{FN}(M_2)) \oplus s' \text{ and } w'' = w \oplus w'.$$

It's easy to see that $s'' \subseteq s \oplus s'$, so according to *Proposition 4.6* we have $|j_1(C_1)|\ \mathcal{R}^{s\oplus s',w\oplus w'}_{[\![\sigma]\!]}\ |j_2(C_2)|$, and subsequently, by the definition of $\mathcal{R}^{s,w}_{[\![\sigma]\!] \to T[\![\sigma']\!]}$,

$$[\![M_1[j_1(C_1)/x]]\!]\ \mathcal{R}^{s\oplus s',w\oplus w'}_{T[\![\sigma']\!]}\ [\![M_2[j_2(C_2)/x]]\!].$$

Now, by the induction of (ii), there is a span $Q : s_1 \oplus i_1(s'_1) \rightleftharpoons s_2 \oplus i_2(s'_2)$ defined by (4.2) such that $M_1[j_1(C_1)/x]\ Q_{\sigma'}\ M_2[j_2(C_2)/x]$, and by *Lemma 4.4*, taking the two renaming functions just as j_1^- and j_2^-, we have $M_1[C_1/x]\ Q_{\sigma'}\ M_2[C_2/x]$ with a span $Q' : s_1 \oplus s'_1 \rightleftharpoons s_2 \oplus s'_2$ defined by (4.1). According to the definition of R, R' and Q', $Q' = R \oplus R'$, then, by the definition of the operational logical relation, $\lambda x^\sigma.M_1\ R_{\sigma\to\sigma'}\ \lambda x^\sigma.M_2$.

(ii) There is a span $R : s_1 \rightleftharpoons s_2$ such that $\lambda x^\sigma.M_1\ R_{\sigma\to\sigma'}\ \lambda x^\sigma.M_2$. For any two renaming function i_1 on s_1 and i_2 on s_2 satisfying (4.3), let $s = i_1(s_1) \cup i_2(s_2)$ and w be defined by 4.4 with span R. For any renaming function $i : s \to s'$ and a name set $w' \subseteq s' - i(s)$, assuming $|C_1|\ \mathcal{R}_\sigma^{s',i(w)\oplus w'}\ |C_2|$, by the induction of (i), we have a span $Q : \mathrm{FN}(C_1) \rightleftharpoons \mathrm{FN}(C_2)$ defined by (4.2) such that $C_1\ Q_\sigma\ C_2$. And by *Lemma 4.4*, we can define a span $R' : i(i_1(s_1)) \rightleftharpoons i(i_2(s_2))$ such that $\lambda x^\sigma.i(i_1(M_1))\ R'_{\sigma\to\sigma'}\ \lambda x^\sigma.i(i_2(M_2))$. Then by (4.2) the definition of span R' can be written as

$$\forall n_1 \in i(i_1(s_1)), n_2 \in i(i_2(s_2))\ (n_1\ R'\ n_2 \Longleftrightarrow n_1 = n_2).$$

Let $s'' = s' - i(s)$ and define a span $R'' : s'' \rightleftharpoons s''$ as

$$n_1\ R''\ n_2 \Longleftrightarrow n_1 = n_2\ \&\ n_1, n_2 \in w',$$

then according to *Lemma 4.1*, $C_1\ (R' \oplus R'')_\sigma\ C_2$, and by the definition of $\lambda x^\sigma.i(i_1(M_1))\ R'_{\sigma\to\sigma'}\ \lambda x^\sigma.i(i_2(M_2))$, we have

$$i(i_1(M_1))[C_1/x]\ (R' \oplus R'')_{\sigma'}\ i(i_2(M_2))[C_2/x].$$

By the induction of (ii), just taking the identity function as the renaming functions, we have

$$[\![i(i_1(M_1))[C_1/x]]\!]\ \mathcal{R}_{T[\![\sigma']\!]}^{s',i(w)\oplus w'}\ [\![i(i_2(M_2))[C_2/x]]\!],$$

and subsequently, $|\lambda x^\sigma.i_1(M_1)|\ \mathcal{R}_{[\![\sigma]\!]\to T[\![\sigma']\!]}^{s,w}\ |\lambda x^\sigma.i_2(M_2)|$, by the definition of $\mathcal{R}_{[\![\sigma]\!]\to T[\![\sigma']\!]}^{s,w}$. □

This theorem not only shows the equivalence up to renaming between the operational logical relation and the indexed Kripke logical relation, but also presents a way of relating two nu-calculus terms in one logical relations when they are already related by the other one.

Chapter 5

Conclusion

We have shown in this report that in the computational metalanguage, we can define a logical relation for the nu-calculus which is equivalent to the operational logical relation due to renaming. According to Pitts and Stark's study on the operational logical relation, all the logical relations we have discussed are not complete up to contextual equivalence. It is still possible to improve these logical relations, but it is not clear yet whether or not there is a complete logical relation for the nu-calculus. It seems impossible to find such a logical relation in an operational way or just on the interpretation of the nu-calculus in the metalanguage, but it's worth trying in some more abstract way, for example, with a categorical model. Moreover, the decidability is a big problem when we try to find a complete logical relation. We have to find a balance.

In *Chapter 1*, we mentioned Sumii and Pierce's cryptographic lambda-calculus, which is a nice formal language to describe the cryptographic protocols, but the logical relation they define on this language is also constructed in an operational way [7]. Since logical relations have proved useful for contextual equivalence, our future study will be concentrated on studying the logical relations on the cryptographic lambda-calculus, including the completeness and decidability. It is also necessary to define a denotational semantics for the cryptographic lambda-calculus, which will provide some different ways to construct logical relations on the language.

Appendix A

Reasoning Rules in the Metalanguage

Stark gave a detailed set of rules for deriving sequents. Part *A.1* gives the usual rules for Horn clauses and euqational logic, congruence rules for all the term forming operations and β, η axioms at function types. Part *A.2* contains rules particular to the computational metalanguage.

A.1

Horn clauses:

$$\frac{}{\Gamma; \Phi \vdash \phi}(\phi \in \Phi) \qquad \frac{\Gamma; \Phi \vdash \phi \qquad \Gamma; \Psi, \phi \vdash \psi}{\Gamma; \Phi \cup \Psi \vdash \psi}$$

Equality:

$$\frac{\Gamma \vdash a : A}{\Gamma \vdash a = a} \qquad \frac{\Gamma \vdash a, a' : A}{\Gamma; a = a' \vdash a' = a} \qquad \frac{\Gamma \vdash a_1, a_2, a_3 : A}{\Gamma; a_1 = a_2, a_2 = a_3 \vdash a_1 = a_3}$$

Congruence:

$$\frac{}{\Gamma \vdash x = x}(x : A \in \Gamma) \quad \frac{}{\Gamma \vdash \mathit{new} = \mathit{new}} \quad \frac{}{\Gamma \vdash \mathit{tt} = \mathit{tt}} \quad \frac{}{\Gamma \vdash \mathit{ff} = \mathit{ff}}$$

$$\frac{\Gamma \vdash b, b' : \mathit{Bool} \qquad \Gamma \vdash a_1, a_1', a_2, a_2' : A}{\Gamma; b = b', a_1 = a_1', a_2 = a_2' \vdash \mathit{cond}(b, a_1, a_2) = \mathit{cond}(b', a_1', a_2')}$$

$$\frac{\Gamma \vdash n_1, n_1', n_2, n_2' : \mathit{Name}}{\Gamma; n_1 = n_1', n_2 = n_2' \vdash \mathit{eq}(n_1, n_2) = \mathit{eq}(n_1' = n_2')}$$

$$\frac{\Gamma \vdash f, f' : A \to B \qquad \Gamma \vdash a, a' : A}{\Gamma; f = f', a = a' \vdash fa = f'a'}$$

$$\frac{\Gamma \vdash a, a' : A}{\Gamma; a = a' \vdash [a] = [a']} \qquad \frac{\Gamma, x : A; \Phi \vdash b = b' : B}{\Gamma; \Phi \vdash \lambda x^A.b = \lambda x^A.b'} \quad (x \notin \mathrm{FV}(\Phi))$$

$$\frac{\Gamma \vdash e_1, e_1' : TA \qquad \Gamma, x : A; \Phi \vdash e_2 = e_2' : TB}{\Gamma; \Phi, e_1 = e_1' \vdash (\mathit{let}\ x \Leftarrow e_1\ \mathit{in}\ e_2) = (\mathit{let}\ x \Leftarrow e_1'\ \mathit{in}\ e_2')} \quad (x \notin \mathrm{FV}(\Phi))$$

Functions:

$$\beta \quad \frac{\Gamma, x : A \vdash b : B \qquad \Gamma \vdash a : A}{\Gamma \vdash (\lambda x^A.b)a = b[a/x]} \qquad \eta \quad \frac{\Gamma \vdash f : A \to B}{\Gamma \vdash f = \lambda x^A.fx}$$

A.2

Computations:

$$\frac{\Gamma \vdash e : TA}{\Gamma \vdash (\mathit{let}\ x \Leftarrow e\ \mathit{in}\ [x]) = e} \qquad (\mathrm{MONO}) \quad \frac{\Gamma \vdash a, a' : A}{\Gamma; [a] = [a'] \vdash a = a'}$$

$$\frac{\Gamma \vdash a : A \quad \Gamma, x : A \vdash e : TB}{\Gamma \vdash (\mathit{let}\ x \Leftarrow [a]\ \mathit{in}\ e) = e[a/x]}$$

$$\frac{\Gamma \vdash e : TA \qquad \Gamma, x : A \vdash e' : TA' \qquad \Gamma, x' : A' \vdash e'' : TA''}{\Gamma \vdash (\mathit{let}\ x' \Leftarrow (\mathit{let}\ x \Leftarrow e\ \mathit{in}\ e')\ \mathit{in}\ e'') = (\mathit{let}\ x \Leftarrow e\ \mathit{in}\ (\mathit{let}\ x' \Leftarrow e'\ \mathit{in}\ e''))}$$

Booleans:

$$\frac{\Gamma; \Phi, b = \mathit{tt} \vdash \phi \qquad \Gamma; \Phi, b = \mathit{ff} \vdash \phi}{\Gamma; \Phi \vdash \phi} \qquad \frac{\Gamma; \Phi \vdash \mathit{tt} = \mathit{ff}}{\Gamma; \Phi \vdash \phi}$$

$$\frac{\Gamma \vdash a, a' : A}{\Gamma \vdash \mathit{cond}(\mathit{tt}, a, a') = a} \qquad \frac{\Gamma \vdash a, a' : A}{\Gamma \vdash \mathit{cond}(\mathit{ff}, a, a') = a'}$$

Testing names:

$$\frac{\Gamma \vdash n : \mathit{Name}}{\Gamma \vdash \mathit{eq}(n, n) = \mathit{tt}} \qquad \frac{\Gamma \vdash n, n' : A}{\Gamma; \mathit{eq}(n, n') = \mathit{tt} \vdash n = n'}$$

Generating names:

$$\text{(DROP)} \qquad \frac{\Gamma \vdash e : TA}{\Gamma \vdash e = (\mathit{let}\ n \Leftarrow \mathit{new}\ \mathit{in}\ e)} \quad (n : \mathit{Name} \notin \Gamma)$$

$$\text{(SWAP)} \qquad \frac{\Gamma, n, n' : \mathit{Name} \vdash e : TA}{\begin{array}{l}\Gamma \vdash \mathit{let}\ n \Leftarrow \mathit{new}\ \mathit{in}\ (\mathit{let}\ n' \Leftarrow \mathit{new}\ \mathit{in}\ e) \\ \qquad = \mathit{let}\ n' \Leftarrow \mathit{new}\ \mathit{in}\ (\mathit{let}\ n \Leftarrow \mathit{new}\ \mathit{in}\ e)\end{array}}$$

$$\text{(FRESH)} \qquad \frac{\Gamma \vdash n : \mathit{Name} \qquad \Gamma, n' : \mathit{Name}; \Phi, eq(n, n') = \mathit{ff} \vdash e = e'}{\Gamma; \Phi \vdash (\mathit{let}\ n' \Leftarrow \mathit{new}\ \mathit{in}\ e) = (\mathit{let}\ n' \Leftarrow \mathit{new}\ \mathit{in}\ e')}$$

Bibliography

[1] J. Goubault-Larrecq, S. Lasota, D. Nowak. Logical relations for monadic types. To appear in *CSL'2002*, LNCS. Springer, 2002.

[2] G. Lowe. An attack on the needham-schroeder public-key authentication protocol. *Information Processing Letters*, 56(3):131-133, 1995.

[3] J. C. Mitchell. *Foundations for Programming Languages*. MIT Press, 1996.

[4] E. Moggi. Computational lambda-calculus and monads. In *4th LICS Conf.* IEEE, 1989.

[5] A. M. Pitts, I. D. B. Stark. Observable properties of higher order functions that create local names, or: What's new? In *MFCS'93*, LNCS 711, pages 122-141. Springer-Verlag, 1993.

[6] I. D. B. Stark. *Names and Higher-Order Functions*. PhD thesis, University of Cambridge, 1994.

[7] E. Sumii, B. C. Pierce. Logical relation for encryption. To appear in *Journal of Computer Security*, IOS Press, the Netherlands.