University of Sussex

A Program Logic For Fresh Name Generation

Harold Pancho Gordon Eliott

Submitted for the degree of Doctor of Philosophy

University of Sussex

August 2021

Declaration of Authorship

I hereby declare that this thesis has not been and will not be submitted in whole or in part to another University for the award of any other degree.

Signature:

Harold Pancho Gordon Eliott

Acknowledgements

My primary thanks are to my supervisor, Dr Martin Berger. From the day we met and discussed the possibility of me doing a PhD in logic, to the day I submit this thesis, Martin has been all I could ask for in a supervisor. From his patience as I switched gears from maths and physics and stumbled through the basics of programming languages to the never-ending process of finishing the research and thesis, Martin has been invaluable to my journey as a researcher. This work would not have been possible without his helpful guidance, words of encouragement and useful insights and ideas. I have been extremely fortunate to have him as a supervisor and a friend. For all this, Martin, thank you!

Thanks also to my second supervisor Dr Bernhard Reus for his guidance, support and supervision, and to Dr Ian Mackie for his many words of advice and wisdom over the years.

I am thankful to the Department of Informatics at the University of Sussex for funding my PhD, without which this undertaking would have been impossible.

To the friends who have been so supportive and motivating over the years, Serena, my 'odd-family', Constantin, Susana, and others, I will forever be grateful. I would like to thank all my friends and colleagues in the PhD office over the years for their friendship and interesting discussions, and in general, making work more enjoyable.

To my belated father Gordon, who always encouraged learning and thinking I am forever grateful. I am thankful to the rest of my family, my mother Elena, and brother Waby for being supportive of my endeavour through the ups and downs, and in particular my sister Jara, for also diligently proofreading this thesis for grammatical errors.

An enormous thanks to my partner Gabrielle, who not only put up with me and supported me throughout the years but gave meaning to life outside of the PhD. Finally, thanks to our child, whom Gabrielle currently bears, for ultimately providing me with even more of a reason to finish the thesis before their birth.

Abstract

This thesis introduces a program logic for an extension of the call-by-value simply typed λ-calculus (STLC), with a mechanism for the generation of fresh names via gensym, which is an adaptation of Pitts and Stark's ν-calculus [52]. Names can be compared for equality and inequality, producing programs with subtle observable properties.

Hidden names, produced by interactions between name generation and λ-abstraction, are captured logically with a new restricted quantification. The restrictions require only derived values from previously derived terms, ensuring hidden names are not revealed. The concept of derivation is extended to type contexts and models, ensuring hidden names are not revealed at later stages. Type contexts are adapted to include an order and the ability to represent future extensions. The logic quantifies over future extensions, using a second-order quantification over future type contexts. This quantification names the future context to allow for them to be reasoned about within the logic.

A new model construction is introduced to replicate the order in which names and values are produced with potentially hidden names. The semantics of the logic in the new model are used to prove each axiom and rule sound and as such the soundness of the logic. A proof that the logic is an extension of the STLC logic is given alongside a sketch of the proof that the extension is conservative.

Usage of the logic is illustrated through reasoning about numerous examples. These examples range from simple STLC and ν-calculus examples to well-known difficult programs from the literature.

Preface

This thesis is an extended version of the work presented in [17]. The paper was written by myself as the lead author, and my supervisor, Dr Martin Berger, as the co-author, however, the work outlined is substantially my own. The program logic, the model and the soundness proofs have all been designed and proven by myself under the supervision of Dr Martin Berger. The text from the paper has primarily been expanded upon and no longer exists in its original form in this thesis. This thesis extends the paper with some new axioms and more detailed explanations and reasoning examples, however the core mathematical concepts remain the same. Unlike the paper, the soundness proofs and all the requirements for it are included in their entirety in this thesis.

Notation

Notational abbreviations are included here to provide an easy reference for the reader. The abbreviations F.O.L., M.P. and I.H. are shorthand for First Order Logic, Modus Ponens and Induction Hypothesis, respectively. When particular definitions, semantics, lemmas and theorems are referred to, these are abbreviated to Def., Sem., Lem.and Thm. respectively. When chapters and (sub)sections reference a particular part of the text, they are abbreviated to Chapt. and Sec. respectively.

The base program logic will be written in standard text where needed but when reasoning about this base logic, a meta-logic is required and for this a colour coded version is used i.e. $\boldsymbol{meta-logic}$. This applies to all logical symbols listed here: $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftarrow$, $\leftrightarrow$, $\forall$, $\exists$.

Logical formulae or meta logical formulae which extend over the page width are rewritten over multiple lines with the assumption that horizontal alignment is within the same bracket i.e. $A \wedge \forall X.(B \rightarrow C)$ is identical to the following.

$$\begin{aligned} & A \\ \wedge \forall X.\ & B \\ & \rightarrow C \end{aligned}$$

Many soundness proofs and program logic derivations are displayed in the following form.

1	Step 1	*remark 1*
2	Step 2	*remark 2*
3	Step 3	*remark 3*

The line numbers are included to be able to discuss the (meta) logical statement on the line with the remark on the right explaining how the statement is implied by the line(s) above using lemmas/tautologies or rules and axioms. Sometimes a $\rightarrow$ or $\leftrightarrow$ is included before the "Step X" to show that the proof works in a single direction or in both directions.

A summary of the notation used throughout this thesis is introduced in the following table, with the location of first introduction for convenience.

Notation	Meaning	First Defined In
$\rightarrow \quad (\rightarrow^*)$	one (multiple) step reduction	Def. 4
$\Downarrow$	conversion relation	Def. 4
STLC	The simply typed λ-calculus	Sec. 2.1.2
ν_{GS}-calculus	The gensym version of the ν-calculus	Sec. 3.1
ν_{PS}-calculus	The Original version of the ν-calculus	Sec. 3.2
λ-logic	program logic for the STLC	Sec. 2.2.2
ν-logic	program logic for the ν_{GS}-calculus	Chapt. 4
Local-logic	program logic for the STLC with local state	Sec. 2.2.3
Alias-logic	program logic for the STLC with state and aliasing	Sec. 2.2.3
$\{A\}M\{B\}$	Hoare triples (imperative)	Sec. 2.2.1
$\{A\}\ M :_u \{B\}$	Hoare triples (functional)	Sec. 2.2.2
$\mathsf{fv}(\cdot)$	free variables	Def. 1
Γ	Standard Type Context (**STC**)	Sec. 2.1.2
$\mathbb{\Gamma}$	Logic Type Context (**LTC**)	Sec. 4.1
δ	Type Context Variables (**TCV**)	Sec. 4.1
$\mathbb{\Gamma}\backslash_{-TCV}$	removal of all TCV from LTC	Def. 41
$\mathbb{\Gamma}\backslash x$	removal of variable x from LTC	Def. 41
$\mathbb{\Gamma}\downarrow_{-TC}$	convert LTC to STC	Def. 41
$\mathring{\mathsf{a}}(\cdot)$	all names	Def. 20
$\mathbb{\Gamma} \Vdash \quad (\Gamma \vdash)$	LTC (STC) typing judgement	Def. 4.2 (Def. 36)
$\mathsf{ftcv}(\cdot)$	free type context variables	Def. 47
$[e/x]_{\mathbb{\Gamma}}$	logical substitution (ν-logic)	Def. 50
$[\mathbb{\Gamma}'/\delta]_{\mathbb{\Gamma}}$	logical substitution of LTC (ν-logic)	Def. 53
$\textsc{ExtInd}_{Syn} \quad (_{Sem})$	syntactic (semantic) extension independence	Def. 54 (Def. 77)
$\textsc{Thin}_{\ Syn} \quad (_{Sem})$	syntactic (semantic) thinness	Def. 55 (Def. 78)
$\cong^G_\alpha$	contextual congruence (ν_{GS}-calculus)	Def. 29
ξ	model	Def. 58
$[\![\cdot]\!]_\xi$	interpretation in model ξ	Def. 13
$M \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V$	LTC derived value	Def. 68
$\xi \preccurlyeq \xi' \quad (\preccurlyeq^\star)$	single (multi) step model extension	Def. 69 (Def. 70)
$\mathbb{\Gamma} \triangleright \xi$	LTC constructed model	Def. 71
$\xi \models \cdot$	semantics in model ξ	Def. 74
$\xi \cong \xi'$	congruent models	Def. 106

Contents

List of Figures

Chapter 1

Introduction

This chapter introduces the underlying concepts for the thesis at a level of understanding for all. The concept of names are introduced alongside the idea of program logics in Sec. 1.1 and Sec. 1.2 respectively. The overlap of these two concepts form the basis of the contributions of this thesis which are covered in Sec. 1.3. The outline of the thesis is given in Sec. 1.4.

1.1 Names

At birth each human is (normally) given a name. Each name uniquely identifies that single person however the name itself may not be unique. Consider the name *Steve McQueen* which may refer to the *king of cool*, the actor *Terrance Steven McQueen* or the Oscar winning director *Steve Rodney McQueen*. Now consider the name *Pancho Eliott* which, to the best of my knowledge, refers only to the author of this thesis. Names are used to refer to a wide range of "things" including companies, buildings, pets, ... you name it. Some names have structure such as the IP address *172.16.254.1*, other names do not such as *gobbledygook1234XYZ* (or any other random combinations of letters and numbers). Names may not even be unique to homo-sapiens, signature whistles are thought to be used by bottlenose dolphins (Tursiops truncatus) to address each other [30, 32] and similar behaviour has been observed in parrotlets (Forpus passerinus) [3].

Names can be public knowledge, such as the web address *www.google.com*, whilst others are known to a select few, for example a secret password. In contrast, anybody can name any item by any name, this can be taken to their grave or shared with whomever they care to, but in either case these names are equally valid even if only one person uses such a name. The sharing of a name can be thought of as a *scope*, where a name never shared

has a small scope, while a name that is widely shared has a large scope. The scope of a name is fundamental to names themselves, considering that without the ability to share names they can only be known to one person and hence have very limited use.

One key aspect of names is the ability to distinguish them, i.e. is the *king of rock and roll* the same name as *Elvis Presley*? which is clearly false as they use different combinations of letters. A secondary aspect of names is the item they represent (or denote) i.e. is the *king of rock and roll* the same person as *Elvis Presley*? which is clearly true given the wide usage of the former to refer to the latter. The distinguishing of the identifier used as a name and the actual item the name denotes has been investigated at length throughout history [35, 21, 45]. This thesis focusses on the former by abstracting the idea of names and ignores the latter by not assigning meaning to names.

In many instances the actual name does not play an important role, it is purely the fact that everybody uses the same name to refer to the same object which is important. Consider the name *George* in English, this name can be, and often is, replaced by the name *Jorge* in Spanish or *Giorgio* in Italian, and it isn't the specific name which is important but the fact that everybody uses the same name when referring to the same person (or item). If the two members of a conversation agree on a common name change for an item then the exact same meaning is conveyed by using the different name if used in the same context. This assumes the new name is not used elsewhere, otherwise complications arise. This same idea applies to programs which use variables. The variables do not have an inherent meaning so the names of the variables can be swapped if the swapping is done consistently. This feature is known as α-convertibility and is introduced in more detail later in Def. 2.

Sometimes systematic methods of producing names are required. Examples of these include the names of pictures taken by digital cameras. These names may be given by a human (for instance a descriptive name), or sometimes these will be given by a machine. The machine may just number the pictures using information such as the date i.e. *Picture1/1/2021at00:00*, *Picture1/1/2021at00:04*, *Picture3/1/2021at12:00*... Alternatively a counter system may be used to name the images using some number that increases for each image i.e. *Picture1*, *Picture2*, ... The infiniteness of the number system and the purity of a counter, which adds one each time, makes this an attractive method to come up with new names.

Without considering what names represent, the most general idea of names includes a method to create new names and a method to share names (i.e. scope) and a method to

compare whether two names are equal. In common natural language, names are not often created, due to the learnt dictionary of names instilled in us from a young age, however any time anything new is created, a new name is often created for said item (including people).

Names are used in computers and programming languages to introduce identity in a variety of applications. Take the simple case of naming files which can be done by the human user or sometimes by the machine itself. More technical applications of names in programming languages include references, objects, exceptions, channels and many more which builds on the idea of names, often by making the name represent something within the language. For each of these examples the names are of a particular type such that the names used in one application cannot be used in another. For example a channel-name cannot be used to refer to an exception-name even if the name itself uses the same characters in the same position (exceptions to this statement do exist).

Given the importance of names, studying them in the setting of programming languages in their simplest form, ensures the essence of the concepts described above are represented in the language and studied without the complications that may arise from more complicated uses of names. The π-calculus introduced names in their simplest form using a single constructor for names (the fresh name generator) and a single destructor for names (the equality of names). However, the π-calculus mixes names with message passing and parallelism, which is not necessarily the simplest use case. The ν-calculus introduced names to the minimal sequential programming language of the STLC. This captures the essence of names without the complexity of the specific applications of names (in channels and parallelism), meaning any work on this simple extension of pure names can be applied to the more complicated (sequential) uses of names.

In typed programming languages every type has a constructor and a destructor. This includes the type of names in the ν-calculus (introduced in more detail in Chapt. 3). The constructor for names is $\nu\mathsf{n}.M$, which creates a fresh name n which can be used within the M part of the program. The destructor for names is the equality operator $M = M'$, which compares the two names at M and M' for equality and returns true if they are equal and false otherwise. The dual inequality operator can also be included to mean two names are not equal. The fresh name n in $\nu\mathsf{n}.M$ may share the name with the outside world if M shares the name n. For example: $\nu\mathsf{n}.\mathsf{n}$ creates a fresh name and outputs that same fresh name; however, $\nu\mathsf{n}.\mathsf{true}$ creates the fresh name n but never shares the name because it only outputs the value true, so this name is created but lost forever as it is never "shared". More

complicated examples exist which may even use the freshly created name but never share it.

In day to day life there is nothing to stop names being guessed. If the name is common or clues (such as a pattern) are given then this guessing becomes easier. However, given a limited alphabet from which to make names, and given an unlimited number of guesses there is always a way to systematically work through all possible names, i.e. a, b, ... , aa, ab, ... ba, bb, ... and so on. Clearly the longer the name the longer it will take to guess, however it is just a matter of time before it is guessed. This guessing feature is not considered in the ν-calculus introduced in Chapt. 3, as the names are defined such that they cannot be guessed. This ensures that the only access to names is through producing them and sharing them and not some "guess". The field of cryptography studies the ability for names (or passwords) to be "guessed" in many different forms, however this is not in the remit of this work.

1.2 Program Logics

The ubiquity of computers in everyday life means life depends more and more on the results these machines output. Defining what programs do can be simple in the case of running the program i.e. run program X in situation Y, or can be tricky if a general property of the program is required as the definition, i.e. program X does not leak credit card information in any situation. Common practice for programs is to provide documentation (which is often hand written) stating what each part of the program does. This is useful but does not guarantee that what the program does will coincide exactly with what the documentation states.

It is possible to formally define what a program does by stating how the program is "run" (using reduction semantics) however this requires doing this for each possible initial state the program is run in. For example if $x = 3$ in the initial state then $y := x+1$ returns $y = 4$ as the final state. However this doesn't hold if $x \neq 3$ in the initial state, hence the whole reduction needs to be performed in each initial state. This may seem trivial for this example, but given the infiniteness of numbers it is impossible and for more complex programs this can get even more complicated very quickly.

Ideally for the program $y := x + 1$ it can be stated that for any number x in the initial state then after running $y := x + 1$ then $y = x + 1$ in the final state. This abstraction is captured by Hoare logics (which are often referred to as program logics). They allow for the reasoning about what programs do in an abstract manner, so that they can be used under

any initial starting condition. This is referred to as compositionality. More complicated properties of the initial and final state can be defined. Taking the same program $y := x+1$ with the initial state which contains an odd number at x then in the final state clearly y must be an even number. More complicated properties can be defined for more complicated programs which increases the power of what this method of reasoning can achieve.

Program logics build on a foundational mathematical logic (for example Set theory or Peano arithmetic) to reason about program behaviours. For terminating (or stateful) programs, the logical formula and the (initial or final) state share variables which must be compatible. For instance the state with the variable x storing 3 and the formula $x = 2$ fail to be compatible as $3 = 2$ is false. However the state with x storing 3 and the formulae $x \leq 4$ are compatible as $3 \leq 4$ is true. Peano arithmetic then allows for the manipulation of these logical formulae, for instance $x \leq 4$ and ($x \leq 3$ or $x = 4$) are identical. The examples introduced here are simple but the formulae can get more complex the larger the number of variables and the more complex the programs become along with the specific behaviours the program logic is intending to capture.

Hoare logic relates the initial state requirements, a program and the final state requirements through the use of these logical formulae as follows. The Hoare triple $\{A\}M\{B\}$ states that if the initial state satisfies the pre-condition formula A, and M is a program run in this state which produces a final state, then this final state satisfies the post-condition formula B. If M never terminates then there is no final state and the Hoare triple is of no real use. Rules dictate the derivation of these triples based on the structure of the pre- and post-conditions, or the structure of the program being reasoned about. The intention of deriving a Hoare triple using rules is that the triple states something useful (and 'true') regarding the program itself which can also be used in reasoning about applications of the program in a compositional manner.

This leads to the question:

> Can a simple sequential language with names be reasoned about using a Hoare-style program logic in a simple manner that allows for compositional reasoning?

1.3 Contributions

This thesis aims to answer the question posed above. The programming language reasoned about is an adaptation of the ν-calculus which separates the scope of names from the generation of fresh names. The program logic introduces a version of type contexts, into

the logic itself, alongside a new higher order quantification over type contexts, which has the benefit of being able to name said type context within the logic. A new restricted quantification over values derived from a type context is also introduced to replace standard universal quantification such that the possible names that a quantifier can now range over are derived only from previous uses of the name, ensuring that hidden names do not become revealed. The program logic is proven sound and a sketch for the conservativity of the program logic for the STLC is given. Numerous programs are reasoned about using the program logic provided to show applications of the logic.

1.4 Thesis Outline

The thesis content is organised as follows.

Chapter 2 introduces the technical background required in the thesis. This includes the core programming languages, starting with the untyped λ-calculus, then the STLC. Program logics for a simple imperative programming language and functional programming language are introduced as the foundation of the work in this thesis in Sec. 2.2.1 and Sec. 2.2.2 respectively. A programming language with local state, which has clear uses of names, and an adaptation of the logic for the local state language is introduced in Sec. 2.2.3.

Chapter 3 introduces the different approaches in the literature, to adding names to the STLC. Sec. 3.1 introduces the ν_{GS}-calculus which generates names via gensym. This will be reasoned about using the program logic of the future chapters. The original ν_{PS}-calculus is introduced in Sec. 3.2 and comparisons are made between the two versions in Sec. 3.3. A summary of the different methods to prove equality in the original ν_{PS}-calculus is found in Sec. 3.4. The $\lambda\nu$-calculus, which introduces names to the CBN STLC, is introduced in Sec. 3.5, alongside its relation to the ν_{PS}-calculus in Sec. 3.6.

Chapter 4 formally defines the ν-logic, the logic used to reason about programs in the ν_{GS}-calculus. This includes formally defining the logical syntax, type checking, logical substitutions, syntactic properties of formulae, the axioms/axiom schemas and the rules required to reason about programs, in Sec. 4.1 to Sec. 4.6. Discussions on design choices are also included in Sec. 4.7, to show alternative options in the development process of the logic.

Chapter 5 provides a mathematical model in which to interpret the ν-logic. The syntactic properties of formulae are given a semantic version of the property in Sec. 5.3. To make the soundness proof shorter and more readable, various lemmas regarding the model

are introduced and proven in Sec. 5.4.

The soundness proof of the ν-logic is provided in Chapter 6. The soundness proof requires that syntactic properties of formulae imply the semantic properties of the formulae, for which a proof is provided in Sec. 6.1. The soundness of all axioms (Sec. 6.2) and all rules (Sec. 6.3) form the basis of the soundness proof of the ν-logic in Sec. 6.4. The ν-logic is proven to be an extension of the program logic for the STLC in Sec. 6.5, whilst a sketch is also provided to prove that the extension is a conservative extension.

Key programs in the ν_{GS}-calculus are reasoned about in Chapter 7, including most programs found in the literature.

Chapter 8 concludes with an analysis of the research, with a discussion on possible future work and related ideas including further generalisations of the axioms, relations to both nominal logic and hybrid logic, mechanisation of the proofs and further applications of the ν-logic.

Chapter 2

Technical Background

This chapter gives an introduction to the technical background required to understand the content on this thesis. The basics of λ-calculus are introduced in Sec. 2.1 covering the untyped λ-calculus and the STLC. The STLC will form the basis of calculus introduced in Chapt. 3, which adds names to the STLC which in turn is the core language researched in this thesis.

Program logics are introduced in Sec. 2.2 in detail for three different languages: an imperative language, the STLC and an extension which adds local state to the STLC in Sec. 2.2.1, Sec. 2.2.2 and Sec. 2.2.3 respectively.

2.1 The λ-Calculus

The λ-calculus was introduced by Alonzo Church in 1932 [12] as a logical system for the foundations of mathematics in a time before computers. The theoretical understanding of computation developed in parallel to, but independently from, the development of physical machines. At a similar time various attempts at capturing the essence of computation were also developed:

- Turing Machines: Alan Turing in 1936 [66]
- μ-recursive functions: Kurt Gödel in 1934 [23]
- Rewrite systems: Emil Post in 1936 [54]
- Combinatory Logic: Moses Schönfinkel in 1924 [61], further developed by Haskell Curry in 1929 [16].

All of these systems model computation in differing forms, but all turned out to be computationally equivalent including the λ-calculus, i.e. they could each model each other.

Each of these systems has been developed and extended in many different directions, with works often overlapping and work in one area often inspiring developments in another.

Stephen Kleene and John B. Rosser proved Church's logical system inconsistent using what is now known as the Kleene-Rosser paradox [33]. This logical system was later fixed using types in the STLC [13]. The logical system of the λ-calculus is adapted to be used as a programming language (or calculus), with some alterations made to introduce the ideas behind term reduction. Both the untyped and typed λ-calculus are introduced below.

2.1.1 The Untyped λ-Calculus

Introducing the untyped λ-calculus as a programming language requires the syntax from the logical form of λ-calculus introduced by Alonzo Church, however a slightly different notion of reduction is introduced in the style of computers "running" programs. This calculus is refered to as the untyped λ-calculus, to distinguish it from the STLC, introduced in Sec. 2.1.2. Both calculi are introduced here to have a common syntactic basis.

The syntax for the untyped λ-calculus is defined recursively as the terms M in Fig. 2.1.

$$M \quad ::= \quad x \;\mid\; \lambda x.M \;\mid\; MM$$

Figure 2.1: Syntax of the untyped λ-calculus.

The infinite set of variables range over x, y, z, ... but are introduced as above for simplicity. Functions $\lambda x.M$ bind the variable x in the term M, meaning the variable x may occur freely in M. Functions are applied to other terms as $(\lambda x.M)M'$ where the M' is substituted (in some form) for the occurrences of x in M. Variables are used as place holders or binders for where terms will be substituted.

Definition 1 (Free and bound variables)**.** *Bound variables of a term M are those variables occurring in M that are bound by a λ within M. Variables that occur unbounded are defined as* free variables *and the set of free variables in a term M is defined $\mathsf{fv}(M)$ inductively as follows.*

$$\begin{aligned} \mathsf{fv}(x) &\stackrel{\text{def}}{=} \{x\} \\ \mathsf{fv}(\lambda x.M) &\stackrel{\text{def}}{=} \mathsf{fv}(M) - \{x\} \\ \mathsf{fv}(MN) &\stackrel{\text{def}}{=} \mathsf{fv}(M) \cup \mathsf{fv}(N) \end{aligned}$$

Consider the two terms $\lambda x.x$ and $\lambda y.y$, the exact name of the binding variable can be replaced if the uses are also swapped within the bound term, meaning the manner in which

the bound name is used is more important than the specific variable used. This gives rise to renaming of bound variables known as α-conversion as follows. Let the terms $M(x)$ and $M(y)$ be the same term with y replacing all free occurrences of x in the latter.

Definition 2 (α-equivalence)**.** *Two terms are deemed* α-equivalent *if they differ only in the names of bound variables.*

Two α-equivalent terms are also *α-conversions* of one another and vice versa hence the two notions are often used interchangeably.

The α-equivalence is similar to how names can be swapped if every use of the name is also swapped for a fresh name (as mentioned in Sec. 1.1).

Definition 3 (Capture avoiding substitution)**.** *Free variables are used to define substitution of a term N for the free occurrences of the variable x in a term M written $M[N/x]$. This is defined inductively as follows.*

$$
\begin{array}{rcll}
y[N/x] & \stackrel{\text{def}}{=} & y & \text{if } \neq y \\
x[N/x] & \stackrel{\text{def}}{=} & N & \\
(\lambda x.M)[N/x] & \stackrel{\text{def}}{=} & \lambda x.M & \\
(\lambda y.M)[N/x] & \stackrel{\text{def}}{=} & \lambda y.(M[N/x]) & \text{if } x \neq y \text{ and } y \notin \mathsf{fv}(N) \\
(\lambda y.M)[N/x] & \stackrel{\text{def}}{=} & (\lambda z.(M[z/y]))[N/x] & \text{if } x \neq y \text{ and } y \in \mathsf{fv}(N) \text{ and } z\text{-fresh} \\
(MM')[N/x] & \stackrel{\text{def}}{=} & (M[N/x])(M'[N/x]) &
\end{array}
$$

Line 4 ensures for the substitution $(\lambda y.M)[N/x]$ that the bound variable y does not occur free in N, without which the free name y appearing in N would become bound by the $\lambda y.$-binder, hence the name: *capture avoiding substitution.*

Definition 4 (Reduction relation)**.** *The untyped λ-calculus reduction relation $M \to N$, relates the term on the left M, to the term on the right N, as a reduction from M to N. The reflexive, transitive closure of the $\to$ relation is written $\to^*$. When values are defined, if $M \to^* V$ for some value V then this is written as $M \Downarrow V$.*

The common reduction relations are seen in Fig. 2.2.

$$
\begin{array}{lll}
(\lambda x.M)N & \to_\beta & M[N/x] \\
\lambda x.M & \to_\alpha & \lambda y.M[y/x] \quad y \notin \mathsf{fv}(M)
\end{array}
$$

Figure 2.2: Reduction relations of the untyped λ-calculus.

A function $\lambda x.M$ applied to the term N as in $(\lambda x.M)N$ is reduced by the β rule where the function substitutes in N for the λ-bound variable x occurring inside M. The $\rightarrow_\alpha$ relation is that of Def. 2 and is often taken for granted as it simply allows for the renaming of bound variables.

The η relation: $\lambda x.(Mx) \rightarrow_\eta M$ if $x \notin \mathsf{fv}(M)$ captures the idea of extensionality meaning functions are the same if they give the same results for each argument. If $(\lambda x.Mx)$ is applied to any term N, it will β-reduce to MN which is the same as just applying M to N. So this rule pre-empts the application of terms of the form $\lambda x.Mx$ to another term. The η-equivalence rule is not always included in reduction relations thus two reductions are often discussed: β-reduction and $\beta\eta$-reduction, where the former excludes η-equality, and the latter includes it in the possible reduction rules. From here onwards, calculi will not include η-reduction, but will freely rename bound variables whenever convenient via the α-equivalence. If further constructors such as $\mathsf{if}\ B\ \mathsf{then}\ M\ \mathsf{else}\ M'$ or $\langle M, M' \rangle$ are included then further reduction relations are required.

There is no specific order in which to apply reduction relations, meaning there are often a number of ways to reduce a term. Consider the example $(\lambda x.((\lambda y.(xy))a))b \rightarrow_\beta (\lambda x.xa)b$ which holds by reducing the $(\lambda y.(xy))a \rightarrow_\beta xa$ first. However, $(\lambda x.((\lambda y.(xy))a))b \rightarrow_\beta (\lambda y.by)a$ also holds as it evaluates the outermost function application first. Both reduction relations are valid and both, in turn, reduce to ba. A term is defined as *(β-)normal form* (or a value) if no further (β-)reductions are possible.

It is of interest whether the application of these rules results in a term which can or cannot be reduced further. A program which can always be reduced further is defined as non-terminating. An example of a non-terminating program is $\Omega \stackrel{def}{=} (\lambda x.xx)(\lambda x.xx)$ as $\Omega \rightarrow_\beta (xx)[(\lambda x.xx)/x] \stackrel{def}{=} \Omega$. Under any reduction strategy Ω reduces to itself hence can always be reduced again. Strong normalisation is the property of a language that each term reduces to a single normal form, hence the untyped λ-calculus is not strongly normalizing.

Evaluation strategies are introduced to specify the order of applying the reduction rules. Although others exist, the most common evaluation strategies are *Call-By-Name (CBN)* and *Call-By-Value (CBV)*. In general, evaluation strategies can be expressed in small-step semantics which describes each step of the reduction, or big-step operational semantics which describe the final result, although other approaches also exist. For example, the evaluation of $(\lambda x.((\lambda y.(xy))a))b$ via big-step gives $(\lambda x.((\lambda y.(xy))a))b \Downarrow ba$ directly, and via small-step can give the steps of the reduction as $(\lambda x.((\lambda y.(xy))a))b \rightarrow_\beta (\lambda y.(by))a) \rightarrow_\beta ba$. The example Ω cannot be reduced via big-step operational semantics as it never evaluates

to a value, however small-step semantics can show that $\Omega \to \Omega$ and hence express the non-termination.

The CBV evaluation strategy is introduced here in both big-step and small-step semantics. This first requires a definition of values.

Definition 5 (Values). *Values consist only of λ functions and variables, as these cannot be reduced further.*

$$V ::= x \mid \lambda x.M$$

Definition 6 (Call-By-Value (CBV) operational semantics). *The semantics of the CBV evaluation strategy requires the application of a λ-abstraction to be applied to a value, as opposed to a term.*

The small-step operational semantics is found in Fig. 2.3.

$$\frac{M \to M'}{MN \to M'N}\ \textsc{AppL}_{(CBV)} \qquad \frac{N \to N'}{VN \to VN'}\ \textsc{AppR}_{(CBV)} \qquad \frac{-}{(\lambda x.M)V \to M[V/x]}\ \beta_{(CBV)}$$

Figure 2.3: CBV small-step operational semantics of the untyped λ-calculus.

The big-step operational semantics is found in Fig. 2.4 consisting of one rule which in essence combines all the rules of the small-step semantics into one rule and fails to evaluate any term which does not satisfy the assumptions (unless it is already a value).

$$\frac{-}{V \Downarrow V}\ \textsc{Value} \qquad \frac{M \Downarrow \lambda x.M' \quad N \Downarrow V' \quad M'[V'/x] \Downarrow C}{MN \Downarrow V}\ \beta^*_{(CBV)}$$

Figure 2.4: CBV big-step operational semantics of the untyped λ-calculus.

An alternative method of expressing these reduction strategies is through evaluation contexts as shall be introduced here for the CBV reduction relation in the style of [15].

Definition 7 (CBV operational semantics: evaluation contexts). *The reduction relation is defined as above with $\beta_{(CBV)}$ rule in Fig. 2.3, then the order of evaluation is defined by evaluation contexts $\mathcal{E}[\cdot]$ which are defined in Fig. 2.5.*

$$\mathcal{E}[\cdot] \quad ::= \quad [\cdot] \mid \mathcal{E}[\cdot]M \mid V\mathcal{E}[\cdot]$$

Figure 2.5: CBV evaluation contexts of the untyped λ-calculus.

The evaluation contexts define when the reductions can be applied inside a term using the following rule.

$$M \to N \quad \rightarrow \quad \mathcal{E}[M] \to \mathcal{E}[N]$$

If the language is extended with new terms, the relevant reduction relations are added alongside the relevant new evaluation contexts. This thesis will tend to express the reduction relation through evaluation contexts, hence their introduction here.

2.1.2 The STLC

Although many variations of the untyped λ-calculus exist, one primary development was the restriction of the untyped λ-calculus via types, introduced by Alonzo Church in 1940 [13]. In the simple case of the STLC described here, types ensure non-termination does not occur. By definition types restrict the number of possible programs the language allows.

Types are an abstraction of programs, in the sense that functions obtain the type $\alpha_1 \to \alpha_2$ where α_2 is the type the function outputs having accepted the term of type α_1. Constants are added to the language with respective types which ensures types obtain some fixed type. The types of these constants are called *base types*. The binary type Bool (short for Boolean) with the constants true and false, with negation $\neg M$, are included with the conditional operator $\mathsf{if} \cdot \mathsf{then} \cdot \mathsf{else} \cdot$ as the destructor. The pair type $\alpha_1 \times \alpha_2$ represents a 2-tuple with the first element being of type α_1 and the second element of type α_2. Constructors of pairs are $\langle \cdot, \cdot \rangle$ whilst the destructors are $\pi_1(\cdot)$ and $\pi_2(\cdot)$ to obtain the first and second element of the pair respectively. Projections are often written $\pi_i(\cdot)$ to mean for each $i \in \{1, 2\}$. The unary type Unit and the respective constant $()$ is included without a destructor.

The syntax for the STLC defined in Fig. 2.6 extends the untyped λ-calculus with the types α described above and typing contexts Γ mapping variables to types which is used in typing judgments. Values and terms remain almost unchanged from the untyped λ-calculus with the exception of a type annotation for the λ-bound variables (which is often dropped) and including the constants in the values. The pairs and pair projection are also added

here to make future extensions more interesting. The common syntactic abbreviation is included here as $\mathsf{let}\ x^{\alpha} = M\ \mathsf{in}\ N \stackrel{def}{=} (\lambda x^{\alpha}.N)M$.

$$
\begin{array}{lrcl}
\text{(Types)} & \alpha & ::= & \mathsf{Unit} \mid \mathsf{Bool} \mid \alpha \to \alpha \mid \alpha \times \alpha \\
\text{(Typing Context)} & \Gamma & ::= & \emptyset \mid \Gamma, x : \alpha \\
\text{(Values)} & V & ::= & () \mid \mathsf{true} \mid \mathsf{false} \mid \lambda x^{\alpha}.M \mid \langle V, V \rangle \\
\text{(Terms)} & M & ::= & x \mid V \mid \neg M \mid MM \mid \mathsf{if}\ M\ \mathsf{then}\ M\ \mathsf{else}\ M \\
 & & & \mid \langle M, M \rangle \mid \pi_1(M) \mid \pi_2(M) \mid \mathsf{let}\ x^{\alpha} = M\ \mathsf{in}\ M
\end{array}
$$

Figure 2.6: Syntax of the STLC.

Further base types such as integers are often included alongside relevant operations on these base types, however these are not included here.

Terms (and types) are classified as zeroth-order (/ground type/base types) if they are a constant i.e. contains no functions. A term is of order k if its type is of order k. The order of a type can be defined by the $\mathsf{O}(\cdot)$ function of a term as follows.

$$
\begin{aligned}
\mathsf{O}(\mathsf{Bool}) \stackrel{def}{=} \mathsf{O}(\mathsf{Unit}) &\stackrel{def}{=} 0 \\
\mathsf{O}(\alpha_1 \to \alpha_2) &\stackrel{def}{=} \mathsf{max}(\mathsf{O}(\alpha_2), 1 + \mathsf{O}(\alpha_1)) \\
\mathsf{O}(\alpha_1 \times \alpha_2) &\stackrel{def}{=} \mathsf{max}(\mathsf{O}(\alpha_1), \mathsf{O}(\alpha_2))
\end{aligned}
$$

Typing judgments are a check on programs prior to evaluation to ensure programs terminate. The typing rules can be found in Fig. 2.7 where $\Gamma(x)$ is the type that x maps to in Γ.

The untyped λ-calculus program $\Omega \stackrel{def}{=} (\lambda x.xx)(\lambda x.xx)$ is not a STLC program as it fails to type check. This fails as $\lambda x^{\alpha_1 \to \alpha_2}.xx$ requires $\Gamma, x : \alpha_1 \to \alpha_2 \vdash x : \alpha_1 \to \alpha_2$ whilst simultaneously requiring $\Gamma, x : \alpha_1 \to \alpha_2 \vdash x : \alpha_1$.

Programs such as $(\lambda x^{\mathsf{Bool}}.x)\mathsf{true}$ however are typed correctly as seen below.

$$
\dfrac{\dfrac{\dfrac{\Gamma, x : \mathsf{Bool} \vdash x : \mathsf{Bool}}{\Gamma \vdash \lambda x^{\mathsf{Bool}}.x : \mathsf{Bool} \to \mathsf{Bool}} \qquad \dfrac{-}{\Gamma \vdash \mathsf{true} : \mathsf{Bool}}}{\Gamma \vdash (\lambda x^{\mathsf{Bool}}.x)\mathsf{true} : \alpha}}
$$

Hence, reducing $(\lambda x^{\mathsf{Bool}}.x)\mathsf{true}$ will produce a new term of type Bool. This property is known as *type soundness* and is often summarized as "well-typed programs cannot go wrong" [39]. Type soundness ensures that for any program M, if $\Gamma \vdash M : \alpha$ then M is either a value or will reduce to a new term of type α. Hence if a well typed program of type α terminates it will produce a value of type α.

$$\frac{-}{\Gamma \vdash () : \mathsf{Unit}} \qquad \frac{c \in \{\mathsf{true},\ \mathsf{false}\}}{\Gamma \vdash c : \mathsf{Bool}} \qquad \frac{\Gamma(x) = \alpha}{\Gamma \vdash x : \alpha}$$

$$\frac{\Gamma, x : \alpha_1 \vdash M : \alpha_2}{\Gamma \vdash \lambda x^{\alpha_1}.M : \alpha_1 \to \alpha_2} \qquad \frac{\Gamma \vdash M : \alpha_1 \to \alpha_2 \quad \Gamma \vdash N : \alpha_1}{\Gamma \vdash MN : \alpha_2}$$

$$\frac{\Gamma \vdash M : \mathsf{Bool}}{\Gamma \vdash \neg M : \mathsf{Bool}} \qquad \frac{\Gamma \vdash M : \mathsf{Bool} \quad \Gamma \vdash M_i : \alpha \quad i \in \{1, 2\}}{\Gamma \vdash \mathsf{if}\ M\ \mathsf{then}\ M_1\ \mathsf{else}\ M_2 : \alpha}$$

$$\frac{\Gamma \vdash M_1 : \alpha_1 \quad \Gamma \vdash M_2 : \alpha_2}{\Gamma \vdash \langle M_1, M_2 \rangle : \alpha_1 \times \alpha_2} \qquad \frac{\Gamma \vdash M : \alpha_1 \times \alpha_2}{\Gamma \vdash \pi_i(M) : \alpha_i}$$

$$\frac{\Gamma \vdash M : \alpha_1 \quad \Gamma, x : \alpha_1 \vdash N : \alpha_2}{\Gamma \vdash \mathsf{let}\ x^{\alpha_1} = M\ \mathsf{in}\ N : \alpha_2}$$

Figure 2.7: Typing rules of the STLC.

Definition 8 (Evaluation of the STLC(evaluation contexts))**.** *Evaluation of STLC terms builds on that of the CBV untyped λ-calculus in Fig. 2.5 if the type annotations and constants are included. The evaluation contexts are those of Fig. 2.8.*

$$\begin{aligned}\mathcal{E}[\cdot] \quad ::= \quad & [\cdot] \mid \mathcal{E}[\cdot]M \mid V\mathcal{E}[\cdot] \mid \mathcal{E}[\cdot] = M \mid V = \mathcal{E}[\cdot] \mid \neg\mathcal{E}[\cdot] \\ & \mid \mathsf{if}\ \mathcal{E}[\cdot]\ \mathsf{then}\ M\ \mathsf{else}\ M' \mid \mathsf{if}\ V\ \mathsf{then}\ \mathcal{E}[\cdot]\ \mathsf{else}\ M' \mid \mathsf{if}\ V\ \mathsf{then}\ W\ \mathsf{else}\ \mathcal{E}[\cdot] \\ & \mid \langle \mathcal{E}[\cdot], M \rangle \mid \langle V, \mathcal{E}[\cdot] \rangle \mid \pi_i(\mathcal{E}[\cdot]) \mid \mathsf{let}\ x = \mathcal{E}[\cdot]\ \mathsf{in}\ M\end{aligned}$$

Figure 2.8: Evaluation contexts of the STLC.

The reduction rules are those of Fig. 2.9.

$$
\begin{array}{rcll}
V & \to & V & \\
(\lambda x^{\alpha}.M)V & \to & M[V/x] & \\
b = b & \to & \mathsf{true} & b \in \{\mathsf{true}, \mathsf{false}\} \\
b = b' & \to & \mathsf{false} & b \neq b',\ b, b' \in \{\mathsf{true}, \mathsf{false}\} \\
\neg\mathsf{true} & \to & \mathsf{false} & \\
\neg\mathsf{false} & \to & \mathsf{true} & \\
\mathsf{if\ true\ then}\ V\ \mathsf{else}\ V' & \to & V & \\
\pi_i(\langle V_1, V_2 \rangle) & \to & V_i & \\
\mathsf{let}\ x = V\ \mathsf{in}\ M & \to & M[V/x] & \\
M \to M' & \to & \mathcal{E}[M] \to \mathcal{E}[M'] &
\end{array}
$$

Figure 2.9: Reduction rules of the STLC.

This version of the STLC is strongly normalizing (or terminating) as recursion is not included and terms must be well typed.

The STLC provides an excellent base to express features in (functional) programming languages. Numerous extensions are common in the literature including integers with integer arithmetic [13], recursion, a store, parametric polymorphic types, names (introduced in Chapt. 3), and many more.

2.1.3 Contextual Equivalence (STLC)

The most common method of distinguishing or equating two terms is contextual equivalence, first introduced by Morris in 1969 [41]. Two programs are considered contextually equivalent if they cannot be distinguished by any context in the same programming language. If two programs are contextually indistinguishable then they can be swapped for one another in any situation they may occur and produce the same result.

Contextual equivalence is introduced here for the STLC, however extends to most other languages.

Definition 9 (Single holed contexts)**.** *For the STLC, a single-holed context is defined in Fig. 2.10.*

$$
\begin{aligned}
\mathsf{C}[\cdot]_\alpha \quad ::= \quad & [\cdot]_\alpha \mid \lambda x^{\alpha_1}.\mathsf{C}[\cdot]_\alpha \mid \mathsf{C}[\cdot]_\alpha M \mid M\mathsf{C}[\cdot]_\alpha \mid \neg\mathsf{C}[\cdot]_\alpha \mid \mathsf{C}[\cdot]_\alpha = M \mid M = \mathsf{C}[\cdot]_\alpha \\
& \mid \mathsf{if}\ \mathsf{C}[\cdot]_\alpha\ \mathsf{then}\ M'\ \mathsf{else}\ M'' \mid \mathsf{if}\ M\ \mathsf{then}\ \mathsf{C}[\cdot]_\alpha\ \mathsf{else}\ M'' \mid \mathsf{if}\ M\ \mathsf{then}\ M'\ \mathsf{else}\ \mathsf{C}[\cdot]_\alpha \\
& \mid \langle \mathsf{C}[\cdot]_\alpha, M\rangle \mid \langle M, \mathsf{C}[\cdot]_\alpha\rangle \mid \pi_i(\mathsf{C}[\cdot]_\alpha) \\
& \mid \mathsf{let}\ x = \mathsf{C}[\cdot]_\alpha\ \mathsf{in}\ M \mid \mathsf{let}\ x = M\ \mathsf{in}\ \mathsf{C}[\cdot]_\alpha
\end{aligned}
$$

Figure 2.10: Single holed contexts of the STLC.

The hole $[\cdot]_\alpha$, in a single holed context, can be filled with a term of type α such that the typing conditions still hold. Any context filled with an appropriately typed term is a STLC term and can thus be typed accordingly. Unlike capture free substitution, the context may capture free variables that occur in the term filling the hole. For instance $\mathsf{C}[\cdot] \equiv \lambda x.[\cdot]$ binds any free occurrence of the variable x in the term which fills the hole, i.e. $\mathsf{C}[x]$ becomes $\lambda x.x$.

Contexts (with an empty hole) can be typed using the typing rules in Fig. 2.7 and the following rule for holes.

$$\frac{-}{\Gamma \vdash [\cdot]_\alpha : \alpha}$$

Definition 10 (Contextual equivalence in the STLC)**.** *Contextual equivalence of two closed terms M and N both of type α (written $M \cong_\alpha^{STLC} N$) is defined as the inability for any context to distinguish these terms, formally defined as follows.*

$$M \cong_\alpha^{STLC} N \quad \stackrel{\text{def}}{=} \quad \forall\, \mathsf{C}[\cdot]_\alpha.\ \emptyset \vdash \mathsf{C}[\cdot]_\alpha : \mathsf{Bool} \to (\mathsf{C}[M] \Downarrow \mathsf{true} \leftrightarrow \mathsf{C}[N] \Downarrow \mathsf{true})$$

The contexts are chosen to evaluate to true, however the definition holds identically if the constant false is chosen.

This definition quantifies over all single-holed contexts which makes it difficult to reason about. Some methods of proving more coarse or fine relations are introduced to make the task of proving equality or inequality easier. In Sec. 3.4 some of these techniques are introduced for the ν_{PS}-calculus which by extension also hold for the STLC.

2.2 Program Logics

In 1969, Tony Hoare introduced a formal system for reasoning about correctness of programs [24] based on work by Robert W. Floyd [20]. Hoare logic (or Floyd-Hoare logic) originally used classical First Order Logic (F.O.L.) as the basis for logical assertions to

reason about the state prior and post evaluation of programs in an imperative programming language. A Hoare (logic) triple of the form $\{P\}M\{Q\}$ states that if assertion P (pre-condition) holds, then executing program M establishes assertion Q (post-condition). Originally this was introduced for the While language introduced in Sec. 2.2.1 and has since evolved to reason about numerous extensions and other different languages. For example the triple $\{\mathsf{Odd}(x)\}x := x * 2\{\mathsf{Even}(x)\}$ states that if x is initially odd and the program $x := x * 2$ (which assigns $x * 2$ to x) is evaluated then x is now even. The pre-condition $\mathsf{Odd}(x)$ and post-condition $\mathsf{Even}(x)$ are logical formulae in the formal language of Peano arithmetic.

The name *program logic* is often interchangeable for the name *Hoare logic* however in this thesis the latter will refer only to program logics for imperative languages, whereas the former will be used for all languages. The Hoare logic for a simple imperative language is introduced in Sec. 2.2.1 and referred to as the While-logic. The program logic for the STLC is introduced in Sec. 2.2.2 and referred to as the λ-logic, alongside the program logic for an extension to the STLC which includes local state in Sec. 2.2.3 referred to as Local-logic.

2.2.1 Hoare Logic for a Simple Imperative Language (While-Logic)

The Hoare logic for the simple, yet useful, While programming language is introduced here. The While language consists of the syntax in Fig. 2.11.

$$
\begin{array}{lcl}
E & ::= & n \in \mathbb{Z} \mid V \mid E + E \mid E - E \mid E \times E \mid \ldots \\
B & ::= & \mathsf{true} \mid \mathsf{false} \mid E = E \mid E \leq E \mid \ldots \\
M & ::= & V := E \mid M; M \mid \mathsf{if}\ B\ \mathsf{then}\ M\ \mathsf{else}\ M \mid \mathsf{while}\ B\ \mathsf{do}\ M
\end{array}
$$

Figure 2.11: Syntax of the While language.

The set V of variables range over (uppercase) $X, Y, Z, \ldots$ Expressions E, range over the natural numbers n, variables and primitive operations on expressions. Boolean statements B include the Boolean constants and operations on expressions which return Booleans. Commands (or programs) M, include assignment of an expression E to a variable in V in the store, composition of commands, conditionals and a while loop. The execution of these commands with a state (consisting of non-aliased variables in V mapped to natural numbers) is as expected and not discussed here.

The logical language of formulae used in the pre- and post-conditions of the Hoare triple

is that of F.O.L. (or more precisely Peano arithmetic) with (auxiliary) variables ranging over (lowercase) a, b, c, ... different to the set of variables V in the programming language. The formulae are formally defined in Fig. 2.12 using an an extension to expressions E from Fig. 2.11 to include the auxiliary variables a as E.

$$\begin{array}{lcl} \mathsf{E} & ::= & n \in \mathbb{Z} \mid a \mid V \mid \mathsf{E} + \mathsf{E} \mid \ldots \\ P & ::= & \mathsf{T} \mid \mathsf{F} \mid \mid \mathsf{E} \leq \mathsf{E} \mid \neg P \mid P \wedge P \mid \forall a.P \mid \end{array}$$

Figure 2.12: Syntax of formulae in the While-logic.

The logical expressions E, consist of integers, auxiliary and logical variables and operations on expressions. These logical expressions are a superset of the program language expressions. The formula P, consist of the Boolean constants truth and falsity, predicates on expressions, negation, conjunction and universal quantification which quantifies over integers. The standard logical definitions are normally included as follows: disjunction $A \vee B \stackrel{def}{=} \neg(\neg A \wedge \neg B)$, implication $A \rightarrow B \stackrel{def}{=} A \vee \neg B$, if and only if $A \leftrightarrow B \stackrel{def}{=} A \rightarrow B \wedge B \rightarrow A$, existential quantification $\exists a.A \stackrel{def}{=} \neg\forall a.\neg A$.

Substitution in the logic $P[E/X]$ replaces all occurrences of variable X in P with expression E, which is defined inductively as expected.

The proof rules of propositional rules can be used to reason about these formulae. These consist of the axioms or axiom schemas (referred to simply as axioms from here on) of propositional logic and F.O.L., some of which are seen in Fig. 2.13 and Fig. 2.18 respectively, where the formulae are represented by A, B and C instead of P and Q to coincide with later logics. These axioms are not discussed here but are widely used and are introduced as they are required in later logics. These are not full lists of axioms and also not minimal as some axioms are derivable from other axioms.

Partial correctness Hoare triples $\{P\}M\{Q\}$, state that whenever the program M is executed in a state satisfying the pre-condition P, then **if** M terminates **then** the state in which M terminates satisfies the post-condition Q. Partial correctness states nothing about the case where M is non-terminating. Total correctness Hoare triples $[P]\,M\,[Q]$, state that whenever the program M is executed in a state satisfying the pre-condition P then M terminates and the state in which M terminates satisfies the post-condition Q.

The proof rules of Hoare logic are rules of inference that allow for the derivation of triples. The Hoare rules for the While-logic are introduced in Fig. 2.14.

(Truth)	T	$\equiv$	$\neg\mathsf{F}$	
$(\neg 1)$	$(\neg\neg A)$	$\equiv$	A	(Double negation)
$(\wedge 1)$	A	$\equiv$	$A \wedge A$	(Idempotency)
$(\vee 1)$	A	$\equiv$	$A \vee A$	(Idempotency)
$(\wedge 2)$	$A \wedge \neg A$	$\equiv$	F	(Consistency)
$(\vee 2)$	$A \vee \neg A$	$\equiv$	T	(Law of Excluded Middle)
$(\wedge 3)$	$(A \wedge B)$	$\equiv$	$(B \wedge A)$	(Commutativity)
$(\vee 3)$	$(A \vee B)$	$\equiv$	$(B \vee A)$	(Commutativity)
$(\leftrightarrow 1)$	$(A \leftrightarrow B)$	$\equiv$	$(B \leftrightarrow A)$	(Commutativity)
$(\wedge 4)$	$(A \wedge (B \wedge C))$	$\equiv$	$((A \wedge B) \wedge C)$	(Distributivity of $\wedge$)
$(\vee 4)$	$(A \vee (B \vee C))$	$\equiv$	$((A \vee B) \vee C)$	(Distributivity of $\vee$)
$(\wedge 6)$	$A \vee (B \wedge C)$	$\equiv$	$(A \vee B) \wedge (A \vee C)$	(Associativity of $\wedge$)
$(\vee 6)$	$A \wedge (B \vee C)$	$\equiv$	$(A \wedge B) \vee (A \wedge C)$	(Associativity of $\vee$)
$(\wedge\mathsf{T}1)$	$A \wedge \mathsf{T}$	$\equiv$	A	(Identity for $\wedge$)
$(\wedge\mathsf{F}1)$	$A \wedge \mathsf{F}$	$\equiv$	F	(Annihilator for $\wedge$)
$(\vee\mathsf{T}1)$	$A \vee \mathsf{T}$	$\equiv$	T	(Annihilator for $\wedge$)
$(\vee\mathsf{F}1)$	$A \vee \mathsf{F}$	$\equiv$	A	(Identity for $\wedge$)
$(\wedge 7)$	$(A \wedge (A \vee B))$	$\equiv$	A	(Absorption)
$(\vee 7)$	$(A \vee (A \wedge B))$	$\equiv$	A	(Absorption)
$(\rightarrow 1)$	$(A \rightarrow B)$	$\equiv$	$(\neg B) \rightarrow (\neg A)$	(Contrapositive)
(MP)	$A \wedge (A \rightarrow B)$	$\rightarrow$	B	(Modus Ponens)
(MT)	$\neg B \wedge (A \rightarrow B)$	$\rightarrow$	$\neg A$	(Modus Tollens)

Figure 2.13: Logical axioms (or axiom schemas) of propositional logic.

$$\frac{-}{\{P[E/X]\}X := E\{P\}}\,[\textsc{Assign}]_{Imp} \qquad \frac{\{P\}M\{Q\} \quad \{Q\}M'\{R\}}{\{P\}M; M'\{R\}}\,[\textsc{Sequence}]_{Imp}$$

$$\frac{\{P \wedge B\}M\{Q\} \quad \{P \wedge \neg B\}M'\{Q\}}{\{P\}\mathsf{if}\ B\ \mathsf{then}\ M\ \mathsf{else}\ M'\{Q\}}\,[\textsc{If}]_{Imp} \qquad \frac{\{P \wedge B\}M\{P\}}{\{P\}\mathsf{while}\ B\ \mathsf{do}\ M\{\neg B \wedge Q\}}\,[\textsc{While}]_{Imp}$$

$$\frac{P \to P' \quad \{P'\}M\{Q'\} \quad Q' \to Q}{\{P\}M\{Q\}}\,[\textsc{Conseq}]_{Imp}$$

Figure 2.14: Hoare (inference) rules of the While-logic (partial correctness).

The logic of axioms is introduced to the logic of triples via the $[\textsc{Conseq}]_{Imp}$ rule. The other rules are standard and represent the programs in logical form. The $[\textsc{Assign}]_{Imp}$ rule substitutes the expression E for the variable X in the pre-condition. Consider the simple example $\{Y = 1\}X := Y\{X = 1\}$, where $Y = 1 \equiv (X = 1)[Y/X]$ which is precisely the result of applying the $[\textsc{Assign}]_{Imp}$ rule. Both $[\textsc{If}]_{Imp}$ and $[\textsc{While}]_{Imp}$ require the Boolean B, to be a logical statement that can be used directly in pre-condition and reflect the semantics of the program. The sequentially executed commands M and M' are reasoned about in the $[\textsc{Sequence}]_{Imp}$ rule which first reasons about M and then M'.

An alternative, yet similar set of rules to Fig. 2.14 for total correctness exists with a more restrictive $[\textsc{While}]_{Imp}$ rule which ensures termination, however this is not included here. Partial and total correctness are equivalent for a language which always terminates.

A Hoare triple derived from the rules and axioms is written as $\vdash \{P\}M\{Q\}$. The triples can be given semantics in a mathematical model (similar in idea to the model in Sec. 2.2.2) and when a Hoare triple is satisfiable in every model, it is written $\vDash \{P\}M\{Q\}$. Soundness ensures that $\vdash \{P\}M\{Q\}$ implies $\vDash \{P\}M\{Q\}$, whereas completeness ensures the opposite i.e. $\vDash \{P\}M\{Q\}$ implies $\vdash \{P\}M\{Q\}$. Due to the undecidability of the underlying logical basis (i.e. F.O.L. or Peano arithmetic) the completeness in this form is not achievable. Hence, the definition is weakened to state that: if an oracle is used to prove the implications in the logic of axioms in each application of the $[\textsc{Conseq}]_{Imp}$ rule, then the logic is relatively complete.

Extensions of Hoare logic include: separation logic, reasoning about programs that manipulate pointers in various different ways and settings [59, 29, 48, 58, 11]; incorrectness logic, the ability to reason about bugs [49]; quantum Hoare logics, to reason about quantum programs [71, 69]; adding to the imperative language a constructor which generates fresh

pointer names [11]; and many more. The next section introduces a program logic for the functional language (STLC), introduced in Sec. 2.1.2.

2.2.2 Program Logics for the STLC

Since the inception of Hoare logics in 1969 a Hoare logic for functional programming languages was always a possibility. Although not the first attempt, in 2004 such a logic was introduced for the STLC which succinctly captured the desired aspects for such a logic [28]. The original paper [28], has been refined over time to a more stable syntax and semantics, introduced (in part) here.

The original Hoare logic reasoned about programs with assignable global variables hence the logic itself reasons about these variables, this is not the case for the STLC. The Hoare triple for the STLC is thus extended to contain an anchor variable which allows for the assignment of values to variables. This results in a judgement of the form $\{A\}\ M :_u \{B\}$. This is still referenced to as a Hoare triple or just triple even though there are now 4 elements to the triple. The triple states that if A holds, then M terminates to a value denoted by u then B holds. The variable u is the anchor, which cannot occur in the pre-condition A, but can occur in the post-condition B.

The program logic for the STLC in Sec. 2.1.2 is referred to here as the λ-logic. It is introduced here as it will be built upon in future program logics for languages which extend the STLC. The program logic in [28] reasons about the STLC containing integers (and operations on integers), recursion and more however these are not included here. The common abbreviation for equality of Booleans is used as $M = M' \stackrel{def}{=} \mathsf{if}\ M\ \mathsf{then}\ M'\ \mathsf{else}\ \neg M'$.

The logical syntax is introduced in Fig. 2.15 such that the types α range over units, Booleans and function types and pair types with the standard type context Γ, mapping variables to types. Expressions e range over the Boolean and unit values (constants $\mathsf{c} \in \{(), \mathsf{true}, \mathsf{false}\}$), variables ranging over $a, b, c,\ \ldots$ and pairs and projections. Formulae (or assertions) A are logical statements using expressions in various manners and are referred to by any variable A, B, C, ... Formulae consist of equality, negation, conjunction, universal quantification and evaluation formulae. Evaluation formulae allow reasoning regarding functions, as $e \bullet e' = m\{A\}$ states that the application of function e to term e' returns a value which, when denoted by the anchor variable m, satisfies A. From these logical constructors some shorthand notation is used which is common in other logics i.e. $e \neq e'$, $A \vee A$, $A \to A$, $\exists x.A$. The evaluation formula $e \bullet e' = m\{m = e''\}$ can be shortened as $e \bullet e' = e''$. The shorthand notation $\{A\}e \bullet e' = m\{B\}$, represents $A \to e \bullet e' = m\{B\}$.

The type annotations in formulae are often dropped.

$$
\begin{array}{lcl}
\alpha & ::= & \mathsf{Unit} \mid \mathsf{Bool} \mid \alpha \to \alpha \mid \alpha \times \alpha \\
\Gamma & ::= & \emptyset \mid \Gamma, x : \alpha \\
e & ::= & () \mid \mathsf{true} \mid \mathsf{false} \mid x \mid \langle e, e \rangle \mid \pi_i(e) \\
A & ::= & e = e \mid \neg A \mid A \wedge A \mid \forall x^{\alpha}.A \mid e \bullet e' = m^{\alpha}\{A\}
\end{array}
$$

Figure 2.15: Syntax of the λ-logic.

All expressions and formulae are type checked using typing judgments mimicking the type checking of STLC terms. The typing judgment for expressions, formulae, and triples are written $\Gamma \vdash e : \alpha$, $\Gamma \vdash A$, $\Gamma \vdash \{A\}\ M :_u \{B\}$ respectively. The typing rules for expressions, formulae and triples are seen in Fig. 2.16. Most logical constructs are typed as expected. The typing of evaluation formulae ensures the type of the function is that which: accepts the type of the expression it is applied to, and outputs the type of the anchor. The typing of triples adds the anchor of the type of the term being reasoned about, to the type context to type check the post-condition. All other typing rules are elementary.

$$
\frac{b \in \{\mathsf{true}, \mathsf{false}\}}{\Gamma \vdash b : \mathsf{Bool}} \quad \frac{-}{\Gamma \vdash () : \mathsf{Unit}} \quad \frac{\Gamma(x) = \alpha}{\Gamma \vdash x : \alpha} \quad \frac{\Gamma \vdash e_1 : \alpha_1 \quad \Gamma \vdash e_2 : \alpha_2}{\Gamma \vdash \langle e_1, e_2 \rangle : \alpha_1 \times \alpha_2} \quad \frac{\Gamma \vdash e : \alpha_1 \times \alpha_2}{\Gamma \vdash \pi_i(e) : \alpha_i}
$$

$$
\frac{\Gamma \vdash e_1 : \alpha \quad \Gamma \vdash e_2 : \alpha}{\Gamma \vdash e_1 = e_2} \quad \frac{\Gamma \vdash e : \alpha_1 \to \alpha_2 \quad \Gamma \vdash e' : \alpha_1 \quad \Gamma, x : \alpha_2 \vdash A}{\Gamma \vdash e \bullet e' = x^{\alpha_2}\{A\}}
$$

$$
\frac{\Gamma \vdash A_1 \quad \Gamma \vdash A_2}{\Gamma \vdash A_1 \wedge A_2} \quad \frac{\Gamma \vdash A}{\Gamma \vdash \neg A} \quad \frac{\Gamma, x : \alpha \vdash A}{\Gamma \vdash \forall x^{\alpha}.A} \quad \frac{\Gamma \vdash A \quad \Gamma \vdash M : \alpha \quad \Gamma, m : \alpha \vdash B}{\Gamma \vdash \{A\}\ M :_m \{B\}}
$$

Figure 2.16: Typing rules for expressions, formulae and triples in the λ-logic.

The notion of free variables of terms is extended to define the free variables of expressions and formulae. All variables occurring in expressions and those that are not bound by a quantifier or the anchor in an evaluation formulae, are free.

Definition 11 (Free variables in expressions and formulae)**.** *The free variables of an ex-*

pression e and formula A, written $\mathsf{fv}(e)$ and $\mathsf{fv}(A)$ respectively, are defined as follows.

$$\begin{array}{rcll}
\mathsf{fv}(\mathsf{c}) & \stackrel{\text{def}}{=} & \emptyset & \mathsf{c} \in \{(), \mathsf{true}, \mathsf{false}\} \\
\mathsf{fv}(x) & \stackrel{\text{def}}{=} & \{x\} & \\
\mathsf{fv}(\langle e_1, e_2 \rangle) & \stackrel{\text{def}}{=} & \mathsf{fv}(e_1) \cup \mathsf{fv}(e_2) & \\
\mathsf{fv}(\pi_i(e)) & & \mathsf{fv}(e) &
\end{array}$$

$$\begin{array}{rcl}
\mathsf{fv}(e_1 = e_2) & \stackrel{\text{def}}{=} & \mathsf{fv}(e_1) \cup \mathsf{fv}(e_2) \\
\mathsf{fv}(\neg A) & \stackrel{\text{def}}{=} & \mathsf{fv}(A) \\
\mathsf{fv}(A \wedge B) & \stackrel{\text{def}}{=} & \mathsf{fv}(A) \cup \mathsf{fv}(B) \\
\mathsf{fv}(e_1 \bullet e_2 = m\{A\}) & \stackrel{\text{def}}{=} & \mathsf{fv}(e_1) \cup \mathsf{fv}(e_2) \cup (\mathsf{fv}(A) \setminus \{m\}) \\
\mathsf{fv}(\forall x^{\alpha}.A) & \stackrel{\text{def}}{=} & \mathsf{fv}(A) \setminus \{x\}
\end{array}$$

A variable x is not in A, written $A^{\text{-}x}$, if x does not occur in the free variables of A.

Logical substitution is defined similarly to that of Hoare logic, with the exception that variables now only appear in expressions, hence substitution is defined on expressions and formulae as follows.

Definition 12 (Logical substitution)**.** *Substitution of an expression e for a variable x in an expression e' is written $e'[e/x]$, whilst the same substitution in a formula A is written $A[e/x]$. Both are defined as follows assuming $x \neq y$.*

$$\begin{array}{rcll}
\mathsf{c}[e/x] & \stackrel{\text{def}}{=} & \mathsf{c} & \\
x[e/x] & \stackrel{\text{def}}{=} & e & \\
y[e/x] & \stackrel{\text{def}}{=} & y & (x \neq y) \\
(e_1 = e_2)[e/x] & \stackrel{\text{def}}{=} & (e_1[e/x]) = (e_2[e/x]) & \\
\langle e_1, e_2 \rangle[e/x] & \stackrel{\text{def}}{=} & \langle e_1[e/x], e_2[e/x] \rangle & \\
\pi_i(e_1)[e/x] & \stackrel{\text{def}}{=} & \pi_i(e_1[e/x]) & \\
(\neg A)[e/x] & \stackrel{\text{def}}{=} & \neg(A[e/x]) & \\
(A \wedge B)[e/x] & \stackrel{\text{def}}{=} & A[e/x] \wedge B[e/x] & \\
(\forall x^{\alpha}.A)[e/x] & \stackrel{\text{def}}{=} & \forall x^{\alpha}.A & \\
(\forall y^{\alpha}.A)[e/x] & \stackrel{\text{def}}{=} & \forall y^{\alpha}.(A[e/x]) & x \neq y,\ y \notin \mathsf{fv}(e) \\
(\forall y^{\alpha}.A)[e/x] & \stackrel{\text{def}}{=} & (\forall z^{\alpha}.(A[z/y]))[e/x] & x \neq y,\ y \in \mathsf{fv}(e), z - \textit{fresh} \\
(e_1 \bullet e_2 = m\{A\})[e/x] & \stackrel{\text{def}}{=} & e_1[e/x] \bullet e_2[e/x] = m\{A[e/x]\} & x \neq m,\ m \notin \mathsf{fv}(e) \\
(e_1 \bullet e_2 = m\{A\})[e/x] & \stackrel{\text{def}}{=} & (e_1 \bullet e_2 = u\{A[u/m]\})[e/x] & x \neq m,\ m \in \mathsf{fv}(e), u - \textit{fresh}
\end{array}$$

In a similar fashion to the program logic in Sec. 2.2.1 the logic can be split into the logic of axioms and the logic of rules. The logic of axioms consisting of the axioms and

axiom schemas (referred to simply as axioms) include the axioms of predicate logic seen in Fig. 2.13, the axioms of equality seen in Fig. 2.17, axioms of F.O.L. which can be seen in Fig. 2.18, and axioms of evaluation formulae seen in Fig. 2.19. Most axioms are standard with the axioms of equality representing reflexivity, symmetry and transitivity, and substitution.

$$
\begin{array}{lrcl}
(eq1)_\lambda & e = e & & \\
(eq2)_\lambda & e = e' & \leftrightarrow & e' = e \\
(eq3)_\lambda & e = e' \wedge e' = e'' & \rightarrow & e = e'' \\
(eq4)_\lambda & x = e \wedge A & \rightarrow & A[e/x]
\end{array}
$$

Figure 2.17: Axioms for equality of the λ-logic.

An axiom for pairs is often not included, but is the following tautology.

$$(p1) \qquad \pi_i(\langle e_1, e_2 \rangle) = e_i$$

The axioms of F.O.L. are standard from [38] in Fig. 2.18. This list is not minimal as some axioms are derivable from others, but included for convenience. Later the universal quantifier will be adapted with the relevant adaptions to these axioms.

Evaluation formula axioms in Fig. 2.19 originate from how functions interact, expressed in logical form. These axioms are for the terminating STLC language without recursion. Some axioms differ when non-terminating features are added. Axiom $(ext)_\lambda$ requires $\mathsf{Ext}(e, e') \stackrel{def}{=} \forall x. e \bullet x = m\{e' \bullet x = n\{m = n\}\}$, and implies extensionality as the functions e and e' cannot be distinguished through application. Axioms $(e1)$, $(e2)$ and $(e3)$ show conjunction, negation and invariance, within the evaluation formulae. Axiom $(e4)$ shows how evaluation formulae interact with universal quantification under certain restrictions, these ensure the quantified variable does not interact with the evaluation formula variables. Axiom (e_α) is essentially the idempotency of evaluation formulae combined with α-equivalence.

$(u1)_\lambda$	$\forall x^\alpha.A$	$\rightarrow$	$A[e/x]$	(Instantiation)
$(u2)_\lambda$	$x \notin \mathsf{fv}(A) \rightarrow (A$	$\leftrightarrow$	$\forall x^\alpha.A)$	(Vacuous Generalisation/Instantiation)
$(u3)_\lambda$	$\forall x^\alpha.A \wedge \forall x^\alpha.B$	$\leftrightarrow$	$\forall x^\alpha.A \wedge B$	(Distribution)
$(ex1)_\lambda$	$A[e/x]$	$\rightarrow$	$\exists x.A$	(Existential Generalization)
$(ex2)_\lambda$	$A^{-x} \wedge \exists x.B$	$\leftrightarrow$	$\exists x.(A \wedge B)$	(Derivable)
$(ud1)_\lambda$	$\forall x^{\alpha_1}.\forall y^{\alpha_2}.A$	$\leftrightarrow$	$\forall y^{\alpha_2}.\forall x^{\alpha_1}.A$	(Derivable)
$(ud2)_\lambda$	$\exists x^{\alpha_1}.\exists y^{\alpha_2}.A$	$\leftrightarrow$	$\exists y^{\alpha_2}.\exists x^{\alpha_1}.A$	(Derivable)
$(ud3)_\lambda$	$\exists x^\alpha.A \vee \exists x^\alpha.B$	$\leftrightarrow$	$\exists x^\alpha.A \vee B$	(Derivable)
$(ud4)_\lambda$	$\exists x^{\alpha_1}.\forall y^{\alpha_2}.A$	$\rightarrow$	$\forall y^{\alpha_2}.\exists x^{\alpha_1}.A$	(Derivable)
$(ud5)_\lambda$	$\forall x^\alpha.(A \rightarrow B)$	$\rightarrow$	$(\forall x^\alpha.A) \rightarrow (\forall x^\alpha.B)$	(Derivable)

Figure 2.18: Axioms for first order logic of the λ-logic.

$(e1)_\lambda$	$e \bullet e' = m\{A\} \wedge e \bullet e' = m\{B\}$	$\leftrightarrow$	$e \bullet e' = m\{A \wedge B\}$	
$(e2)_\lambda$	$e \bullet e' = m\{\neg A\}$	$\leftrightarrow$	$\neg e \bullet e' = m\{A\}$	
$(e3)_\lambda$	$e \bullet e' = m\{A \wedge B\}$	$\leftrightarrow$	$A \wedge e \bullet e' = m\{B\}$	$m \notin \mathsf{fv}(A)$
$(e4)_\lambda$	$e \bullet e' = m\{\forall a^\alpha.A\}$	$\leftrightarrow$	$\forall a^\alpha.e \bullet e' = m\{A\}$	$a \notin \mathsf{fv}(e, e', m)$
$(ext)_\lambda$	$e = e'$	$\leftrightarrow$	$\mathsf{Ext}(e, e')$	$e, e' : \alpha_1 \rightarrow \alpha_2$
$(e_\alpha)_\lambda$	$e \bullet e' = m\{A\}$	$\leftrightarrow$	$e \bullet e' = m\{e \bullet e' = a\{a = m \wedge A\}\}$	
				$a \notin \mathsf{fv}(e, e'), a \in \mathsf{fv}(A) \rightarrow a \equiv m$

Figure 2.19: Axioms for evaluation formulae of the λ-logic.

The logic of rules is introduced via rules based on the programming language constructors in Fig. 2.20. The rules intend to represent the behaviour of programs within the logic. All rules are well typed, with certain formulae containing requirements such as C^{-m} to mean $m \notin \mathsf{fv}(C)$, which ensures the typing conditions are met.

The $[\textsc{Lam}]_\lambda$ rule states $\forall x.(B \rightarrow u \bullet x = m\{C\})$ in the post-condition of the conclusion, meaning for any value x such that B holds, if u (i.e. the $\lambda x.M$ value) is applied to x and denoted by m then C holds. This requires the assumption, which states a similar intention

but reasons about the term M and uses the equivalence of M and $(\lambda x.M)x$.

The $[\textsc{App}]_\lambda$ rule requires M and N to be reasoned about individually, resulting in $m \bullet n = u\{C^{-m,n}\}$ where m and n represent M and N, respectively. This states: applying m to n and denoting the result by u implies C holds, which is precisely the required meaning in the conclusion of the rule, where MN is denoted by u. Requiring $C^{-m,n}$ ensures m and n do not occur freely in C, satisfying the type checking of the rule.

The $[\textsc{Var}]_\lambda$ and $[\textsc{Const}]_\lambda$ rules are similar to the $[\textsc{Assign}]_{Imp}$ rule if the assigned variable is considered as the anchor. They require the substitution in the pre-condition to ensure the intention is captured correctly.

The $[\textsc{Neg}]_\lambda$ assumes $\neg m$ as an expression and substitutes it for u in the postcondition of the assumption, ensuring u in the conclusion is the negation of m in the assumption.

The $[\textsc{If}]_\lambda$ rule differs from $[\textsc{If}]_{Imp}$, as the Boolean condition in the λ-logic is not a formula in the logic, hence must be reasoned about first. The substitution of both Boolean constants for b indicates the two possible cases for the conditional and hence reason about the two respective programs.

The $[\textsc{Pair}]_\lambda$ and $[\textsc{Proj}_i]_\lambda$ rules express the idea of pairs and projections in the assumption via their substitution for u, whilst in the conclusion the same pair or projection programs are represented by u. The $[\textsc{Let}]_\lambda$ rule can be derived from the definition $\mathsf{let}\ x^\alpha = M\ \mathsf{in}\ N \stackrel{def}{=} (\lambda x^\alpha.N)M$ and the relevant rules, but is included for convenience.

The $[\textsc{Eq}]_\lambda$ rule reasons about M and N, and then requires the result to be in the form that substitutes $m = n$ for u in C. This implies that the result of equating M and N can be substituted for u which indicates precisely the conclusion. It is common to write $C[m = n/u]$ in the $[\textsc{Eq}]_\lambda$ rule, even though $m = n$ is not an expression, but this is shorthand for substituting $u = \mathsf{true}$ for $m = n$ in C. The expression $e = e'$ could be included to make this syntax correct but is not included for simplicity and brevity, as is common in the literature [28, 72, 6]. It is also possible to set the evaluation formula as an expression, as is done in [28]. However, this transfers some complications from the axioms to the manipulation of expressions. This balance of where to focus the reasoning is a matter of preference and fashion, but has no computational benefits except readability.

$$\frac{-}{\{A[x/m]\}\ x :_m \{A\}}\ [\textsc{Var}]_\lambda \qquad \frac{\{A\}\ M :_m \{B\} \quad \{B\}\ N :_n \{C^{-m,n}[m = n/u]\}}{\{A\}\ M = N :_u \{C\}}\ [\textsc{Eq}]_\lambda$$

$$\frac{-}{\{A[\mathsf{c}/m]\}\ \mathsf{c} :_m \{A\}}\ [\textsc{Const}]_\lambda \qquad \frac{\{A^{-x} \wedge B\}\ M :_m \{C\}}{\{A\}\ \lambda x^\alpha.M :_u \{\forall x^\alpha.(B \to u \bullet x = m\{C\})\}}\ [\textsc{Lam}]_\lambda$$

$$\frac{\{A\}\ M :_m \{B\} \quad \{B\}\ N :_n \{m \bullet n = u\{C^{-m,n}\}\}}{\{A\}\ MN :_u \{C\}}\ [\textsc{App}]_\lambda \qquad \frac{\{A\}\ M :_m \{C[\neg m/u]\}}{\{A\}\ \neg M :_u \{C\}}\ [\textsc{Neg}]_\lambda$$

$$\frac{\{A\}\ M :_m \{B\} \quad \{B[b_i/m]\}\ N_i :_u \{C^{-m}\} \quad b_1 = \mathsf{true} \quad b_2 = \mathsf{false} \quad i = 1,2}{\{A\}\ \mathsf{if}\ M\ \mathsf{then}\ N_1\ \mathsf{else}\ N_2 :_u \{C\}}\ [\textsc{If}]_\lambda$$

$$\frac{\{A\}\ M :_m \{B\} \quad \{B\}\ N :_n \{C^{-m,n}[\langle m,n\rangle/u]\}}{\{A\}\ \langle M,N\rangle :_u \{C\}}\ [\textsc{Pair}]_\lambda \qquad \frac{\{A\}\ M :_m \{C^{-m}[\pi_i(m)/u]\}}{\{A\}\ \pi_i(M) :_u \{C\}}\ [\textsc{Proj}_i]_\lambda$$

$$\frac{\{A\}\ M :_x \{B\} \quad \{B\}\ N :_u \{C^{-x}\}}{\{A\}\ \mathsf{let}\ x = M\ \mathsf{in}\ N :_u \{C\}}\ [\textsc{Let}]_\lambda$$

Figure 2.20: Inference rules of the λ-logic.

Structural rules introduced in Fig. 2.21 allow for the manipulation of any triple under specific circumstances. The $[\textsc{Conseq}]_\lambda$ rule introduces the logic of axioms to the triples in both the pre-condition and post-condition. The other rules allow for combining two triples of similar form $[\wedge\text{-}\textsc{Post}]_\lambda$ and $[\vee\text{-}\textsc{Pre}]_\lambda$ or allow certain formulae to be moved from the pre-condition to the post-condition or vice versa under certain conditions in $[\wedge \to]_\lambda$, $[\to \wedge]_\lambda$ and $[\textsc{Invar}]_\lambda$.

$$\frac{\{A\}\ M :_u \{B\} \quad \{A\}\ M :_u \{C\}}{\{A\}\ M :_u \{B \wedge C\}}\ [\wedge\text{-}\textsc{Post}]_\lambda \qquad \frac{\{A\}\ M :_u \{C\} \quad \{B\}\ M :_u \{C\}}{\{A \vee B\}\ M :_u \{C\}}\ [\vee\text{-}\textsc{Pre}]_\lambda$$

$$\frac{\{A \wedge B\}\ M :_u \{C\}}{\{A\}\ M :_u \{B \to C\}}\ [\wedge \to]_\lambda \qquad \frac{\{A\}\ M :_u \{B^{-u} \to C\}}{\{A \wedge B\}\ M :_u \{C\}}\ [\to \wedge]_\lambda$$

$$\frac{\{A\}\ M :_u \{B\}}{\{A \wedge C\}\ M :_u \{B \wedge C\}}\ [\textsc{Invar}]_\lambda \qquad \frac{A \to A' \quad \{A'\}\ M :_u \{B'\} \quad B' \to B}{\{A\}\ M :_u \{B\}}\ [\textsc{Conseq}]_\lambda$$

Figure 2.21: Structural inference rules of the λ-logic.

Similar to the While-logic, if a triple $\{A\}\ M :_u \{B\}$ can be derived from the rules and axioms, this is written $\vdash \{A\}\ M :_u \{B\}$. Well typed terms in this STLC always terminate, meaning this program logic is partially correct if and only if it is totally correct. However, the inclusion of non-termination into the language (and rules) means this no longer holds as in [28] which contains recursion.

Example 1. The simple example of the identity function applied to true which should clearly output true is proven below, using this logic. The resulting triple in line 6 states that in any valid initial state (represented by T), if the program $(\lambda x^{\mathsf{Bool}}.x)\mathsf{true}$ is run and denoted by the variable u then u must be equivalent to true, which is exactly what is required.

1	$\{x = x\}\ x :_u \{u = x\}$	$[\textsc{Var}]_\lambda$
2	$\{\mathsf{T}\}\ \lambda x^{\mathsf{Bool}}.x :_m \{\forall x^{\mathsf{Bool}}.m \bullet x = u\{u = x\}\}$	$[\textsc{Lam}]_\lambda$
3	$\forall x^{\mathsf{Bool}}.m \bullet x = u\{u = x\} \ \rightarrow\ m \bullet \mathsf{true} = u\{u = \mathsf{true}\}$	$(u1)_\lambda$
4	$\{\mathsf{T}\}\ \lambda x^{\mathsf{Bool}}.x :_m \{m \bullet \mathsf{true} = u\{u = \mathsf{true}\}\}$	$[\textsc{Conseq}]_\lambda$, *2, 3*
5	$\{m \bullet \mathsf{true} = u\{\mathsf{true} = u\}\}\ \mathsf{true} :_n \{m \bullet n = u\{u = \mathsf{true}\}\}$	$[\textsc{Const}]_\lambda$
6	$\{\mathsf{T}\}\ (\lambda x^{\mathsf{Bool}}.x)\mathsf{true} :_u \{u = \mathsf{true}\}$	$[\textsc{App}]_\lambda$, *4, 5*

Example 2. Reasoning about the program $\lambda x^{\mathsf{Bool}}.\mathsf{if}\ x\ \mathsf{then}\ \mathsf{false}\ \mathsf{else}\ \mathsf{true}$ which should

essentially return $\neg x$ can be proven as follows.

1	$\{x = \mathsf{true}\}\ x :_d \{x = \mathsf{true} = d\}$	$[\textsc{Var}]_\lambda$
2	$\{(x = \mathsf{true} = d)[\mathsf{true}/d]\}\ \mathsf{false} :_b \{b = \mathsf{false}\}$	$[\textsc{Const}]_\lambda$
3	$\{(x = \mathsf{true} = d)[\mathsf{false}/d]\}\ \mathsf{true} :_b \{\mathsf{F}\}$	$[\textsc{Const}]_\lambda$
4	$\{x = \mathsf{true}\}\ \mathsf{if}\ x\ \mathsf{then\ false\ else\ true} :_b \{b = \mathsf{false}\}$	$[\textsc{If}]_\lambda$, *1, 2, 3*
5	$\{\mathsf{T}\}\ \mathsf{if}\ x\ \mathsf{then\ false\ else\ true} :_b \{x = \mathsf{true} \to b = \mathsf{false}\}$	$[\wedge \to]_\lambda$, *4*
6	$\{x = \mathsf{false}\}\ x :_d \{x = \mathsf{false} = d\}$	$[\textsc{Var}]_\lambda$
7	$\{(x = \mathsf{false} = d)[\mathsf{false}/d]\}\ \mathsf{false} :_b \{\mathsf{F}\}$	$[\textsc{Const}]_\lambda$
8	$\{(x = \mathsf{false} = d)[\mathsf{true}/d]\}\ \mathsf{true} :_b \{b = \mathsf{true}\}$	$[\textsc{Const}]_\lambda$
9	$\{x = \mathsf{false}\}\ \mathsf{if}\ x\ \mathsf{then\ false\ else\ true} :_b \{b = \mathsf{true}\}$	$[\textsc{If}]_\lambda$, *6, 7, 8*
10	$\{\mathsf{T}\}\ \mathsf{if}\ x\ \mathsf{then\ false\ else\ true} :_b \{x = \mathsf{false} \to b = \mathsf{true}\}$	$[\wedge \to]_\lambda$, *9*
11	$\{\mathsf{T}\}\ \mathsf{if}\ x\ \mathsf{then\ false\ else\ true} :_b \{x = \neg b\}$	$[\wedge\text{-}\textsc{Post}]_\lambda$, *5, 10*
12	$\{\mathsf{T}\}\ \lambda x^{\mathsf{Bool}}.\mathsf{if}\ x\ \mathsf{then\ false\ else\ true} :_a \{\forall x^{\mathsf{Bool}}.a \bullet x = \neg x\}$	$[\textsc{Lam}]_\lambda$, *11*

Example 3. The program $(\lambda f^{\mathsf{Bool}\to\mathsf{Bool}}.f\mathsf{true})(\lambda x^{\mathsf{Bool}}.\mathsf{false})$ which should return false is reasoned about as follows.

1	Let $B(f) \equiv f \bullet \mathsf{true} = \mathsf{false}$	
2	Let $D(c) \equiv \forall f.B(f) \to c \bullet f = \mathsf{false}$	
3	$\{B(f)\}\ f :_g \{B(g)\}$	$[\textsc{Var}]_\lambda$
4	$\{B(g)\}\ \mathsf{true} :_h \{g \bullet h = \mathsf{false}\}$	$[\textsc{Const}]_\lambda$
5	$\{B(f)\}\ f\mathsf{true} :_a \{a = \mathsf{false}\}$	$[\textsc{App}]_\lambda$, *3, 4*
6	$\{\mathsf{T}\}\ \lambda f.f\mathsf{true} :_c \{D(c)\}$	$[\textsc{Lam}]_\lambda$, *5*
7	$\{\mathsf{T}\}\ \mathsf{false} :_a \{a = \mathsf{false}\}$	$[\textsc{Const}]_\lambda$
8	$\{\mathsf{T}\}\ \lambda x^{\mathsf{Bool}}.\mathsf{false} :_f \{\forall x.f \bullet x = \mathsf{false}\}$	$[\textsc{Lam}]_\lambda$, *7*
9	$\{\mathsf{T}\}\ \lambda x^{\mathsf{Bool}}.\mathsf{false} :_f \{B(f)\}$	$[\textsc{Conseq}]_\lambda$, $(u1)_\lambda$, *8*
10	$\{D(c)\}\ \lambda x.\mathsf{false} :_f \{D(c) \wedge B(f)\}$	$[\textsc{Invar}]_\lambda$, *9*
11	$\{D(c)\}\ \lambda x.\mathsf{false} :_f \{c \bullet f = \mathsf{false}\}$	$[\textsc{Conseq}]_\lambda$, $(u1)_\lambda$, *M.P., 10*
12	$\{\mathsf{T}\}\ (\lambda f^{\mathsf{Bool}\to\mathsf{Bool}}.f\mathsf{true})(\lambda x^{\mathsf{Bool}}.\mathsf{false}) :_a \{a = \mathsf{false}\}$	$[\textsc{App}]_\lambda$, *6, 11*

In Chapt. 7 these examples will be reasoned about again using the ν-logic, showing that an equivalent triple is derivable.

Model

The logic is given semantics by first defining a model and then defining a satisfaction relation for triples. This requires an interpretation of expressions and the semantics of formulae in the model. The model and semantics are introduced here for the λ-logic to be built upon in Chapt. 5 to model the ν-logic.

A model ξ, is defined as for the While-logic, as an unordered mapping between variables and closed values. The value mapped to by x in a model ξ is written $\xi(x)$. The domain of a model ξ is the list of variables from which the model maps to values, and the codomain (range) of the model is the list of values to which the model maps, $\mathsf{dom}(\xi)$ and $\mathsf{cod}(\xi)$ respectively. A model ξ is typed by Γ (or is a model of type Γ) if Γ and ξ have the same domains and for each variable x in the domain then $\xi(x) : \Gamma(x)$. The model is used to interpret expressions as programs in Def. 13. The semantics (or interpretation) of formulae

are given by Def. 14. This leads to the modelling of triples in Def. 16 which are used to prove soundness of the rules.

Definition 13 (Interpretation of expressions)**.** *The interpretations of the expression e in the model ξ written $[\![e]\!]_\xi$ are defined as follows.*

$$\begin{array}{rcl@{\qquad}rcl} [\![()]\!]_\xi & \stackrel{\text{def}}{=} & () & [\![\mathsf{true}]\!]_\xi & \stackrel{\text{def}}{=} & \mathsf{true} \\ [\![x]\!]_\xi & \stackrel{\text{def}}{=} & \xi(x) & [\![\mathsf{false}]\!]_\xi & \stackrel{\text{def}}{=} & \mathsf{false} \\ [\![\langle e_1, e_2\rangle]\!]_\xi & \stackrel{\text{def}}{=} & \langle [\![e_1]\!]_\xi, [\![e_2]\!]_\xi\rangle & [\![\pi_i(e)]\!]_\xi & \stackrel{\text{def}}{=} & \pi_i([\![e]\!]_\xi) \end{array}$$

Definition 14 (Semantics of formulae)**.** *A model ξ interprets a formula A written $\xi \models A$, also referred to as semantics of formulae or ξ models A, is defined as follows.*

$$\begin{array}{rcl} \xi \models e = e' & \stackrel{\text{def}}{=} & [\![e]\!]_\xi \cong_\alpha^{STLC} [\![e']\!]_\xi \\ \xi \models \neg A & \stackrel{\text{def}}{=} & \xi \not\models A \\ \xi \models A \wedge B & \stackrel{\text{def}}{=} & \xi \models A \ \wedge\ \xi \models B \\ \xi \models e \bullet e' = m\{A\} & \stackrel{\text{def}}{=} & [\![e]\!]_\xi [\![e']\!]_\xi \Downarrow V \ \wedge\ \xi \cdot m : V \models A \\ \xi \models \forall x^\alpha . A & \stackrel{\text{def}}{=} & \forall V^\alpha . \xi \cdot x : V \models A \end{array}$$

The semantics for equivalence requires the interpretation of expressions to be contextually equivalent as per Def. 10. The semantics for the evaluation formulae is for total correctness as the language is strongly-normalizing, however if non-termination exists in the language this can be redefined for partial correctness. The semantics of universal quantification, quantifies over all possible values of the correct type and assigns them to the quantified variable in the model to satisfy the formula A. The semantics of other formulae are standard.

It is often referenced that an open term (or expression) is evaluated, this is short for the closure of said term by a model is evaluated. The model must be typed by the same type context that types the term, ensuring all free variables in the term are mapped in the model.

Definition 15 (Closure of terms)**.** *Model ξ closes the open program M written $M\xi$ defined as follows, where ξ^{-x} is the model ξ with the mapping for x removed assuming it exists.*

$$\begin{array}{rcl@{\qquad}rcl} ()\xi & \stackrel{\text{def}}{=} & () & (\lambda x^\alpha . M)\xi & \stackrel{\text{def}}{=} & \lambda x^\alpha . (M\xi^{-x}) \\ \mathsf{true}\xi & \stackrel{\text{def}}{=} & \mathsf{true} & (MN)\xi & \stackrel{\text{def}}{=} & (M\xi)(N\xi) \\ \mathsf{false}\xi & \stackrel{\text{def}}{=} & \mathsf{false} & \langle M, N\rangle\xi & \stackrel{\text{def}}{=} & \langle M\xi, N\xi\rangle \\ x\xi & \stackrel{\text{def}}{=} & \xi(x) & (\pi_i(M))\xi & \stackrel{\text{def}}{=} & \pi_i(M\xi) \\ & & & (\mathsf{let}\ x = M\ \mathsf{in}\ N)\xi & \stackrel{\text{def}}{=} & \mathsf{let}\ x = M\xi\ \mathsf{in}\ N\xi^{-x} \end{array}$$

The previous three definitions are required to define the semantics of triples.

Definition 16 (Semantics of triples)**.** *The semantics of triples mimics the description of the Hoare triples with the model closing the free variables in the term within the triple.*

$$\xi \models \{A\}\ M :_u \{B\} \quad \stackrel{\text{def}}{=} \quad \xi \models A \ \rightarrow\ \exists V.(M\xi \Downarrow V \ \wedge\ \xi \cdot u : V \models B)$$

If $\Gamma \vdash \{A\}\ M :_u \{B\}$*, write* $\models \{A\}\ M :_u \{B\}$ *if* $\forall \xi^{\Gamma}.\ \xi \models \{A\}\ M :_u \{B\}$.

Definition 17 (Soundness and completenesses)**.** *The* λ*-logic is defined as:*

- Sound *if* $\vdash \{A\}\ M :_u \{B\}$ *implies* $\models \{A\}\ M :_u \{B\}$
- Complete *if* $\models \{A\}\ M :_u \{B\}$ *implies* $\vdash \{A\}\ M :_u \{B\}$
  However as with the While-logic, the incompleteness of F.O.L. [23] makes completeness impossible and hence different notions of completeness are introduced [14, 26].

The λ-logic presented above (and in [28]) is sound, and various notions of completeness are all proven for extensions to the STLC with recursion [26] and also aliasing and local state [4], hence these results hold for the smaller STLC presented here.

2.2.3 Program Logic for Higher-Order Functions with Local State

Names are an abstraction of many concepts (see Sec. 1.1), one of which is local state. The addition of local state to the STLC is reasoned about by a program logic introduced in [72]. This logic is introduced here due to the similarities of local state to the ν-calculus.

The full language in [72] is complex with a store and the ability to generate fresh references to the store. The program $\mathsf{ref}(M)$ generates and returns a reference, and fills it with the value M evaluates to. The references denoted by M can be dereferenced by $!M$ and assigned a new value M' evaluates to written $M := M'$. Take for example the function $\mathsf{let}\ x = \mathsf{ref}(0)\ \mathsf{in}\ \lambda().x :=!x + 1; !x$ where "$\mathsf{ref}(0)$" produces a new reference (or location) to the store which contains the value 0. This reference is assigned to the variable x in $\mathsf{let}\ x = \mathsf{ref}(0)\ \mathsf{in}$ ... which allows the reference to be used in the ... part via the use of x. When the function $\lambda().x :=!x + 1; !x$, is applied (to unit) it will increment the value stored at the reference x, and return the new value.

The inclusion of a store and fresh reference generation has many useful applications. However, here the restriction of the logic is to only contain the STLC extended with the $\mathsf{ref}()$ constructor. The syntax is extended from the STLC as seen in Fig. 2.22. The $\mathsf{ref}()$ function produces a fresh reference each time it is evaluated, with no distinguishing content stored at the location, i.e. (). This is in essence the generation of fresh names. The

programming language is referred to here as the Local-STLC. The program logic in [72] is adapted and introduced here for this very minimal extension to the STLC and described below. This program logic is referred to here as the Local-logic.

$$\begin{array}{lcl} \alpha & ::= & \ldots \mid \mathsf{Ref}(\mathsf{Unit}) \\ M & ::= & \ldots \mid \mathsf{ref}() \mid M = M \\ V & ::= & \ldots \mid l \end{array}$$

Figure 2.22: New syntax of the Local-STLC extending the STLC syntax.

The new type $\mathsf{Ref}(\mathsf{Unit})$ represents the type of memory location storing () however this is essentially the type Nm. The $\mathsf{ref}()$ term constructor of locations returns a fresh location (or reference) which ranges over l, l', l'', ... (or sometimes l_i) which cannot be used directly in the language and whose content cannot be accessed without access to the specific location. The equality constructor is also included as the destructor for references. The typing rules for the new constructors are those of Fig. 2.23.

$$\frac{-}{\Gamma \vdash \mathsf{ref}() : \mathsf{Ref}(\mathsf{Unit})} \qquad \frac{\Gamma \vdash M : \mathsf{Ref}(\mathsf{Unit}) \quad \Gamma \vdash M' : \mathsf{Ref}(\mathsf{Unit})}{\Gamma \vdash M = M' : \mathsf{Bool}}$$

Figure 2.23: Typing rules of the Local-STLC.

Given the locations all store (), the concept of a store is not required and hence is not included in this version of the language. The reduction relation adds another additional construct which is that of the hiding of locations as $(\nu\tilde{l})$, where ν is a (hiding) binder of the names in the vector $\tilde{l}$, so that the CBV reduction relation is as follows $(\nu\tilde{l})M \to (\nu\tilde{l'})M'$. The evaluation contexts and reduction rules are those of the STLC in Fig. 2.8 and Fig. 2.9 respectively with the additional reduction rules defined in Fig. 2.24.

$$
\begin{array}{rcll}
\mathsf{ref}() & \to & (\nu l)(l) & \\
l = l & \to & \mathsf{true} & \\
l_1 = l_2 & \to & \mathsf{false} & (l_1 \neq l_2) \\
M \to M' & \to & (\nu \tilde{l})M \to (\nu \tilde{l})M' & \\
(\nu \tilde{l_1})M \to (\nu \tilde{l_2})M' & \to & (\nu \tilde{l} \tilde{l_1})\mathcal{E}[M] \to (\nu \tilde{l} \tilde{l_2})\mathcal{E}[M'] &
\end{array}
$$

Figure 2.24: Reduction rules of the Local-STLC.

Contextual equivalence of configurations are defined as standard, requiring all single-holed contexts filled with the terms to evaluate to the same Boolean.

The syntax of the logical expressions and formulae build on the λ-logic in Fig. 2.25.

$$
\begin{array}{rcl}
e & ::= & x \mid c \\
A & ::= & e = e' \mid \neg A \mid A \wedge A \mid \forall x^{\alpha}.A \mid \nu x.A \mid e \bullet e' = m\{A\} \mid \Box A \mid e \hookrightarrow e'
\end{array}
$$

Figure 2.25: Syntax of the Local-logic.

The expressions build on the λ-logic with constants c now including locations l. This may be a mistake in [72], given the language does not have direct access to locations, however it is included here.

Formulae are extended from λ-logic by first adding the new hiding quantifier $\nu x.A$, which means for some hidden reference x then A holds, with its dual $\bar{\nu} x.A \stackrel{def}{=} \neg \nu x.\neg A$, meaning for all hidden references x then A holds. The ν quantifier sits between $\forall$ and $\exists$ quantifiers in quantifying power given $\exists x.\forall y.A \to \forall y.\exists x.A$ and $\nu x.\forall y.A \to \forall y.\nu x.A$ but $\exists x.\nu y.A \to \nu y.\exists x.A$.

The modal quantifiers $\Box A$ and its dual $\Diamond A \stackrel{def}{=} \neg \Box \neg A$, come from modal logic and state that A holds in all reachable states and in some reachable state respectively. Finally $e \hookrightarrow e'$ states that the reference denoted by e' can be reached by the datum e, with the dual $e' \# e \stackrel{def}{=} \neg e \hookrightarrow e'$ meaning the reference e' cannot be reached by (or is fresh from) the datum e. Substitution of expressions e for the variable x in the formula A, written $A[e/x]$, is defined similarly to the λ-logic.

The model builds on the type context for variables and hidden references hence the model $\mathcal{M} \stackrel{def}{=} (\nu \tilde{l})\xi$ hides the names in $\tilde{l}$ and maps variables to values in ξ. This is a

simplification of the model in [72].

The free locations of a term M are defined as expected and written $\mathsf{fl}(M)$. The *free plain names* of an expression e is the set of references in e which do not occur dereferenced, which in this limited case is all references. A model $\mathcal{M}'$ is an extension of a model $\mathcal{M}$ if the extension contains all mappings of the extended model (i.e. model equivalence, written $\mathcal{M} \approx \mathcal{M}'$) and potentially some new mappings, this is written as $\mathcal{M} \rightsquigarrow \mathcal{M}'$. These are defined as expected. The semantics for formulae is given using these definitions of the models, with the new logical constructor semantics defined as follows (adapted from [72] for this simpler language).

$$\begin{array}{lcl} \mathcal{M} \models \Box A & \stackrel{def}{=} & \forall \mathcal{M}'. \mathcal{M} \rightsquigarrow \mathcal{M}' \rightarrow \mathcal{M}' \models A \\ \mathcal{M} \models \nu x.A & \stackrel{def}{=} & \exists \mathcal{M}'. (\nu l)\mathcal{M}' \approx \mathcal{M} \wedge \mathcal{M}'[x : l] \models A \\ \mathcal{M} \models e_1 \hookrightarrow e_2 & \stackrel{def}{=} & [\![e_2]\!]_{\xi,\sigma} \in \mathsf{fl}([\![e_1]\!]_{\xi,\sigma}) \text{ for each } \mathcal{M} \approx (\nu \tilde{l})(\xi, \sigma) \end{array}$$

Properties of formulae are introduced to restrict certain axioms and rules to ensure soundness (and completeness) proofs hold.

Definition 18 (Properties of formulae in the Local-logic)**.** Thinness *is defined for a formula if a particular variable x, can be removed from the model $\mathcal{M}$ alongside its mapping, written $\mathcal{M}/x$, and still satisfy the formula.* Monotonicity *of a formula is defined if references can be hidden in the model and still satisfy the formulae. The formula A is* antimonotone *if $\neg A$ is monotone. A formula A is* stateless *if it holds in all future states.*

$$\begin{array}{rcl} A\text{-}thin\ w.r.t\ x & \stackrel{\text{def}}{=} & \forall\, \mathcal{M}.\ \mathcal{M} \models A \rightarrow \mathcal{M}/x \models A \\ A\text{-}monotone & \stackrel{\text{def}}{=} & \forall\, \mathcal{M}, l.\ \mathcal{M} \models A \wedge l \notin \mathsf{fl}(A) \rightarrow (\nu l)\mathcal{M} \models A. \\ A\text{-}stateless & \stackrel{\text{def}}{=} & A \rightarrow \Box A \end{array}$$

The paper proposes that monotonicity and antimonotonicity are not required but include them for the soundness proofs. Thinness is a development of A^{-x}, meaning x is not in the free variables of A but covers this more complex model which cannot remove variables in a formula even when not used in the formula. For example the model $\mathcal{M} \equiv x : l, y : \lambda z.\mathsf{if}\ z = l\ \mathsf{then}\ \mathsf{true}\ \mathsf{else}\ \mathsf{false}$ and the formula $\exists x'.y \bullet x' = \mathsf{true}\{\}$ is not satisfied by $\mathcal{M}/x$ as l is only derivable from x and is the only input to y that returns true.

Syntactic definitions of each of these properties are defined, allowing for syntactic checks on whether these properties hold for certain formulae. These syntactic definitions cover all formulae that are required to obtain certain properties in the reasoning, however may not cover all formulae that obey the property.

The axioms from [72] are extensive, however the related axioms to the logic introduced here are introduced below. Many of these axioms will be replicated in essence in the ν-logic

in Sec. 4.5, hence their introduction here. Axioms of note are $(v5)$ which states how the ν-quantifier interacts with the $\forall$-quantifier, $(r4)$ states how freshness proves inequality of references, the axioms for modality are standard, apart from the 2 introduced below, which define how modality interact with the ν-quantifier(s).

$(u1)\quad A \rightarrow \forall x.A \qquad x \notin \mathsf{fv}(A) \wedge A\text{-thin w.r.t } x$

$(v1)\quad A \rightarrow \nu x.A \qquad x \notin \mathsf{fv}(A)$

$(v2)\quad \nu x.A \leftrightarrow A \qquad x \notin \mathsf{fv}(A) \wedge A - \mathsf{monotone}$

$(v3)\quad \nu x.(A \wedge A') \rightarrow \nu x.A \wedge \nu x.A'$

$(v4)\quad \nu x.(A \vee A') \leftrightarrow \nu x.A \vee \nu x.A'$

$(v5)\quad \nu x.\forall y.A \rightarrow \forall y.\nu x.A$

$(v6)\quad \exists y.\nu x.A \rightarrow \nu x.\exists y.A$

$(v7)\quad \nu x.\bar{\nu} y.A \rightarrow \bar{\nu} y.\nu x.A$

$(v8)\quad \nu x.\nu y.A \leftrightarrow \nu y.\nu x.A$

$(r1)\quad x \hookrightarrow x$

$(r2)\quad x \hookrightarrow y \wedge y \hookrightarrow z \rightarrow x \hookrightarrow z$

$(r3)\quad x\#y^{\alpha} \qquad \alpha \in \{\mathsf{Unit}, \mathsf{Bool}\}$

$(r4)\quad x\#y \rightarrow x \neq y$

$(r5)\quad x\#y \wedge y \hookrightarrow z \rightarrow x\#z$

$(n1)\quad \Box\bar{\nu} x.A \leftrightarrow \bar{\nu} x.\Box A$

$(n2)\quad \nu x.\Box A \rightarrow \Box\nu x.A$

Figure 2.26: Axioms for the new logical constructors in the Local-logic.

The rules of inference for triples build off the λ-logic and some key extensions are seen in Fig. 2.27. The rules differ from the λ-logic in various ways, primarily the inclusion of the auxiliary variable i. This variable i is used to reference any datum that already exists in the state. The $[\textsc{Lam}]_{Local}$ rule includes a quantification over $\forall i$ which is a quantification over arbitrary variables (of arbitrary type). The i is used in $[\textsc{Ref}]_{Local}$ to represent this freshness alongside the ν-binder to ensure x is hidden. The i can be quantified over in: $[\textsc{Aux}_{\forall} V]_{Local}$ if the program is a value; $[\textsc{Aux}_{\forall}]_{Local}$ if the type of i is a base type; or $[\textsc{Lam}]_{Local}$ if applicable. The i can also be replaced by an expression using $[\textsc{Subs}]_{Local}$ which allows this variable to be seen as a quantification over all possible locations (or expressions) in the initial state and hence it should not be possible to obtain the contradiction $u\#u$, however it is not entirely obvious or easy to use this variable i.

The $[\textsc{Ref}()]_{Local}$ rule is the only rule that introduces the ν-binder to hide the name that the reference represents within the logic. This rule can be used to reason about $\nu n.n$ by letting $\mathsf{ref}(\ ()\ ) \stackrel{def}{=} \nu n.n$, then the resulting triple would be $\{\mathsf{T}\}\ \mathsf{ref}(\ ()\ )\ :_u\ \{\nu x.(u = x \wedge u\#i)\}$ meaning the reference is hidden but in this case can be accessed by u, and is fresh from all other references which can be accessed in the initial state. The definition of the

initial state comes from the $[\textsc{Subs}]_{Local}$ rule instantiation forbidding $[u/i]$ as $u \notin \mathsf{fpn}(e)$.

For the example $\lambda().\mathsf{ref}()$ (which will be thought of as a translation of gensym in the ν_{GS}-calculus), the triple $\{\mathsf{T}\}\ \lambda().\mathsf{ref}() :_m \{\forall i.m\bullet() = u\{\nu x.(u = x \wedge u\#i)\}\}$ can be derived. This means i can be instantiated as any reference except that of x and u and hence this means x (and u) are fresh from any previously generated reference.

Finally the $[\textsc{Cons-Eval}]_{Local}$ rule is included here in this simplified version of the logic. The rule is a strengthened version of the standard consequence rule which represents the compositionality of the language alongside the chosen method of expressing freshness of references.

$$\frac{\{C \wedge A^{-x\tilde{i}}\}\ M :_m \{C'\}}{\{A\}\ \lambda x.M :_u \{\forall x\tilde{i}.C \to u \bullet x = m\{C'\}\}}\ [\textsc{Lam}]_{Local}$$

$$\frac{i - \text{auxiliary}}{\{C\}\ \mathsf{ref}() :_u \{\nu x.(C \wedge u\#i^X \wedge u = x\}}\ [\textsc{Ref}()]_{Local}$$

$$\frac{\{C^{-i}\}\ V :_u \{C'\}}{\{C\}\ V :_u \{\forall i.C'\}}\ [\textsc{Aux}_\forall V]_{Local} \qquad \frac{\{C^{-i}\}\ M :_u \{C'\} \quad \alpha \text{ - base type}}{\{C\}\ M :_u \{\forall i^\alpha.C'\}}\ [\textsc{Aux}_\forall]_{Local}$$

$$\frac{\{A\}\ M :_u \{B\} \quad u \notin \mathsf{fpn}(e)}{\{A[e/i]\}\ M :_u \{B[e/i]\}}\ [\textsc{Subs}]_{Local}$$

$$\frac{\begin{array}{l}\{C_0\}\ M :_m \{C_0'\} \qquad x \text{ fresh; } \tilde{i} \text{ auxiliary} \\ \Box\forall\tilde{i}.\{C_0\}x \bullet () = m\{C_0'\} \to \Box\forall\tilde{i}.\{C\}x \bullet () = m\{C'\}\end{array}}{\{C\}\ M :_m \{C'\}}\ [\textsc{Cons-Eval}]_{Local}$$

Figure 2.27: Key new inference rules of the Local-logic.

The $[\textsc{LetOpen}]_{Local}$ rule below does not occur directly in [72] but as a rule for an extension of the logic which contains a similar rule. This rule allows for the temporary revealing of a reference within a $\mathsf{let}\ x = M\ \mathsf{in}\ N$ constructor. This rule does not seem to exist in this exact form in the paper, but is expected to be sound.

$$\frac{\{A\}\ M :_x \{\nu\tilde{y}.C\} \quad \{C\}\ N :_u \{B\} \quad B\text{-thin w.r.t } x}{\{A\}\ \mathsf{let}\ x = M\ \mathsf{in}\ N :_u \{\nu\tilde{y}.B\}}\ [\textsc{LetOpen}]_{Local}$$

The Local-logic is proven sound and three completeness results are proven in [4] however the $[\textsc{LetOpen}]_{Local}$ is not included in this proof.

Compared to the original program logic in [72], the Local-logic above ignores all the elements of state that are not related to the naming of fresh locations. The original logic is complex, as it builds on the program logic for state with aliasing [5], referred to here as the Alias-logic. The Alias-logic is for a language which uses locations directly, and does not include the fresh generation of locations, $\mathsf{ref}(M)$. In both logics with memory, a new logical substitution is introduced to deal with aliasing in memory. A new logical constructor to ensure (in)dependence of a particular location in the state is also required.

The quantifiers $\nu x.A$ or $\bar{\nu} x.A$ are not required in the Alias-logic, as these are used purely for the fresh reference generation. Similarly, the modalities $\Box A$ and $\diamondsuit A$ are also not required when simply discussing non-fresh references as all references exist at initiation, and hence reasoning about future references (or states) is not required. Both logics are proven sound, alongside different notions of completeness [4].

The extra complications of the logic, which are originally introduced due to the values stored in the state, encouraged the search for a simpler logic specifically for the ν-calculus.

2.2.4 Other Logics of Interest

This thesis builds upon the Hoare logic introduced in Sec. 2.2.2 for the STLC [28]. Alternative approaches to Hoare logic and their relation to names (or their applications) are briefly introduced here.

Separation Logic

Reynolds' separation logic [59] is an extension of Hoare logic which allows for reasoning about states consisting of the store and heap. The primary concept is the separation of the heap memory into disjoint heaps and reasoning about these disjointly. The frame rule introduced enables local reasoning about component programs and their effect on a portion of memory whilst not affecting another disjoint part of the heap.

Extensions to separation logic in [7] do allow for reasoning about fresh name generation via the standard use of $\mathsf{Ref}()$ as the name generator, alongside equality of references. Further work is required to extract the essence of the logic required for reasoning purely about fresh names and identify any issues which may arise. It is not clear whether key examples such as the "hard" example introduced later in Ex. 19 can be reasoned about using this logic.

A Hoare Logic For CBV Functional Languages

Reǵis-Gianas and Pottier introduced a higher-order typed Hoare logic for CBV functional languages in [57] similar to [28] with some technical differences. Crucially, equality is only considered for values and hence function applications (and other non-value terms) are not considered in the formulae. This simplification of the logic does have the added benefit that verification conditions can be proven interactively in Coq or with an automated theorem prover. Names or local state are not explicitly considered in this paper.

Hoare Type Theory

Hoare Type Theory combines Hoare triples with dependent types, formalizing the generating process of verification conditions for effectful computations using monadic judgments [44, 43]. This combines standard static checking techniques with logical verifications. The introduction of local state to the former work can be seen in [42].

2.3 Summary

The underlying principles have been introduced in this chapter that will form the basis of the rest of the thesis. The untyped λ-calculus and the STLC introduced with the common syntax that will be used throughout the rest of the thesis alongside important ideas such as types and equivalence. The basics of program logics are covered for an imperative language and a functional language. The extension to the STLC which includes local state, is introduced with the program logic for the case where the state only stores (), which draws parallels to the study of names.

In the next chapter the STLC will be built upon via the introduction of names in programming languages referred too as the ν_{GS}-calculus and $\lambda\nu$-calculus, alongside an analysis of relations between these languages and methods to prove equivalence, all from the literature. The program logic for the ν_{GS}-calculus extends that of the λ-logic introduced in this chapter. The standard technique of proving soundness in this chapter is then extended to prove soundness of the ν-logic.

Chapter 3

The ν-Calculus

The π-calculus introduced the concept of scope of names with fresh (channel) name generation which was one of its fundamental developments over previous attempts at developing process calculi [40]. In 1993 Andrew Pitts and Ian Stark developed the idea of scope and name generation from the π-calculus to the STLC [52]. The addition of names to a simple programming language such as the STLC comes in the form of the ν-calculus, referred to here as the ν_{PS}-calculus. The simple method of generating names (constructor) and equating names (destructor) are introduced which captures the essence of names without any complications of what the names represent.

In the following two sections, two versions of the ν_{PS}-calculus are introduced. In Sec. 3.1 the first version uses the fresh name constructor gensym in the programming language which shall be referred to as ν_{GS}-calculus. The following chapters will develop a program logic for ν_{GS}-calculus hence its introduction here. The original ν_{PS}-calculus, which combines the creation of the fresh name with a scope of the name, is introduced in Sec. 3.2. A translation between the ν_{GS}-calculus and ν_{PS}-calculus is provided in Sec. 3.3, showing the two languages are essentially equivalent.

Previous work on the ν_{PS}-calculus focuses primarily on the method of proving equality of programs in the ν_{PS}-calculus and the various methods are summarised in Sec. 3.4. These methods consist of: an equational logic in Sec. 3.4.1, logical relations (and predicated logical relations) in Sec. 3.4.2, Kripke logical relations in Sec. 3.4.3, environmental bisimulations in Sec. 3.4.4, nominal games in Sec. 3.4.5 and probabilistic programming in Sec. 3.4.6.

In 1993 Martin Odersky introduced a CBN version of the STLC with names and equality known as the $\lambda\nu$-calculus, which is introduced in Sec. 3.5 [46]. The relation between the ν_{PS}-calculus and the $\lambda\nu$-calculus were discussed and formalised in [36] which is summarised in Sec. 3.6.

3.1 The ν_{GS}-Calculus

The gensym function (short for **gen**erate **sym**bol) is used in the LISP programming language and is defined in LISP 1.5 Programmer's Manual [37] as follows:

> "The function gensym has no arguments. Its value is a new, distinct, and freshly created atomic symbol with a print name of the form G00001, G00002, . . . , G99999.
>
> This function is useful for creating atomic symbols when one is needed; each one is guaranteed unique. gensym names are not permanent and will not be recognized if read back in. " [37]

In essence, gensym produces fresh names (or atoms) which have useful applications within the language (LISP). In LISP, gensym can be used to create hygienic macros so that variables in the macro are guaranteed to be distinct from any other variable outside of the macro.

The ν_{GS}-calculus introduces gensym to the STLC as a method of generating fresh names. In this case, names do not attach any additional meaning such as the store or variable names. The only operation on names is that of equating them. This simple addition of names captures the essence of names as described in the introduction (Sec. 1.1). The ν_{GS}-calculus is introduced formally in the Sec. 3.1.1, and some key examples of programs in the ν_{GS}-calculus introduced in Sec. 3.1.2.

3.1.1 The ν_{GS}-Calculus Programming Language

Syntax

The types of ν_{GS}-calculus extend the STLC types in Sec. 2.1.2 with the single new type for names, Nm.

The terms are extended from the STLC in Sec. 2.1.2 with three constructs: gensym to produce fresh names, $=$ to equate names (similar to the equality of Booleans) and names themselves ranging over n, n', n_1, ... A possibly empty set of names is written G.

The equality operator could be split into an equality on names and an equality on Booleans, however for simplicity they are combined into one operator.

$$
\begin{array}{lcl}
\alpha & ::= & \mathsf{Unit} \mid \mathsf{Bool} \mid \mathsf{Nm} \mid \alpha \to \alpha \mid \alpha \times \alpha \\
\Gamma & ::= & \emptyset \mid \Gamma, x : \alpha \\
V & ::= & () \mid \mathsf{true} \mid \mathsf{false} \mid \lambda x^{\alpha}.M \mid \mathsf{n} \mid \mathsf{gensym} \mid \langle V, V \rangle \\
M & ::= & V \mid x \mid \neg M \mid MM \mid M = M \mid \mathsf{if}\ M\ \mathsf{then}\ M\ \mathsf{else}\ M \\
 & & \mid \langle M, M \rangle \mid \pi_1(M) \mid \pi_2(M) \mid \mathsf{let}\ x = M\ \mathsf{in}\ M \\
G & ::= & \text{Set of names}
\end{array}
$$

Figure 3.1: Syntax of the ν_{GS}-calculus.

Typing rules

A type check in this format does not require a secondary check as this will be done separately, hence the typing judgment is as in the STLC, i.e. $\Gamma \vdash M : \alpha$

Definition 19 (Typing rules for the ν_{GS}-calculus)**.** *The typing rules for the ν_{GS}-calculus are those of Fig. 3.2 and only require that names are indeed names.*

$$\frac{-}{\Gamma \vdash () : \mathsf{Unit}} \qquad \frac{b \in \{\mathsf{true}, \mathsf{false}\}}{\Gamma \vdash b : \mathsf{Bool}} \qquad \frac{\mathsf{n} - \text{name}}{\Gamma \vdash \mathsf{n} : \mathsf{Nm}} \qquad \frac{-}{\Gamma \vdash \mathsf{gensym} : \mathsf{Unit} \to \mathsf{Nm}}$$

$$\frac{\Gamma, x : \alpha \vdash M : \alpha'}{\Gamma \vdash \lambda x^{\alpha}.M : \alpha \to \alpha'} \qquad \frac{\Gamma \vdash M : \alpha \to \alpha' \quad \Gamma \vdash N : \alpha}{\Gamma \vdash MN : \alpha'} \qquad \frac{\Gamma \vdash M : \mathsf{Bool}}{\Gamma \vdash \neg M : \mathsf{Bool}}$$

$$\frac{\Gamma \vdash M : \alpha_x \quad \Gamma \vdash N : \alpha_x \quad \alpha_x \in \{\mathsf{Bool}, \mathsf{Nm}\}}{\Gamma \vdash M = N : \mathsf{Bool}} \qquad \frac{\Gamma \vdash M : \mathsf{Bool} \quad \Gamma \vdash M_1 : \alpha \quad \Gamma \vdash M_2 : \alpha}{\Gamma \vdash \mathsf{if}\ M\ \mathsf{then}\ M_1\ \mathsf{else}\ M_2 : \alpha}$$

$$\frac{\Gamma \vdash M : \alpha \quad \Gamma \vdash N : \alpha'}{\Gamma \vdash \langle M, N \rangle : \alpha \times \alpha'} \qquad \frac{\Gamma \vdash M : \alpha_1 \times \alpha_2 \quad i = 1, 2}{\Gamma \vdash \pi_i(M) : \alpha_i} \qquad \frac{\Gamma \vdash M : \alpha \quad \Gamma, x : \alpha \vdash N : \alpha'}{\Gamma \vdash \mathsf{let}\ x = M\ \mathsf{in}\ N : \alpha'}$$

Figure 3.2: Typing rules of the ν_{GS}-calculus.

Unlike LISP, the type of gensym is $\mathsf{Unit} \to \mathsf{Nm}$, this is because the STLC does not execute commands but reduces them, hence this clarifies that the function is called when it is applied to (). The typing rules ensure that gensym will always be applied to a () and hence these two forms of gensym are equivalent.

Operational Semantics

There are no binders for names and thus the idea of bound or free names of a term becomes simply all names in a term. A set of names is defined as a *nameset*.

Definition 20 (All names in terms)**.** **A**ll names *of a term M written $\mathring{a}(M)$ are defined as follows.*

$$
\begin{array}{rcl@{\qquad}rcl}
\mathring{a}(()) & \stackrel{\text{def}}{=} & \emptyset & \mathring{a}(\lambda x^{\alpha}.M) & \stackrel{\text{def}}{=} & \mathring{a}(M) \\
\mathring{a}(\mathsf{true}) & \stackrel{\text{def}}{=} & \emptyset & \mathring{a}(M = N) & \stackrel{\text{def}}{=} & \mathring{a}(M) \cup \mathring{a}(N) \\
\mathring{a}(\mathsf{false}) & \stackrel{\text{def}}{=} & \emptyset & \mathring{a}(\neg M) & \stackrel{\text{def}}{=} & \mathring{a}(M) \\
\mathring{a}(\mathsf{n}) & \stackrel{\text{def}}{=} & \{\mathsf{n}\} & \mathring{a}(\mathsf{if}\ M\ \mathsf{then}\ M_1\ \mathsf{else}\ M_2) & \stackrel{\text{def}}{=} & \mathring{a}(M) \cup \mathring{a}(M_1) \cup \mathring{a}(M_2) \\
\mathring{a}(\mathsf{gensym}) & \stackrel{\text{def}}{=} & \emptyset & \mathring{a}(\langle M, N\rangle) & \stackrel{\text{def}}{=} & \mathring{a}(M) \cup \mathring{a}(N) \\
\mathring{a}(x) & \stackrel{\text{def}}{=} & \emptyset & \mathring{a}(\pi_i(M)) & \stackrel{\text{def}}{=} & \mathring{a}(M) \\
 & & & \mathring{a}(\mathsf{let}\ x = M\ \mathsf{in}\ N) & \stackrel{\text{def}}{=} & \mathring{a}(M) \cup \mathring{a}(N)
\end{array}
$$

The abbreviation $\mathring{a}(M, N)$ is used to mean $\mathring{a}(M) \cup \mathring{a}(N)$.

Definition 21 (Configurations)**.** *A nameset G (stands for **G**enerated) and a term M are combined into a* configuration *of the form $(G,\ M)$.*

A configuration is valid *if the nameset contains all the names in the term*

$$(G,\ M) \text{ is valid } \stackrel{\text{def}}{=} \mathring{a}(M) \subseteq G$$

All configurations will be considered valid from here on.

Definition 22 (Reduction relation)**.** *The reduction relation for the ν_{GS}-calculus, $\rightarrow$ relates two configurations $(G,\ M)$ and $(G',\ M')$, as $(G,\ M) \rightarrow (G',\ M')$. The reflexive transitive closure of $\rightarrow$ is written $\rightarrow^*$. If $(G,\ M) \rightarrow^* (G',\ V)$ for some value V, then this is written $(G,\ M) \Downarrow (G',\ V)$.*

Definition 23 (ν_{GS}-calculus CBV operational semantics (evaluation contexts))**.** *The CBV operational semantics of the STLC in Def. 8 are extended to the ν_{GS}-calculus as follows.*

The evaluation contexts for the ν_{GS}-calculus are those of Fig. 3.3 which only differ from those of Def. 8 through the introduction of $=$ for names.

$$
\begin{array}{rcl}
\mathcal{E}[\cdot] & ::= & [\cdot] \mid \mathcal{E}[\cdot]M \mid V\mathcal{E}[\cdot] \mid \neg\mathcal{E}[\cdot] \mid \mathcal{E}[\cdot] = M \mid V = \mathcal{E}[\cdot] \\
 & & \mid \mathsf{if}\ \mathcal{E}[\cdot]\ \mathsf{then}\ M\ \mathsf{else}\ M' \mid \mathsf{if}\ V\ \mathsf{then}\ \mathcal{E}[\cdot]\ \mathsf{else}\ M' \mid \mathsf{if}\ V\ \mathsf{then}\ W\ \mathsf{else}\ \mathcal{E}[\cdot] \\
 & & \mid \langle \mathcal{E}[\cdot], M\rangle \mid \langle V, \mathcal{E}[\cdot]\rangle \mid \pi_i(\mathcal{E}[\cdot]) \mid \mathsf{let}\ x = \mathcal{E}[\cdot]\ \mathsf{in}\ M
\end{array}
$$

Figure 3.3: Evaluation contexts of the ν_{GS}-calculus.

The reduction rules of the ν_{GS}-calculus are introduced in Fig. 3.4. The application of gensym *to* () *is the only manner of creating a new name which must be added to the original nameset. All other rules are standard, except the final rule which allows for the addition or removal of unused names to the nameset in the configuration.*

$$
\begin{array}{rcll}
(G,\ V) & \to & (G,\ V) & \\
(G,\ (\lambda x^{\alpha}.M)V) & \to & (G,\ M[V/x]) & \\
(G,\ \mathsf{gensym}()) & \to & (G \cup \{\mathsf{n}\},\ \mathsf{n}) & \mathsf{n} \notin G \\
(G,\ v = v) & \to & (G,\ \mathsf{true}) & v \in \{\mathsf{true}, \mathsf{false}\} \cup G \\
(G,\ v = v') & \to & (G,\ \mathsf{false}) & v \neq v',\ v, v' \in \{\mathsf{true}, \mathsf{false}\} \cup G \\
(G,\ \neg\mathsf{true}) & \to & (G,\ \mathsf{false}) & \\
(G,\ \neg\mathsf{false}) & \to & (G,\ \mathsf{true}) & \\
(G,\ \mathsf{if\ true\ then}\ V\ \mathsf{else}\ V') & \to & (G,\ V) & \\
(G,\ \mathsf{if\ false\ then}\ V\ \mathsf{else}\ V') & \to & (G,\ V') & \\
(G,\ \pi_i(\langle V_1, V_2\rangle)) & \to & (G,\ V_i) & \\
(G,\ \mathsf{let}\ x = V\ \mathsf{in}\ M) & \to & (G,\ M[V/x]) & \\
(G,\ M) \to (G',\ M') & \to & (G,\ \mathcal{E}[M]) \to (G',\ \mathcal{E}[M']) & \\
(G,\ M) \to (G',\ M') & \leftrightarrow & (G \cup G_0,\ M) \to (G' \cup G_0,\ M') & G' \cap G_0 = \emptyset
\end{array}
$$

Figure 3.4: Reduction rules of the ν_{GS}-calculus.

Definition 24 (Static syntax)**.** *A term is classified as* static syntax *if it contains no names i.e.* $\mathfrak{d}(M) = \emptyset$.

Static syntax is required in the initial state of the reduction relation, hence $(\emptyset,\ M)$ *is the initial configuration if* $\mathfrak{d}(M) = \emptyset$.

Static syntax guarantees that any name produced is freshly produced by an application of gensym. The program logic for the ν_{GS}-calculus introduced in Chapt. 4, only allows reasoning about static syntax programs.

Some properties of the ν_{GS}-calculus are introduced here. The function $(a\ b) \cdot X$ is the swapping function from nominal logic [50], which swaps the names a and b in X uniformly, where in this case X is a configuration.

Definition 25 (Properties of reduction relation)**.** *These are all proven in [68, 62, 22] and are assumed as definitions here.*

Weakening *If* $(G,\ M) \to^* (G',\ M')$ *then* $\forall\, G''.\ G' \cap G'' = \emptyset \ \to\ (G \cup G'',\ M) \to^* (G' \cup G'',\ M')$

Equivariance *If* $(G,\ M) \to^* (G',\ M')$ *then* $\forall\, a, b.\ (a\ b) \cdot (G,\ M) \to^* (a\ b) \cdot (G',\ M')$

Nominal Determinancy *If* $(G,\ M) \to^* (G',\ M')$ *and* $(G,\ M) \to^* (G'',\ M'')$
then $\exists\, a_1, b_1,\ ..., a_n, b_n \notin G.\ (G',\ M')$ *is identical to* $(a_1\ b_1) \cdot ... \cdot (a_n\ b_n) \cdot (G'',\ M'')$.
This means $(G',\ M')$ *and* $(G'',\ M'')$ *are identical up to permutation of fresh names.*

Strong Normalisation *Every well-typed valid configuration terminates.*

Contextual Congruence

Contextual equivalence in the ν_{GS}-calculus is defined with similar intentions to the STLC in Def. 34 now with the new syntax, but first some standard definitions are re-introduced.

Definition 26 (Single holed contexts)**.** *Single-holed contexts are formally defined in Fig. 3.5, with the hole being filled with a term of type* α_0 *in the context* $\mathsf{C}[\cdot]_{\alpha_0}$.

$$
\begin{aligned}
\mathsf{C}[\cdot]_{\alpha_0} \quad ::= \quad & [\cdot]_{\alpha_0} \mid \lambda x^{\alpha}.\mathsf{C}[\cdot]_{\alpha_0} \mid \mathsf{C}[\cdot]_{\alpha_0} M \mid M\mathsf{C}[\cdot]_{\alpha_0} \mid \neg \mathsf{C}[\cdot]_{\alpha_0} \\
& \mid \mathsf{if}\ \mathsf{C}[\cdot]_{\alpha_0}\ \mathsf{then}\ M\ \mathsf{else}\ N \mid \mathsf{if}\ M\ \mathsf{then}\ \mathsf{C}[\cdot]_{\alpha_0}\ \mathsf{else}\ N \mid \mathsf{if}\ M\ \mathsf{then}\ N\ \mathsf{else}\ \mathsf{C}[\cdot]_{\alpha_0} \\
& \mid \langle \mathsf{C}[\cdot]_{\alpha_0}, M \rangle \mid \langle M, \mathsf{C}[\cdot]_{\alpha_0} \rangle \mid \pi_i(\mathsf{C}[\cdot]_{\alpha_0}) \\
& \mid \mathsf{C}[\cdot]_{\alpha_0} = M \mid M = \mathsf{C}[\cdot]_{\alpha_0} \mid \mathsf{let}\ x = \mathsf{C}[\cdot]_{\alpha_0}\ \mathsf{in}\ N \mid \mathsf{let}\ x = M\ \mathsf{in}\ \mathsf{C}[\cdot]_{\alpha_0}
\end{aligned}
$$

Figure 3.5: Single holed contexts of the ν_{GS}-calculus.

Definition 27 (All names in a single-holed context)**.** *The function* $\mathfrak{a}(\cdot)$ *is extended to single-holed context from Def. 20 by including the definition* $\mathfrak{a}([\cdot]) = \emptyset$.

Definition 28 (Typing single-holed contexts)**.** *Contexts are typed using the rules in Fig. 3.2 extended with the obvious rule for the typed hole as follows.*

$$\frac{-}{\Gamma \vdash [\cdot]_{\alpha} : \alpha}$$

Definition 29 (Contextual equivalence (ν_{GS}-calculus))**.** Contextual equivalence *is defined for two closed terms* M *and* N *such that* $\emptyset \vdash M : \alpha$ *and* $\emptyset \vdash N : \alpha$ *with* $\mathfrak{a}(M) \cup \mathfrak{a}(N) \subseteq G$, *written* $M \cong_{\alpha}^{G} N$. *Contextual equivalence holds if all contexts with names in* G *and holes*

filled with the two terms evaluate to the same Boolean constant as follows.

$$M \cong_\alpha^G N \quad \stackrel{\text{def}}{=} \quad \forall\, \mathsf{C}[\cdot]_\alpha.\ \ \emptyset \vdash \mathsf{C}[\cdot]_\alpha : \mathsf{Bool} \ \wedge\ \mathring{a}(\mathsf{C}[\cdot]) \subseteq G$$
$$\rightarrow \left(\begin{array}{c} (\exists\, G'.(G,\ \mathsf{C}[M]) \Downarrow (G',\ \mathsf{true}) \\ \leftrightarrow \\ (\exists\, G''.(G,\ \mathsf{C}[N]) \Downarrow (G'',\ \mathsf{true}) \end{array} \right)$$

Contextual congruence compares two closed terms with the same nameset. Clearly for any equivalent programs $M \cong_\alpha^G N$ then $(G,\ M) \Downarrow (G',\ W)$ and $(G,\ N) \Downarrow (G'',\ V)$ with $G' \cap G'' = G$ does **not** imply $W \cong_\alpha^{G' \cup G''} W$ as the simple counter example of $\mathsf{gensym}() \cong_\alpha^\emptyset \mathsf{gensym}()$ fails this property.

To show how the namesets G' and G'' may differ, consider the example $M \equiv \mathsf{gensym}()$ and $N \equiv \pi_1(\langle \mathsf{gensym}(), \mathsf{gensym}() \rangle)$. Although the result of the outputted Boolean constant true is not affected, the nameset G' would contain one less name than the nameset G'' due to the production of two fresh names by N.

The following lemma states that unused Let assignments (including those pointing to $\mathsf{gensym}()$) can be removed if unused.

Lemma 30 (Let removal)**.**

$$\forall\, G, M, x, N.\ (x \notin \mathsf{fv}(M) \ \wedge\ \mathring{a}(N, M) \subseteq G) \ \rightarrow\ (G,\ M) \cong_\alpha^G (G,\ \mathsf{let}\ x = N\ \mathsf{in}\ M)$$

Proof. Clearly holds given x does not occur free in M meaning any term N cannot affect M. For case where N is $\mathsf{gensym}()$ see [52, Corollary 6]. □

The following lemma shows names can be regenerated assuming $(V(\tilde{x}))[\tilde{\mathsf{n}}/\tilde{x}]$ is identical to $V(\tilde{\mathsf{n}})$ and $\mathring{a}(V(x)) \cap \{\tilde{\mathsf{n}}\} = \emptyset$.

Lemma 31 (Name regeneration)**.**

$$\forall\, G, M, \tilde{x}, \tilde{\mathsf{n}}, V(\tilde{x}), V(\tilde{\mathsf{n}}).\ (V(\tilde{x}))[\tilde{\mathsf{n}}/\tilde{x}] \textit{ is syntactically identical to } V(\tilde{\mathsf{n}})$$
$$\rightarrow$$
$$(G,\ M) \Downarrow (G \cup \{\tilde{\mathsf{n}}\},\ V(\tilde{\mathsf{n}})) \ \rightarrow\ M \cong_\alpha^G \mathsf{let}\ \tilde{x} = \mathsf{gensym}()\ \mathsf{in}\ V(x)$$

Proof. See [52, Corollary 6]. □

The following lemma show that namesets can be added or removed if the names are unused in both terms in a contextual congruence.

Lemma 32 (Adding/removing excess names maintains contextual congruence)**.**

$$\forall\, G, G'.\ \mathring{a}(M, N) \cap \mathring{a}(G') = \emptyset \ \rightarrow\ (M \cong_\alpha^G N \leftrightarrow M \cong_\alpha^{G \cup G'} N)$$

Proof. Clearly holds by Def. 29 and the final rule of Def. 23 which allows for unused names to be added and removed freely from namesets in configuration reductions. □

It is unknown whether contextual equivalence is decidable above first-order [62].

3.1.2 Programs in the ν_{GS}-Calculus

This section gives simple examples of ν_{GS}-calculus programs which will later be reasoned about using the program logic in Chapt. 7. Many of these examples are not original and appear in the literature [52, 62, 2, 68].

Before these examples are introduced, the idea of *reachable* and *unreachable* (hidden) names are introduced. Names can occur in a term but get trapped inside with no escape, meaning there is no method to use the term that allows for the name as output, i.e. hidden. This typically occurs when names are bound outside a λ-binder but are used inside the λ-binder under a destructor (in this case equality). For example the name n is reachable from the terms n, $\langle \mathsf{n}, \mathsf{n}' \rangle$ and $\lambda y.\mathsf{n}$ but is unreachable from n', $\lambda x.\pi_1(\langle \mathsf{n}', \mathsf{n} \rangle)$ and $\lambda x.(x = \mathsf{n})$. This concept extends to a set of terms $\tilde{M}$, such that the term N is reachable by $\tilde{M}$ if there exists some M_0 such that $\mathring{\mathsf{a}}(M_0) = \emptyset$ and $(\mathring{\mathsf{a}}(\tilde{M}),\ M_0\tilde{M}) \Downarrow (G',\ N)$, and is defined as unreachable otherwise.

The following abbreviation is used when numerous fresh names are generated using $\mathsf{let}\ x = \mathsf{gensym}()\ \mathsf{in}\ \ldots$. Sometimes the set of variables $x_1, x_2, \ldots, x_k$ is abbreviated further to $\tilde{x}$.

$$\mathsf{let}\ x_1,\ \ldots, x_k = \mathsf{gensym}()\ \mathsf{in}\ M \stackrel{def}{=} \begin{array}{l} \mathsf{let}\ x_1 = \mathsf{gensym}()\ \mathsf{in} \\ \ldots \\ \mathsf{let}\ x_k = \mathsf{gensym}\ \mathsf{in}\ M \end{array}$$

Example 4 (STLC programs)**.** All STLC programs evaluate identically in the ν_{GS}-calculus in any nameset configuration as follows.

$$M \Downarrow_\lambda V \quad \rightarrow \quad (\emptyset,\ M) \Downarrow (\emptyset,\ V)$$

Example 5 $((\emptyset,\ \mathsf{gensym}()))$**.** The core example of generating a fresh name clearly holds directly from the reduction rules in Fig. 3.4.

$$(\emptyset,\ \mathsf{gensym}()) \Downarrow (\{\mathsf{n}\},\ \mathsf{n})$$

Example 6 $((\emptyset,\ (\lambda x^{\mathsf{Nm}}.x)(\mathsf{gensym}())))$**.** The identity function applied to a fresh name

produces the same fresh name as follows.

$$\frac{\frac{}{1 \quad (\emptyset,\ \mathsf{gensym}()) \Downarrow (\{\mathsf{n}\},\ \mathsf{n})}}{2 \quad (\{\mathsf{n}\},\ (\lambda x^{\mathsf{Nm}}.x)\mathsf{n}) \Downarrow (\{\mathsf{n}\},\ \mathsf{n})}}{3 \quad (\emptyset,\ (\lambda x^{\mathsf{Nm}}.x)(\mathsf{gensym}())) \Downarrow (\{\mathsf{n}\},\ \mathsf{n})}$$

Example 7 $((\emptyset,\ \mathsf{gensym}() = \mathsf{gensym}()))$**.** The equating of one fresh name with another fresh name evaluates as follows.

$$\frac{\frac{\frac{}{1 \quad (\emptyset,\ \mathsf{gensym}()) \Downarrow (\{\mathsf{n}\},\ \mathsf{n})}}{2 \quad (\{\mathsf{n}\},\ \mathsf{gensym}()) \Downarrow (\{\mathsf{n}, \mathsf{n}'\},\ \mathsf{n}')}}{3 \quad (\{\mathsf{n}, \mathsf{n}'\},\ \mathsf{n} = \mathsf{n}') \Downarrow (\{\mathsf{n}, \mathsf{n}'\},\ \mathsf{false})}}{4 \quad (\emptyset,\ \mathsf{gensym}() = \mathsf{gensym}()) \Downarrow (\{\mathsf{n}, \mathsf{n}'\},\ \mathsf{false})}$$

Example 8 $((\emptyset,\ (\lambda x^{\mathsf{Nm}}.x = x)(\mathsf{gensym}())))$**.** Comparing a fresh name with itself is clearly true as follows.

$$\frac{\frac{\frac{}{1 \quad (\emptyset,\ \mathsf{gensym}()) \Downarrow (\{\mathsf{n}\},\ \mathsf{n})}}{2 \quad (\{\mathsf{n}\},\ (\lambda x^{\mathsf{Nm}}.x = x)\mathsf{n}) \to (\{\mathsf{n}\},\ \mathsf{n} = \mathsf{n})}}{3 \quad (\{\mathsf{n}\},\ \mathsf{n} = \mathsf{n}) \to (\{\mathsf{n}\},\ \mathsf{true})}}{4 \quad (\emptyset,\ (\lambda x^{\mathsf{Nm}}.x = x)(\mathsf{gensym}())) \Downarrow (\{\mathsf{n}\},\ \mathsf{true})}$$

Example 9 $((\emptyset,\ \lambda x^{\mathsf{Nm}}.(\lambda y^{\mathsf{Nm}}.x = y)(\mathsf{gensym}())(\mathsf{gensym}())))$**.** The equating of one fresh name with another fresh name after being passed through a function, evaluates as follows.

$$\frac{\frac{\frac{\frac{\frac{}{1 \quad (\emptyset,\ \mathsf{gensym}()) \Downarrow (\{\mathsf{n}\},\ \mathsf{n})}}{2 \quad (\{\mathsf{n}\},\ (\lambda x^{\mathsf{Nm}}.(\lambda y^{\mathsf{Nm}}.x = y)(\mathsf{gensym}()))\mathsf{n}) \to (\{\mathsf{n}\},\ (\lambda y^{\mathsf{Nm}}.\mathsf{n} = y)(\mathsf{gensym}()))}}{3 \quad (\{\mathsf{n}\},\ \mathsf{gensym}()) \Downarrow (\{\mathsf{n}, \mathsf{n}'\},\ \mathsf{n}')}}{4 \quad (\{\mathsf{n}, \mathsf{n}'\},\ (\lambda x^{\mathsf{Nm}}.\mathsf{n} = x)\mathsf{n}') \to (\{\mathsf{n}, \mathsf{n}'\},\ \mathsf{n} = \mathsf{n}')}}{5 \quad (\{\mathsf{n}, \mathsf{n}'\},\ \mathsf{n} = \mathsf{n}') \Downarrow (\{\mathsf{n}, \mathsf{n}'\},\ \mathsf{false})}}{6 \quad (\emptyset,\ \lambda x^{\mathsf{Nm}}.(\lambda y^{\mathsf{Nm}}.x = y)(\mathsf{gensym}())(\mathsf{gensym}())) \Downarrow (\{\mathsf{n}, \mathsf{n}'\},\ \mathsf{false})}$$

Example 10 $((\emptyset,\ \mathsf{let}\ x = \mathsf{gensym}()\ \mathsf{in}\ x))$**.** Using $\mathsf{let}\ x = M\ \mathsf{in}\ x$ in the case below is identical to using the identity function on M in Ex. 6.

$$(\emptyset,\ \mathsf{let}\ x = \mathsf{gensym}()\ \mathsf{in}\ x) \Downarrow (\{\mathsf{n}\},\ \mathsf{n})$$

Example 11 $((\emptyset,\ \mathsf{let}\ x = \mathsf{gensym}\ \mathsf{in}\ x() = x()))$**.** Using gensym prior to application to () and as a function that will create fresh names gives an example that shows how extensionality fails for gensym.

1 $(\emptyset,\ \mathsf{gensym}) \Downarrow (\emptyset,\ \mathsf{gensym})$

---

2 $(\emptyset,\ \mathsf{let}\ x = \mathsf{gensym}\ \mathsf{in}\ x() = x()) \to (\emptyset,\ \mathsf{gensym}() = \mathsf{gensym}())$

---

3 $(\emptyset,\ \mathsf{let}\ x = \mathsf{gensym}\ \mathsf{in}\ x() = x()) \Downarrow (\{\mathsf{n}, \mathsf{n}'\},\ \mathsf{false})$ *See Ex. 7*

Example 12 $(\mathsf{let}\ x = \mathsf{gensym}()\ \mathsf{in}\ \mathsf{let}\ y = \mathsf{gensym}()\ \mathsf{in}\ x = y)$**.** Producing the two fresh name prior to comparing them in $\mathsf{let}\ x = \mathsf{gensym}()\ \mathsf{in}\ \mathsf{let}\ y = \mathsf{gensym}()\ \mathsf{in}\ x = y$ reduces as follows to false.

1 $(\emptyset,\ \mathsf{gensym}()) \Downarrow (\{\mathsf{n}\},\ \mathsf{n})$

---

2 $(\{\mathsf{n}\},\ \mathsf{gensym}()) \Downarrow (\{\mathsf{n}, \mathsf{n}'\},\ \mathsf{n}')$

---

3 $(\{\mathsf{n}, \mathsf{n}'\},\ \mathsf{n} = \mathsf{n}') \Downarrow (\{\mathsf{n}, \mathsf{n}'\},\ \mathsf{false})$

---

4 $(\emptyset,\ \mathsf{let}\ x = \mathsf{gensym}()\ \mathsf{in}\ \mathsf{let}\ y = \mathsf{gensym}()\ \mathsf{in}\ x = y) \Downarrow (\{\mathsf{n}, \mathsf{n}'\},\ \mathsf{false})$

Example 13 $(\mathsf{let}\ x = \mathsf{gensym}()\ \mathsf{in}\ \lambda y^{\mathsf{Nm}}.(x = y))$**.** A key example of a hidden name is $\mathsf{let}\ x = \mathsf{gensym}()\ \mathsf{in}\ \lambda y^{\mathsf{Nm}}.(x = y)$, reduced below. This function can never reveal the name stored at x, no matter how the function is used, as it is produced outside the λ-binder but only occurs deconstructed by equality inside the λ-binder.

1 $(\emptyset,\ \mathsf{gensym}()) \Downarrow (\{\mathsf{n}\},\ \mathsf{n})$

---

2 $(\emptyset,\ \mathsf{let}\ x = \mathsf{gensym}()\ \mathsf{in}\ \lambda y^{\mathsf{Nm}}.(x = y)) \Downarrow (\{\mathsf{n}\},\ \lambda y^{\mathsf{Nm}}.(\mathsf{n} = y))$

Any application of this function should always output false, as it must be applied to a name which can never be equivalent to n.

Example 14 $(\mathsf{let}\ x = \mathsf{gensym}()\ \mathsf{in}\ \langle \lambda y^{\mathsf{Nm}}.(x = y), x\rangle)$**.** In contrast to Ex. 13, the following example $\mathsf{let}\ x = \mathsf{gensym}()\ \mathsf{in}\ \langle \lambda y^{\mathsf{Nm}}.(x = y), x\rangle$ outputs a pair where the first element is the function above and the second element is the name at x. This is reduced below. This output does reveal the name stored at x, so the function in the first element of the pair can now potentially return true.

1 $(\emptyset,\ \mathsf{gensym}()) \Downarrow (\{\mathsf{n}\},\ \mathsf{n})$

---

2 $(\{\mathsf{n}\},\ \mathsf{let}\ x = \mathsf{n}\ \mathsf{in}\ \langle \lambda y^{\mathsf{Nm}}.(x = y), x\rangle) \Downarrow (\{\mathsf{n}\},\ \langle \lambda y^{\mathsf{Nm}}.(\mathsf{n} = y), \mathsf{n}\rangle)$

---

3 $(\emptyset,\ \mathsf{let}\ x = \mathsf{gensym}()\ \mathsf{in}\ \langle \lambda y^{\mathsf{Nm}}.(x = y), x\rangle) \Downarrow (\{\mathsf{n}\},\ \langle \lambda y^{\mathsf{Nm}}.(\mathsf{n} = y), \mathsf{n}\rangle)$

Example 15 (let $x = \mathsf{gensym}()$ in $\lambda y^{\mathsf{Nm}}.x$)**.** In contrast to Ex. 13, a simple example which reveals a name inside a λ-binder is let $x = \mathsf{gensym}()$ in $\lambda y^{\mathsf{Nm}}.x$, reasoned about below. Once evaluated the value will be a function that always returns the same name.

$$\frac{1 \quad (\emptyset,\ \mathsf{gensym}()) \Downarrow (\{\mathsf{n}\},\ \mathsf{n})}{2 \quad (\emptyset,\ \text{let } x = \mathsf{gensym}() \text{ in } \lambda y^{\mathsf{Nm}}.x) \Downarrow (\{\mathsf{n}\},\ \lambda y^{\mathsf{Nm}}.\mathsf{n})}$$

Example 16 (The "chain" example [68])**.** A set of functions that are indexed by some integer p, which creates p fresh names and cyclically permutes the names depending on the input name, can be defined by the functions Chain_p of type $\mathsf{Nm} \to \mathsf{Nm}$ as follows.

$$\begin{aligned}\mathsf{Chain}_p \stackrel{def}{=} \text{let } x_0,\ ..., x_p = \mathsf{gensym}() \text{ in } \lambda x^{\mathsf{Nm}}.\ & \text{if } x = x_0 \text{ then } x_1 \text{ else}\\ & \text{if } x = x_1 \text{ then } x_2 \text{ else}\\ & ...\\ & \text{if } x = x_p \text{ then } x_0 \text{ else } x_0\end{aligned}$$

As an example let $p = 2$ then:

- $(\emptyset,\ \mathsf{Chain}_2) \to (\{\mathsf{n}_0, \mathsf{n}_1, \mathsf{n}_2\},\ V_p)$
- $(\emptyset,\ \text{let } f = \mathsf{Chain}_2 \text{ in } f(\mathsf{gensym}())) \to (\{\mathsf{n}_0, \mathsf{n}_1, \mathsf{n}_2, \mathsf{n}\},\ \mathsf{n}_0)$
- $(\emptyset,\ \text{let } f = \mathsf{Chain}_2 \text{ in } ff(\mathsf{gensym}())) \to (\{\mathsf{n}_0, \mathsf{n}_1, \mathsf{n}_2, \mathsf{n}\},\ \mathsf{n}_1)$
- $(\emptyset,\ \text{let } f = \mathsf{Chain}_2 \text{ in } fff(\mathsf{gensym}())) \to (\{\mathsf{n}_0, \mathsf{n}_1, \mathsf{n}_2, \mathsf{n}\},\ \mathsf{n}_2)$
- $(\emptyset,\ \text{let } f = \mathsf{Chain}_2 \text{ in } ffff(\mathsf{gensym}())) \to (\{\mathsf{n}_0, \mathsf{n}_1, \mathsf{n}_2, \mathsf{n}\},\ \mathsf{n}_0)$
- ...

Clearly if $p \neq p'$ then $\mathsf{Chain}_p \not\cong^{\emptyset}_{\mathsf{Nm}\to\mathsf{Nm}} \mathsf{Chain}_{p'}$.

Example 17 (The "inaccessible chain" example)**.** This example is adapted from Ex. 16 but itself does not seem to appear in the literature. An adaptation to Ex. 16 that creates $p+1$ names with a cyclic dependency on p of these names but never gives access to any part of the cycle can be seen as follows. The function only gives access to a single name not in the cyclical chain part.

$$\begin{aligned}\mathsf{InaccessChain}_p \stackrel{def}{=} \text{let } x_0,\ ..., x_p, y = \mathsf{gensym}() \text{ in } \lambda x^{\mathsf{Nm}}.\ & \text{if } x = x_0 \text{ then } x_1 \text{ else}\\ & \text{if } x = x_1 \text{ then } x_2 \text{ else}\\ & ...\\ & \text{if } x = x_p \text{ then } x_0 \text{ else } y\end{aligned}$$

For each p clearly $\mathsf{InaccessChain}_p$ can access the name y but it needs access to some name at x_k with $0 \leq k \leq p$ to output another name in x_k with $0 \leq k \leq p$ so this circular dependency ensures that none of the names in x_k with $0 \leq k \leq p$ are actually accessible and only the name at y is accessible. For this reason $\mathsf{InaccessChain}_p \cong^{\emptyset}_{\mathsf{Nm}\to\mathsf{Nm}} (\mathsf{gensym}())$ should hold for any p.

Example 18 (The "lasso" example [68])**.** Another adaption of Ex. 16 is $\mathsf{lasso}_{(i,p)}$ defined below. $\mathsf{lasso}_{(i,p)}$ changes the final if-case name in chain_p to some n_i for $0 < i \leq \mathsf{n}$ forming a loop of names that starts after i number of calls to the function. If $i = 0$ then $\mathsf{Chain}_p \equiv \mathsf{lasso}_{(0,p)}$.

$$\begin{aligned}\mathsf{lasso}_{(i,p)} \stackrel{def}{=} \mathsf{let}\ x_0,\ ..., x_p = \mathsf{gensym}()\ \mathsf{in}\ \lambda x^{\mathsf{Nm}}.\ &\mathsf{if}\ x = x_0\ \mathsf{then}\ x_1\ \mathsf{else}\\ &\mathsf{if}\ x = x_1\ \mathsf{then}\ x_2\ \mathsf{else}\\ &...\\ &\mathsf{if}\ x = x_p\ \mathsf{then}\ x_i\ \mathsf{else}\ x_0\end{aligned}$$

As an example let $G_l \equiv \{\mathsf{n}_0, \mathsf{n}_1, \mathsf{n}_2, \mathsf{n}_3, \mathsf{n}\}$ then:

- $(\emptyset,\ \mathsf{let}\ f = \mathsf{lasso}_{(1,3)}\ \mathsf{in}\ f(\mathsf{gensym}())) \to (G_l,\ \mathsf{n}_0)$
- $(\emptyset,\ \mathsf{let}\ f = \mathsf{lasso}_{(1,3)}\ \mathsf{in}\ ff(\mathsf{gensym}())) \to (G_l,\ \mathsf{n}_1)$
- $(\emptyset,\ \mathsf{let}\ f = \mathsf{lasso}_{(1,3)}\ \mathsf{in}\ fff(\mathsf{gensym}())) \to (G_l,\ \mathsf{n}_2)$
- $(\emptyset,\ \mathsf{let}\ f = \mathsf{lasso}_{(1,3)}\ \mathsf{in}\ ffff(\mathsf{gensym}())) \to (G_l,\ \mathsf{n}_3)$
- $(\emptyset,\ \mathsf{let}\ f = \mathsf{lasso}_{(1,3)}\ \mathsf{in}\ fffff(\mathsf{gensym}())) \to (G_l,\ \mathsf{n}_1)$
- $(\emptyset,\ \mathsf{let}\ f = \mathsf{lasso}_{(1,3)}\ \mathsf{in}\ ffffff(\mathsf{gensym}())) \to (G_l,\ \mathsf{n}_2)$
- ...

Example 19 (The "hard" example)**.** An example that deserves some discussion is the "hard" example which has been discussed in [2, 52, 68] and is a recurring problem in many of these papers. The program is M_H below, which is simple but highlights the nuances of the ν-calculus.

$$M_H \stackrel{def}{=} \mathsf{let}\ x, x' = \mathsf{gensym}()\ \mathsf{in}\ \lambda f^{\mathsf{Nm}\to\mathsf{Bool}}.fx = fx'$$

Assume $(\emptyset,\ M_H) \to (\{\mathsf{n}, \mathsf{n}'\},\ \lambda f^{\mathsf{Nm}\to\mathsf{Bool}}.f\mathsf{n} = f\mathsf{n}' \stackrel{def}{=} V_H)$. One might expect V_H to hide the names n and n' under the λf-binder and hence the returning function should be equivalent to $\lambda f.\mathsf{true}$. One could verbally argue that "M_H cannot reveal n or n' and hence

any function of type $\mathsf{Nm} \to \mathsf{Bool}$ to which this is applied should not be able to distinguish n from n' so must treat them equally and hence returning true", however this is not entirely true. Although the names cannot be revealed from the function, the function itself can be used to internally compare the names. Consider the context $C_H[\cdot]$ defined as follows.

$$C_H[\cdot] \stackrel{def}{=} \mathsf{let}\ F = [\cdot]\ \mathsf{in}\ \mathsf{let}\ G = \lambda y^{\mathsf{Nm}}.F(\lambda z^{\mathsf{Nm}}.y = z)\ \mathsf{in}\ FG$$

The reduction of $C_H[M_H]$ is introduced below, where the outermost application of V_H returns true, but within the evaluation there are applications of V_H in lines 7-10 and lines 11-14 such that the applied V_H returns false. However, the two occurrences of applied V_H returning false are equated to each other in line 15 and hence the outermost application of V_H does return true as expected.

1	Let $G_n \stackrel{def}{=} \{\mathsf{n}, \mathsf{n}'\}$	
2	Let: $V_H \equiv \lambda f^{\mathsf{Nm}\to\mathsf{Bool}}.f\mathsf{n} = f\mathsf{n}'$ and $W_H \equiv \lambda x^{\mathsf{Nm}}.V_H(\lambda y^{\mathsf{Nm}}.x = y)$	
3	$(\emptyset,\ M_H) \to (G_n,\ V_H)$	
4	$(\emptyset,\ C_H[M_H]) \to (G_n,\ C_H[V_H])$	*Line.3*
5	$\to (G_n,\ \mathsf{let}\ G = W_H\ \mathsf{in}\ V_HG) \to (G_n,\ V_HW_H)$	
6	$\to (G_n,\ W_H\mathsf{n} = W_H\mathsf{n}')$	
7	$(G_n,\ W_H\mathsf{n}) \to (G_n,\ V_H(\lambda y^{\mathsf{Nm}}.\mathsf{n} = y)))$	
8	$\to (G_n,\ (\lambda y^{\mathsf{Nm}}.\mathsf{n} = y)\mathsf{n} = (\lambda y^{\mathsf{Nm}}.\mathsf{n} = y)\mathsf{n}')$	
9	$\to (G_n,\ (\mathsf{n} = \mathsf{n}) = (\mathsf{n} = \mathsf{n}'))$	
10	$\to (G_n,\ \mathsf{true} = \mathsf{false}) \to (G_n,\ \mathsf{false})$	
11	$(G_n,\ W_H\mathsf{n}') \to (G_n,\ V_H(\lambda y^{\mathsf{Nm}}.\mathsf{n}' = y)))$	
12	$\to (G_n,\ (\lambda y^{\mathsf{Nm}}.\mathsf{n}' = y)\mathsf{n} = (\lambda y^{\mathsf{Nm}}.\mathsf{n}' = y)\mathsf{n}')$	
13	$\to (G_n,\ (\mathsf{n}' = \mathsf{n}) = (\mathsf{n}' = \mathsf{n}'))$	
14	$\to (G_n,\ \mathsf{false} = \mathsf{true}) \to (G_n,\ \mathsf{false})$	
15	$(G_n,\ W_H\mathsf{n} = W_H\mathsf{n}') \to (G_n,\ \mathsf{false} = \mathsf{false}) \to (G_n,\ \mathsf{true})$	*Lines.10, 14*
16	$(\emptyset,\ C_H[M_H]) \to (G_n,\ \mathsf{true})$	*Lines.4-6, 15*

Because the outermost application of V_H does return true the contextual equivalence $M_H \cong^{PS}_{\mathsf{Nm}\to\mathsf{Bool}} \lambda f^{\mathsf{Nm}\to\mathsf{Bool}}.\mathsf{true}$ holds. Proving this equality was one of the key aims of several papers discussed in Sec. 3.4 [52, 68, 2].

Example 20 (The alternative "hard" example)**.** If Ex. 19 is altered by generating the second fresh name under the λ-binder then the result is the following function M'_H.

$$M'_H \stackrel{def}{=} \mathsf{let}\ z' = \mathsf{gensym}()\ \mathsf{in}\ \lambda f^{\mathsf{Nm}\to\mathsf{Bool}}.\mathsf{let}\ z = \mathsf{gensym}()\ \mathsf{in}\ fz = fz'$$

This clearly will not be contextually congruent to $\lambda f^{\mathsf{Nm}\to\mathsf{Bool}}.\mathsf{true}$ as the context $C_{H'}$ below, distinguishes these two programs as it applies the function twice to "re-use" the name n. The evaluation for $C_{H'}[M'_H]$ is seen below to show how it differs from Ex. 19. The two applications of V'_H returning different results in lines 12 and 17 alongside the final comparison of these two results in line 18.

$$C_{H'}[\cdot] \stackrel{def}{=} (\lambda F^{(\mathsf{Nm}\to\mathsf{Bool})\to\mathsf{Bool}}.F(\lambda x^{\mathsf{Nm}}.F(\lambda y^{\mathsf{Nm}}.x = y)))[\cdot]$$

1	Let $G_1 \stackrel{def}{=} \{\mathsf{n}\}$, $G_2 \stackrel{def}{=} \{\mathsf{n}, \mathsf{n}'\}$, $G_3 \stackrel{def}{=} \{\mathsf{n}, \mathsf{n}', \mathsf{n}''\}$, $G_4 \stackrel{def}{=} \{\mathsf{n}, \mathsf{n}', \mathsf{n}'', \mathsf{n}'''\}$	
2	Let: $V'_H \equiv \lambda f^{\mathsf{Nm}\to\mathsf{Bool}}.\mathsf{let}\ z = \mathsf{gensym}()\ \mathsf{in}\ f\mathsf{n} = fz$ and $W'_H \equiv \lambda x^{\mathsf{Nm}}.V'_H(\lambda y^{\mathsf{Nm}}.x = y)$	
3	$(\emptyset,\ M'_H) \to (G_1,\ V'_H)$	
4	$(\emptyset,\ C_{H'}[M'_H]) \to (G_1,\ C_{H'}[V'_H])$	*Line.3*
5	$\to (G_1,\ V'_H W'_H)$	
6	$\to (G_1,\ \mathsf{let}\ z = \mathsf{gensym}()\ \mathsf{in}\ W'_H\mathsf{n} = W'_H z)$	
7	$\to (G_2,\ W'_H\mathsf{n} = W'_H\mathsf{n}')$	
8	$(G_2,\ W'_H\mathsf{n}) \to (G_2,\ V'_H(\lambda y^{\mathsf{Nm}}.\mathsf{n} = y))$	
9	$\to (G_2,\ \mathsf{let}\ z = \mathsf{gensym}()\ \mathsf{in}\ ((\lambda y^{\mathsf{Nm}}.\mathsf{n} = y)\mathsf{n}) = ((\lambda y^{\mathsf{Nm}}.\mathsf{n} = y)z))$	
10	$\to (G_3,\ ((\lambda y^{\mathsf{Nm}}.\mathsf{n} = y)\mathsf{n}) = ((\lambda y^{\mathsf{Nm}}.\mathsf{n} = y)\mathsf{n}''))$	
11	$\to (G_3,\ (\mathsf{n} = \mathsf{n}) = (\mathsf{n} = \mathsf{n}''))$	
12	$\to (G_3,\ \mathsf{true} = \mathsf{false}) \to (G_n,\ \mathsf{false})$	
13	$(G_3,\ W'_H\mathsf{n}') \to (G_3,\ V'_H(\lambda y^{\mathsf{Nm}}.\mathsf{n}' = y))$	
14	$\to (G_3,\ \mathsf{let}\ z = \mathsf{gensym}()\ \mathsf{in}\ ((\lambda y^{\mathsf{Nm}}.\mathsf{n}' = y)\mathsf{n}) = ((\lambda y^{\mathsf{Nm}}.\mathsf{n}' = y)z))$	
15	$\to (G_4,\ ((\lambda y^{\mathsf{Nm}}.\mathsf{n}' = y)\mathsf{n}) = ((\lambda y^{\mathsf{Nm}}.\mathsf{n}' = y)\mathsf{n}'''))$	
16	$\to (G_4,\ (\mathsf{n}' = \mathsf{n}) = (\mathsf{n}' = \mathsf{n}'''))$	
17	$\to (G_4,\ \mathsf{false} = \mathsf{false}) \to (G_4,\ \mathsf{true})$	
18	$(G_n,\ W'_H\mathsf{n} = W'_H\mathsf{n}') \to (G_n,\ \mathsf{false} = \mathsf{true}) \to (G_n,\ \mathsf{false})$	*Lines.7, 8-12, 13-17*
19	$(\emptyset,\ C_{H'}[M'_H]) \to (G_n,\ \mathsf{false})$	*Lines.4-7, 18*

Clearly $(\emptyset,\ C_{H'}[\lambda f.\mathsf{true}]) \Downarrow (\emptyset,\ \mathsf{true})$ hence this context proves by counterexample that $M'_H \not\cong^{\emptyset}_{(\mathsf{Nm}\to\mathsf{Bool})\to\mathsf{Bool}} \lambda f.\mathsf{true}$ as expected.

3.2 The ν_{PS}-Calculus

The ν_{GS}-calculus is an adaptation of what is referred to here as the ν_{PS}-calculus introduced by Andrew **P**itts and Ian **S**tark in 1993 [52]. The ideas originated from π-calculus which uses similar notation and concepts in creating fresh channels using a $\nu n.P$ constructor. The introduction of the ν_{PS}-calculus lead to a flurry of research, e.g. [22, 19, 25, 51, 50, 70].

The language is briefly and informally introduced in Sec. 3.2.1 alongside the definition of contextual congruence. The similarities between the ν_{PS}-calculus and the ν_{GS}-calculus are intentional and the translations between the two versions are introduced in Sec. 3.3.

3.2.1 The Programming Language

The syntax of the ν_{PS}-calculus extends that of the STLC with the type Nm, and a constructor for names now in the form $\nu\mathsf{n}.M$, which binds a new name to n in M. The destructor of type Nm is again the equality operator $=$. Pairs and pair projection are not originally included; however, they are not expected to introduce any new issues.

Unlike the ν_{GS}-calculus, the binding of names now requires the definition of free names, similar to the definition of free variables. Free Names in the ν_{PS}-calculus are similar to the "all-names" function in the ν_{GS}-calculus. The key definition is that of $\mathsf{fn}(\nu\mathsf{n}.M) \stackrel{def}{=} \mathsf{fn}(M) \setminus \{\mathsf{n}\}$.

Type checking for the ν_{PS}-calculus follows that of the STLC but adds a nameset (a list of generated names) s, to the typing judgment as $s, \Gamma \vdash_\nu M : \alpha$. This means M has type α in the type context Γ, with names occurring freely in M contained in the nameset s. Adding name n to the nameset s is written $s \oplus \{\mathsf{n}\}$ and adding two namesets s and s' is written similarly as $s \oplus s'$.

The key new typing rules are those including names provided in Fig. 3.6.

$$\frac{(\mathsf{n} \in s)}{s, \Gamma \vdash_\nu \mathsf{n} : \mathsf{Nm}} \qquad \frac{s \oplus \{\mathsf{n}\}, \Gamma \vdash_\nu M : \alpha}{s, \Gamma \vdash_\nu \nu\mathsf{n}.M : \alpha} \qquad \frac{s, \Gamma \vdash_\nu M : \mathsf{Nm} \quad s, \Gamma \vdash_\nu N : \mathsf{Nm}}{s, \Gamma \vdash_\nu M = N : \mathsf{Bool}}$$

Figure 3.6: Key new typing rules of the ν_{PS}-calculus.

This type check performs two checks. Firstly that the free names used are part of the nameset, secondly that the variables are correctly typed by the type context. This differs from the ν_{GS}-calculus, which splits the two checks by using valid configurations to check the names in the term exist in the nameset and the classic type check for a check on the

variables. The two approaches are equivalent.

As with the typing rules, the reduction relation is also extended from the STLC with a nameset s. The reduction relation becomes $s \vdash M \to (s')\ M'$ and states that the term M with the $\mathsf{fn}(M) \in s$ reduces to a term M' with free names in $s \oplus s'$. Substitution also builds on the STLC definition (Def. 3) to now include names in the terms and values being substituted, as $M[V/x]$ substitutes V for the free occurrences of variable x in M whilst avoiding capturing free variables and names in V.

The operational semantics for the ν_{PS}-calculus is defined using the same evaluation contexts as those of the STLC in Def. 8. The key new reduction rules are those involving names seen in Fig. 3.7.

$$s \vdash \mathsf{n} = \mathsf{n} \to (\emptyset)\ \mathsf{true}$$
$$s \vdash \mathsf{n} = \mathsf{n}' \to (\emptyset)\ \mathsf{false} \quad \mathsf{n} \neq \mathsf{n}'$$
$$s \vdash \nu\mathsf{n}.M \to (\{\mathsf{n}\})\ M$$

Figure 3.7: Reduction rules of the ν_{PS}-calculus.

Bound names can be "α-converted" if required which replaces one of the binding names and any occurrence of that name in a term for another unused name. For any M with fresh names n an n' not occurring bound or free in M, then $\nu\mathsf{n}.M[\mathsf{n}/y] \equiv_\alpha \nu\mathsf{n}'.M[\mathsf{n}'/y]$. This removes the focus on the precise name being used and emphasises the binding relationship and when names are generated similar to the α-equivalence for variables.

Contextual Equivalence in the ν_{PS}-calculus

The definition of *contextual equivalence* for the ν_{PS}-calculus is the standard definition for the STLC in Def. 10 with an adaption to include the names.

Definition 33 (Single holed contexts ν_{PS}-calculus)**.** *The single-holed contexts of the STLC in Sec. 2.1.3 are extended to include the new constructors of the ν_{PS}-calculus as follows. The equality context is included as this now equates names as well as Booleans.*

$$\mathsf{C}[\cdot]_\alpha \quad ::= \quad \ldots \mid \nu\mathsf{n}.\mathsf{C}[\cdot]_\alpha \mid \mathsf{C}[\cdot]_\alpha = M \mid M = \mathsf{C}[\cdot]_\alpha$$

Any context filled with an appropriately typed term is a ν_{PS}-calculus term and can thus be typed accordingly. Contexts can be typed using the typing rules in Fig. 3.6 and

the following rule to type holes.

$$\frac{-}{s, \Gamma \vdash_\nu [\cdot]_\alpha : \alpha}$$

A context $C[\cdot]$ is *M closing* if it binds all free variables in a term M, hence $s, \emptyset \vdash C[M] : \alpha$. Contextual equivalence can now be defined as follows.

Definition 34 (Contextual equivalence (ν_{PS}-calculus))**.** *Two possibly open terms M and N of type α, that are both typed by s, Γ (i.e. $s, \Gamma \vdash M : \alpha$ and $s, \Gamma \vdash N : \alpha$) are defined as contextually equivalent, written $s, \Gamma \vdash M \cong_\alpha^{PS} N$, if any closing context of both M and N is unable to distinguish these terms and is formally defined as follows.*

$$s, \Gamma \vdash M \cong_\alpha^{PS} N \quad \stackrel{\text{def}}{=} \quad \forall\, \mathsf{C}[\cdot]_\alpha.\ s, \emptyset \vdash_\nu \mathsf{C}[M]_\alpha : \mathsf{Bool} \ \wedge\ s, \emptyset \vdash_\nu \mathsf{C}[N]_\alpha : \mathsf{Bool}$$
$$\rightarrow \begin{pmatrix} (\exists\, s'.\ s \vdash \mathsf{C}[M] \Downarrow (s')\ \mathsf{true}) \\ \leftrightarrow \\ (\exists\, s''.\ s \vdash \mathsf{C}[N] \Downarrow (s'')\ \mathsf{true}) \end{pmatrix}$$

3.3 Relation Between the ν_{GS}-Calculus and the ν_{PS}-Calculus

The formulations of ν_{GS}-calculus and ν_{PS}-calculus are equivalent [68]. The ν_{PS}-calculus combines the fresh name generation with the scope of the name, whereas the ν_{GS}-calculus separates these two aspects and uses the λ-abstraction as the basis for the scope of the names. These differences can help in thought processes depending on the application. The term $\mathsf{gensym}()$ is sometimes replaced by the term new [68, 2], however in this thesis the gensym version is used.

One slight difference between the two formulations is that the ν_{PS}-calculus allows for the binding of names within a single-holed context, i.e. $\mathsf{C}[\cdot] \equiv \nu \mathsf{n}.\mathsf{C}'[\cdot]$ is permitted whereas ν_{GS}-calculus cannot bind specific names. However, it seems these contexts are not intended by the definition, or used in practice [68, 62].

A translation from ν_{GS}-calculus to ν_{PS}-calculus, written $\langle\!\langle \cdot \rangle\!\rangle_{GS \to PS}$ can be defined inductively with the key translation being $\langle\!\langle \mathsf{gensym} \rangle\!\rangle_{GS \to PS} \stackrel{def}{=} \lambda().\nu \mathsf{n}.\mathsf{n}$. The reverse translation from ν_{PS}-calculus to ν_{GS}-calculus, written $\langle\!\langle \cdot \rangle\!\rangle_{PS \to GS}$, can be defined inductively with the key case being $\langle\!\langle \nu \mathsf{n}.M \rangle\!\rangle_{PS \to GS} \stackrel{def}{=} \mathsf{let\ n} = \mathsf{gensym}()\ \mathsf{in}\ \langle\!\langle M \rangle\!\rangle_{PS \to GS}$. Ignoring pairs and projections from the ν_{GS}-calculus, the other cases to both these translations are trivial.

This leads to the following statements for any terms M and N in the respective lan-

guages (denoted by the $X \in \{PS,\ GS\}$ in M_X) .

$$\begin{array}{rcl} M_{GS} & \cong_\alpha^G & \langle\!\langle \langle\!\langle M_{GS} \rangle\!\rangle_{GS\to PS} \rangle\!\rangle_{PS\to GS} \\ G \vdash M_{PS} & \cong_\alpha^{PS} & \langle\!\langle \langle\!\langle M_{PS} \rangle\!\rangle_{PS\to GS} \rangle\!\rangle_{GS\to PS} \\ M_{GS} \cong_\alpha^G N_{GS} & \leftrightarrow & G \vdash \langle\!\langle M_{GS} \rangle\!\rangle_{GS\to PS} \cong_\alpha^{PS} \langle\!\langle N_{GS} \rangle\!\rangle_{GS\to PS} \\ G \vdash M_{PS} \cong_\alpha^{PS} N_{PS} & \leftrightarrow & \langle\!\langle M_{PS} \rangle\!\rangle_{PS\to GS} \cong_\alpha^G \langle\!\langle N_{PS} \rangle\!\rangle_{PS\to GS} \end{array}$$

The proof of these statements are trivial.

Hence all programs (and equivalences) in Sec. 3.1.2 can be directly translated to examples in the ν_{PS}-calculus, and vice versa.

3.4 Proof Techniques of Contextual Equivalence in the ν_{PS}-Calculus

Reasoning about programs often means proving contextual equivalence. Proof techniques are introduced to prove certain classes of contextual equivalence for the ν_{PS}-calculus. A key sticking point in many of the contextual equivalence proofs is the *"hard" example equivalence* (Ex. 19) defined as follows.

$$\emptyset, \emptyset \vdash \nu \mathsf{n}_1.\nu \mathsf{n}_2.\lambda f^{\mathsf{Nm}\to\mathsf{Bool}}.(f\mathsf{n}_1 = f\mathsf{n}_2) \cong^{PS}_{(\mathsf{Nm}\to\mathsf{Bool})\to\mathsf{Bool}} \lambda f.\mathsf{true}$$

This needs special attention due to the reuse of hidden names as seen in Ex. 19.

Alternative proof techniques to prove contextual equivalence are sought after, which allow for smaller, more manageable proofs. It is still not known if contextual equivalence is decidable in the ν_{PS}-calculus for all terms [52, 68, 2], however, it is proven that contextual equivalence is decidable for first order types [52]. A simpler proof of contextual equivalence would be useful, and various attempts at more manageable definitions are introduced in [62, 52, 74, 2, 68] these are introduced and discussed in the following subsections.

Given most of the research is focused on the ν_{PS}-calculus variant the rest of this subsection will focus only on this variant. All proof techniques here for the ν_{PS}-calculus are expected to hold in the ν_{GS}-calculus, given the equivalence of the languages.

3.4.1 Equational Logic

In 1998 Ian Stark introduced an equational logic using the CBV β-equivalence, η-equivalence and various other rules to construct the equivalence relation using derivation rules [63]. In equational logics, assertions are of the form $s, \Gamma \vdash M \cong_{EqLog(\alpha)} N$. Equational assertions can be derived using inductive rules

This equational reasoning of two terms implies contextual equivalence and corresponds exactly with contextual equivalence at first order types and ground types. i.e. deriving $s, \Gamma \vdash M \cong_{EqLog(\alpha)} N$ implies $s, \Gamma \vdash M \cong_{\alpha}^{PS} N$, with the opposite direction holding for α a first order or ground type.

This logic is sound, however it cannot distinguish between public and private names, hence fails to prove certain equivalences such as $(\nu \mathsf{n}.\lambda x.x = \mathsf{n}) \cong^{PS}_{\mathsf{Nm}\to\mathsf{Bool}} \lambda x.\mathsf{false}$ meaning it is not complete. This is extended to a more powerful relational logic in [63], similar to the logical relations introduced in Sec. 3.4.2 but with a set of rules to derive these relations.

3.4.2 Logical Relations

In general, logical relations allow for the reasoning of properties such as termination, type safety and contextual equivalence by introducing an intermediate layer of reasoning. A logical relation is a unary, binary or n-ary, type-indexed relation between terms in a language, with suitable closure properties. Termination for the STLC was proven using logical relations [65], and its applications have steadily grown (in number and complexity) over time. Logical relations are typically used for proofs regarding the STLC and its extensions, and have been adapted to the ν_{PS}-calculus in [52, 63]. The ν_{PS}-calculus was proven to terminate using logical relations in [52].

To prove contextual equivalence in the ν_{PS}-calculus, a binary logical relation R_α between two ν_{PS}-calculus values of type α, written $V_1 \; R_\alpha \; V_2$, is defined. The relation R_α has a span written as $R_\alpha : s_1 \leftrightharpoons s_2$, which is an injective partial map from the nameset s_1 to the nameset s_2. The span states which names are public (and hence which are private), and states how the names are related from one value to the other.

The identity partial bijection id^*_α for any nameset, is a logical relation which implies contextual equivalence. However, the contextual equivalence is only proven to imply the identity relation at first order.

Logical relations fail to prove the hard example equivalence, hence an extension to logical relations: *predicated logical relations* was introduced for this specific form of equivalence. Predicated logical relations introduce an extra symmetry of names for each term being related. The identity partial bijection in this new relation is proven to imply contextual equivalence. This can now be used to derive the "hard" example equivalence above. Predicated logical relations are also only complete up to first order.

Further extensions to logical relations are possible and may provide extra insight in this context, but it is suggested the complexity and lack of completeness makes these possible

extensions unworthy of further research [62].

3.4.3 Kripke Logical Relations

A Kripke logical relations derived from the categorical model of the ν_{PS}-calculus and the general notion of Kripke logical relations is equally powerful to Pitts and Starks logical relations in [52] at first order types [74].

> "The failure of the equivalence for high-order function types is due to the non-fully abstract categorical model, while a similar Kripke logical relation defined directly on the computational metalanguage has been shown to be equivalent to the operational logical relation for any type [73]" [74]

This is not discussed any further here.

3.4.4 Environmental Bisimulations

Pitts and Starks' predicated logical relations [63] are derived precisely for examples such as the "hard" example equivalence. A more general approach is introduced in [2, 68], using environmental bisimulations to prove the 'hard" example equivalence and more complex equivalences.

A bisimulation for the extension of the ν_{PS}-calculus which includes assignment was introduced and proven complete but not sound [31], however the "hard" example equivalence fails to hold as the context can distinguish these functions with side-effects.

Environmental bisimulations [64, 34], extend bisimulations as a set of relations as opposed to a single relation. The set of relations grows as the environment of the terms being equated changes.

A sound and complete theory for reasoning about (contextual) equivalence in the ν_{PS}-calculus (without assignment) is provided in [2]. A theory of adequate sets of relations is developed, such that the largest adequate set of relations coincides with contextual equivalence. A set of proof obligations are given, which need to be satisfied to ensure a set is adequate. The resulting proofs of equivalence are simpler compared to other methods.

All examples in [62] are provable using this technique, including the hard example, which uses two applications of this technique, establishing the equivalence through the proof of another equivalence. The proofs in [2] are all proven in Coq, providing a cast iron level of certainty that these hold.

3.4.5 Nominal Games

Call-by-value games provide a semantics for the STLC [27]. Call-by-value games are generalised to nominal games, giving the first fully-abstract model for the ν_{PS}-calculus was first proposed in [1], but there was a bug which was then fixed in [68]. The contextual equivalence coincides with the equational theory provided by the model. A further extension to a fully-abstract model for a language with nominal general references was introduced in [67]. Discrepancies in the proof for the hard example from [1], were discussed and rectified in [68]. This model, although fully abstract, is complicated to use and does not provide a general method for deriving equivalences.

3.4.6 Probabilistic Programming Semantics for Name-Generation

In [60], a model of probabilistic programming called quasi-Borel spaces is used to model the ν-calculus. This takes the idea that each name produced in the ν_{PS}-calculus can be thought of as a random number from the reals ($\mathbb{R}$), and hence to all intents and purposes will always be distinct. These semantics are sound and also fully abstract up to first order-types meaning any two ν_{PS}-calculus programs are observationally equivalent if and only if their interpretations are observationally equivalent in the quasi-Borel spaces.

3.5 The $\lambda\nu$-Calculus

In 1993 Martin Odersky introduced a call-by-name variation of the ν_{PS}-calculus, called the $\lambda\nu$-calculus [46] (later [47]). The $\lambda\nu$-calculus does not contain state (i.e. the namesets in the ν_{PS}-calculus and ν_{GS}-calculus), however does allow λ and ν to commute unlike in the ν_{PS}-calculus.

Definition 35 (Syntax ($\lambda\nu$-calculus))**.** *The $\lambda\nu$-calculus types are identical to the ν_{PS}-calculus extending the STLC types with that of* Nm*. The typing context maps variables to types and names to type* Nm *which simplifies the typing judgment for the language somewhat.*

The $\lambda\nu$-calculus syntax of terms is identical to that of the ν_{PS}-calculus with the same ν-binder, an infinite set of names, and the same equating operator $=$. The values however are different, the $\lambda\nu$-calculus does not allow $\nu\mathsf{n}.\mathsf{n}$ or n (a name) as values but instead requires all programs to be of Boolean type. This ensures all names are always compared within the scope of a ν-binder, meaning the programs $\nu\mathsf{n}.\mathsf{n}$ and $(\nu\mathsf{n}.\mathsf{n}) = (\nu\mathsf{n}'.\mathsf{n}')$ (which are both common in ν_{PS}-calculus) are terms which get "stuck" in the original $\lambda\nu$-calculus. This

reflects that the identity of the name is known only within its scope. A reduced version of the syntax is introduced in Fig. 3.8.

$$
\begin{array}{lcl}
\alpha & ::= & \mathsf{Bool} \mid \mathsf{Nm} \mid \alpha \to \alpha \\
\Gamma & ::= & \emptyset \mid \Gamma, x : \alpha \mid \Gamma, \mathsf{n} : \mathsf{Nm} \\
M & ::= & c \mid \mathsf{n} \mid x \mid \lambda x.M \mid MM \mid \nu \mathsf{n}.M \mid M = M \\
V & ::= & c \mid \mathsf{n} \mid x \mid \lambda x.M
\end{array}
$$

Figure 3.8: Syntax of the $\lambda\nu$-calculus.

Although pairs, and various primitive operations on pairs are included in the original paper [46], they are not included here for brevity. The operator `name?` which checks if a term is a name is also not included here but performs a check on a term to see if it is a name.

Definition 36 (Typing rules ($\lambda\nu$-calculus))**.** *$\lambda\nu$-calculus terms are typed using the STLC typing judgment $\Gamma \vdash M : \alpha$, not including the nameset in the typing judgement as in the ν_{PS}-calculus. The typing context maps variables to types and names to type Nm. The typing rules are introduced in Fig. 3.9.*

$$
\frac{-}{\Gamma, x : \alpha \vdash x : \alpha}\ \textsc{Var} \qquad
\frac{\Gamma, x : \alpha \vdash M : \alpha'}{\Gamma \vdash \lambda x.M : \alpha \to \alpha'}\ \textsc{Lam} \qquad
\frac{\Gamma \vdash M : \alpha' \to \alpha \quad \Gamma \vdash N : \alpha'}{\Gamma \vdash MN : \alpha}\ \textsc{App}
$$

$$
\frac{-}{\Gamma, \mathsf{n} : \mathsf{Nm} \vdash \mathsf{n} : \mathsf{Nm}}\ \textsc{Name} \qquad
\frac{\Gamma, \mathsf{n} : \mathsf{Nm} \vdash M : \alpha}{\Gamma \vdash \nu \mathsf{n}.M : \alpha}\ \textsc{Nu} \qquad
\frac{\Gamma \vdash M : \mathsf{Nm} \quad \Gamma \vdash N : \mathsf{Nm}}{\Gamma \vdash M = N : \mathsf{Bool}}\ \textsc{Eq}
$$

Figure 3.9: Typing rules of the $\lambda\nu$-calculus.

The reduction relation for the $\lambda\nu$-calculus is based on the CBN STLC reduction relation, relating two terms M and M' as $M \to_{\lambda\nu} M'$. A nameset is not required because the name binder is never deconstructed to create names as in the ν_{PS}-calculus.

Definition 37 (Evaluation contexts ($\lambda\nu$-calculus))**.** *Evaluation contexts (of the Felleisen-Heib style [18]) are defined below in Fig. 3.10.*

$$\mathcal{E}[\cdot] \quad ::= \quad [\cdot] \mid \mathcal{E}[\cdot]\, M \mid \nu \mathsf{n}.\, \mathcal{E}[\cdot]$$

Figure 3.10: Evaluation contexts of the $\lambda\nu$-calculus.

The first three cases above are as expected for evaluation contexts for the CBN λ-calculus and the final case $\nu\mathsf{n}.\mathcal{E}[\cdot]$, ensures evaluations occur inside ν-binders which does not occur in the ν_{PS}-calculus.

Definition 38 (Reduction rules ($\lambda\nu$-calculus))**.** *The reduction rules are introduced in Fig. 3.11.*

$$\begin{array}{rcll}
(\lambda x.M)N & \to_{\lambda\nu} & [N/x]M & \\
\mathsf{n} = \mathsf{n} & \to_{\lambda\nu} & \mathsf{true} & \\
\mathsf{n} = \mathsf{n}' & \to_{\lambda\nu} & \mathsf{false} & (\mathsf{n} \neq \mathsf{n}') \\
\nu\mathsf{n}.\lambda x.M & \to_{\lambda\nu} & \lambda x.\nu\mathsf{n}.M & \\
\nu\mathsf{n}.\mathsf{n}' & \to_{\lambda\nu} & \mathsf{n}' & \\
\nu\mathsf{n}.\mathsf{n} & \not\to_{\lambda\nu} & & \\
M \to_{\lambda\nu} N & \to & \mathcal{E}[M] \to_{\lambda\nu} \mathcal{E}[N] &
\end{array}$$

Figure 3.11: Reduction rules of the $\lambda\nu$-calculus.

The first three rules are standard Call-By-Name lambda calculus rules with equality (adapted to include names). The λ-binder and ν-binder commute in only one direction via the fourth rule, which is not permitted in the ν_{PS}-calculus. The fifth rule shows the absorption of the ν-binder by a distinct name. The penultimate rule states that $\nu\mathsf{n}.\mathsf{n}$ cannot be reduced further, hence $\lambda\nu$-calculus with this rule "fails to satisfy the progress part type soundness" [36]. Similar to α-convertibility for bound variables α-convertibility for names ensures bound names can be swapped and specific names are not important. The final rule states the reduction via evaluation contexts.

The $\lambda\nu$-calculus can be extended to model local state with mutable local variables via an extension of the type Nm to $\mathsf{Nm}(\alpha)$, for some type α, allowing the name to map to a term of type α.

Contextual equivalence in the $\lambda\nu$-calculus is defined as for the ν_{PS}-calculus. The

primary difference when compared to ν_{PS}-calculus is that in this case because $\nu \mathsf{n}.\mathsf{n} = \nu \mathsf{n}'.\mathsf{n}'$ gets "stuck" then the definition clearly falters in some aspects.

Definition 39 (Contextual equivalence ($\lambda\nu$-calculus))**.** *Contextual equivalence of two terms M and N is defined if for all contexts $C[\cdot]$ such that $C[M]$ and $C[N]$ are both closed, then $C[M] \to^*_{\lambda\nu} \mathsf{true}$ iff $C[N] \to^*_{\lambda\nu} \mathsf{true}$.*

The $\lambda\nu$-calculus is proven to be a conservative observational extension of the λ-calculus. There is a syntactic embedding of the $\lambda\nu$-calculus in the λ-calculus, which uses a de Bruijn indices style translation to maintain the number of "levels" between the name and ν-binder. This assumes some form of integers in the λ-calculus and the embedding. Although simple, this translation requires a lengthy proof to show the semantics are preserved [47].

3.6 Relating ν_{PS}-Calculus and $\lambda\nu$-Calculus

The two languages ν_{PS}-calculus and $\lambda\nu$-calculus are compared in [36], with the ν_{PS}-calculus being translated using a Continuation Passing Style (CPS) translation into the $\lambda\nu$-calculus.

To make the formalities easier, both calculi are rewritten in an equivalent big-step operational semantics and various syntactic alterations are made.

The only alteration to the language in Löschs $\lambda\nu$-calculus [36] compared to Oderskys $\lambda\nu$-calculus [47] is that the former now defines $\nu \mathsf{n}.\mathsf{n}$ to be a canonical value and similarly $(\nu \mathsf{n}.\mathsf{n}) = (\nu \mathsf{n}.\mathsf{n})$ is chosen to evaluate to true, ensuring type soundness and totality of this variant of $\lambda\nu$-calculus. The later choice of $(\nu \mathsf{n}.\mathsf{n}) = (\nu \mathsf{n}.\mathsf{n}) \to_{\lambda\nu} \mathsf{true}$, must presumably be matched with a similar choice of $(\nu \mathsf{n}.(\mathsf{n} = (\nu \mathsf{n}'.\mathsf{n}')) \to_{\lambda\nu} \mathsf{false}$, however this is not clarified.

A translation from the ν_{PS}-calculus to the $\lambda\nu$-calculus is introduced using CPS, a common extension to the standard CBV to CBN translation [53]. This translation is proven *computationally adequate*, defined as:

(i) All Boolean typed terms in ν_{PS}-calculus and their translation to the $\lambda\nu$-calculus evaluate to the same Boolean constant.

(ii) Two terms of any type that are observationally equivalent (in the ν_{PS}-calculus) must translate to two terms in the $\lambda\nu$-calculus that are observationally equivalent (in the $\lambda\nu$-calculus).

The opposite direction translation from the $\lambda\nu$-calculus to the ν_{PS}-calculus is sketched (in a similar CPS-form) but not covered in detail in the paper.

The translations used between ν_{PS}-calculus and $\lambda\nu$-calculus fails to satisfy full abstraction, that is: "the target language must not be able to observe more about a translated term than is possible in the source language". The two contextually congruent ν_{PS}-calculus programs $\lambda f.(\lambda x.\mathsf{true})(f\,\mathsf{true})$ and $\lambda f.\mathsf{true}$ do not translate to contextually congruent terms in the $\lambda\nu$-calculus as the translation can be distinguished by a context, "this failure of full abstraction has more to do with the nature of continuation-passing transformations than with locally scoped names" [36].

3.7 Summary

The rest of the thesis will be work based on the ν_{GS}-calculus, which has been introduced here. The original ν_{PS}-calculus is introduced as the primary language discussed in the literature, however the two languages ν_{GS}-calculus and ν_{PS}-calculus are proven equivalent. Numerous proof techniques for proving contextual equivalence in the ν_{PS}-calculus are summarised, showing the difficulty that the simple language with names presents. The syntax, typing, reduction relation, and contextual equivalence are introduced for both of these languages. Numerous example programs in the ν_{GS}-calculus are introduced and reduced to show the intricacies of the language. Many of these programs will be reasoned about using the ν-logic.

A summary of the $\lambda\nu$-calculus is provided which extends the CBN version fo the STLC with fresh name generation. The relation between the $\lambda\nu$-calculus and the ν_{PS}-calculus, as summarised in the literature, is also introduced.

Chapter 4

Logical Language

The ν_{GS}-calculus extends the STLC, hence the program logic for the ν_{GS}-calculus (or the ν-logic), is inspired by the λ-logic, with the relevant changes to reason about names.

In this chapter the logical language is introduced. The logical syntax is introduced in Sec. 4.1, with typing of the logical constructs introduced in Sec. 4.2. Two versions of substitution in the ν-logic are introduced in Sec. 4.3, these are inspired by logical substitution in the λ-logic. Two properties of logical formulae are (syntactically) introduced in Sec. 4.4 those are: extension independence and thinness with respect to variables.

The axioms for the logic (also referred to as the the *logic of axioms*) are introduced in Sec. 4.5. The axioms are split into their primary logical constructors i.e. equality, restricted quantification, freshness, quantification over LTCs and evaluation formulae, in that order.

The rules for the logic allow for the reasoning of programs and are introduced in Sec. 4.6. This *logic of rules* introduces classes of rules: core language rules based on the programming language constructors, structural rules and derived rules.

Finally, in Sec. 4.7, a summary of the alternative options regarding the design of this logic is provided.

4.1 Logical Syntax

The logical syntax for the ν_{GS}-calculus builds on the λ-logic, with an adaptation of universal quantification and a new logical operator to quantify over future states with the added benefit of naming the future state. *Expressions*, ranged over by e,e', ..., *formulae*, ranged over by A, B, C, ... and *Logical Type Contexts* (LTCs), ranged over by $\mathbb{\Gamma}$, $\mathbb{\Gamma}'$, $\mathbb{\Gamma}_i$, ... , are defined in Fig. 4.1. The extensions over the λ-logic are those of LTCs and the final two logical operators.

$$
\begin{array}{lcl}
e & ::= & x^{\alpha} \mid c \mid \langle e, e \rangle \mid \pi_i(e) \\
\mathbb{\Gamma} & ::= & \emptyset \mid \mathbb{\Gamma}{+}x : \alpha \mid \mathbb{\Gamma}{+}\delta : \mathbb{TC} \\
A & ::= & e = e \mid \neg A \mid A \wedge A \mid e \bullet e = x^{\alpha}\{A\} \mid \forall x^{\alpha} \in (\mathbb{\Gamma}).A \mid \forall \delta.A
\end{array}
$$

Figure 4.1: Syntax of expressions, LTCs and formulae of the ν-logic.

Expressions, e, are standard as in Sec. 2.2.2, where constants, c, range over Boolean constants true and false and unit constant $()$. Names are not included directly in the expressions and can only be referenced via use of variables. For reasons explained in Sec. 2.2.2 there is no expression $e = e'$; however, it is sometimes treated as an expression for brevity (see $[\text{EQ}]_{\nu}$). The new logical constructs are now explained in detail.

4.1.1 Logical Type Contexts (LTCs)

LTCs extend standard type contexts (STCs) which map variables to types in two ways: LTCs are *ordered*, and LTCs also map *type context variables* (TCVs) to the type $\mathbb{TC}$ (meaning **T**ype **C**ontext). TCVs range over δ, δ', δ_i, ... and are always mapped to the new type $\mathbb{TC}$. This type $\mathbb{TC}$ is normally dropped as it only applies to TCVs.

A key concept is *derivable* expressions from an LTC. Syntactically, an expression is derivable from an LTC if it is typed by the LTC. Derivations ensure names are not revealed when previously hidden.

Consider the function $\lambda x.\langle x, \mathsf{n} \rangle$, the name n is derivable from this function. Now consider the function $\lambda x.x = \mathsf{n}$, the name n can never be derived from this function as it is deconstructed under a λ-binder. Hence, if an LTC contains a variable which represents one of these functions (and no other occurrence of n), then the values derived from that function must use n in the form the function provides it as. This means n may occur freely in the former case, but only in the form $\lambda x.x = \mathsf{n}$ in the latter. In the next chapter, a semantic definition of a value derivable from an LTC (and a model) will be introduced, which has a more concrete meaning.

The shorthand notation $\mathbb{\Gamma}{+}\mathbb{\Gamma}'$ represents the LTC $\mathbb{\Gamma}'$ added to the LTC $\mathbb{\Gamma}$ such that the order is maintained if $\mathbb{\Gamma}$ and $\mathbb{\Gamma}'$ have disjoint domains, formally defined as follows.

$$
\begin{array}{rcl}
\mathbb{\Gamma}{+}\emptyset & \stackrel{def}{=} & \mathbb{\Gamma} \\
\mathbb{\Gamma}{+}(x : \alpha{+}\mathbb{\Gamma}') & \stackrel{def}{=} & (\mathbb{\Gamma}{+}x : \alpha){+}\mathbb{\Gamma}' \\
\mathbb{\Gamma}{+}(\delta{+}\mathbb{\Gamma}') & \stackrel{def}{=} & (\mathbb{\Gamma}{+}\delta){+}\mathbb{\Gamma}'
\end{array}
$$

Definition 40 (Syntactic LTC extensions)**.** *Define the LTC* $\mathbb{\Gamma}'$ *as an* extension *of* $\mathbb{\Gamma}$ *if there exists some* $\mathbb{\Gamma}''$ *such that* $\mathbb{\Gamma}' \equiv \mathbb{\Gamma}+\mathbb{\Gamma}''$*. The LTC* $\mathbb{\Gamma}$ *is then a* contraction *of* $\mathbb{\Gamma}'$*.*

LTCs are the basis for typing expressions, formulae, triples and LTCs themselves. The ordering in LTCs is essential because of the new TCVs. The LTC $\mathbb{\Gamma}+x : \alpha$, means that x is a value of type α, derived from the LTC to its left i.e. $\mathbb{\Gamma}$. Hence the "+" for LTCs is not commutative. In a similar fashion, the LTC $\mathbb{\Gamma}+\delta$, implies that the δ represents some *extension* of the LTC to its left i.e. $\mathbb{\Gamma}$, hence again the "+" for LTCs is not commutative. This will become more apparent when the model is introduced in Chapt. 5.

Definition 41 (Actions on LTCs)**.** *The following actions on LTCs are defined as expected.*

- *Mapping variable* x *to its type in* $\mathbb{\Gamma}$ *written* $\mathbb{\Gamma}(x)$*. The same applies to TCVs,* $\mathbb{\Gamma}(\delta)$ *which always return* $\mathbb{TC}$*.*
- *Obtaining the domain of an LTC* $\mathbb{\Gamma}$ *written* $\mathsf{dom}(\mathbb{\Gamma})$*, defined as all variables and TCVs mapped by the LTC. The codomain is defined similarly and written* $\mathsf{cod}(\mathbb{\Gamma})$*.*
- *Removal of a variable* x *from an LTC* $\mathbb{\Gamma}$ *written* $\mathbb{\Gamma}\backslash x$ *maintains the order of the original LTC. Similarly* $\mathbb{\Gamma}\backslash\delta$ *removes the TCV* δ *from the mapping* $\mathbb{\Gamma}$*.*
- *It is common to require the removal of all TCV from an LTC* $\mathbb{\Gamma}$ *written* $\mathbb{\Gamma}\backslash_{-TCV}$*, which maintains the order element of the LTC .*
- *The removal of TCVs to produce an STC is written* $\mathbb{\Gamma}\downarrow_{-TC}$ *and formally defined as follows.*

$$\begin{aligned} \emptyset\downarrow_{-TC} &\stackrel{\text{def}}{=} \emptyset \\ (\mathbb{\Gamma}+x:\alpha)\downarrow_{-TC} &\stackrel{\text{def}}{=} \mathbb{\Gamma}\downarrow_{-TC}, x:\alpha \\ (\mathbb{\Gamma}+\delta:\mathbb{TC})\downarrow_{-TC} &\stackrel{\text{def}}{=} \mathbb{\Gamma}\downarrow_{-TC} \end{aligned}$$

4.1.2 Standard Formulae

Formulae are constructed similarly to those of Sec. 2.2.2 with the standard equality $e = e'$, negation $\neg A$, conjunction $A \wedge B$ and evaluation formulae $e \bullet e' = m\{A\}$. Evaluation formulae internalise triples and express that if the program denoted by e is executed with an argument denoted by e', then the result, denoted by m, satisfies A. Since the ν-calculus is strongly normalizing, partial and total correctness are not distinguished. The shorthand notation from the λ-logic in Sec. 2.2.2, also applies here.

4.1.3 Restricted Universal Quantification

The meaning of $\forall x^{\alpha} \in (\mathbb{\Gamma}).A$ is intuitively simple: A must be true for all x that range only over values of type α, derived from $\mathbb{\Gamma}$. Deriving the values from $\mathbb{\Gamma}$ ensures hidden names in $\mathbb{\Gamma}$ are *not* revealed in x. This is achieved logically by ensuring that the restricting LTC types each expression quantified over. Logical instantiation of a restricted universal quantification must be an expression which is typed by the restricting LTC (or constructed out of the variables in the LTC), and hence unlike standard universal quantification which would quantify over all names (including those which are hidden), the restricted quantification cannot quantify over hidden names. The name **restricted** quantification comes from the fact that the quantification restricts the possible names and terms that the variable may obtain to those only derivable from the LTC.

Consider the example $(x : \mathsf{Nm}+y : \mathsf{Nm} \times \mathsf{Bool}+z : \mathsf{Nm}) \Vdash \forall p^{\mathsf{Nm}} \in (x+y).A$. Then it is required that p can be instantiated as x or as $\pi_1(y)$ but not as z or $\pi_2(y)$ due to the restricting LTC: $(x : \mathsf{Nm}+y : \mathsf{Nm} \times \mathsf{Bool})$ typing the former two expressions and not the latter.

As will be shown later, if the variables in an LTC $\mathbb{\Gamma}$, map to values and the name n only occurs in the mapping to $\lambda y.y = \mathsf{n}$ then n is hidden from these mappings and hence $\forall x^{\mathsf{Nm}} \in (\mathbb{\Gamma}).A$ cannot quantify over the name n at x. Formalising this requirement is subtle.

The quantification $\forall x^{\mathsf{Nm}} \in (\emptyset).A$ ensures that x is a fresh name, however this is the only way of expressing the introduction of a fresh name and thus this cannot be instantiated. Hence any formula of the form $\forall x^{\mathsf{Nm}} \in (\mathbb{\Gamma}).A$ quantifies over x being a fresh name and any value derivable from $\mathbb{\Gamma}$. The formula $\forall x^{\mathsf{Nm}} \in (\emptyset).A$ draws comparison to the nominal logic constructor "$И x.A$", as both ensure that x is a fresh name in A.

4.1.4 Quantification Over LTCs

The gensym function needs to be reasoned about such that each application of gensym in any state will produce a name which is fresh with respect to that future state, hence the requirement to name the state. The modal operator $\Box A$, is often used to expresses "for all future extensions", but fails to allow for the naming of the future state. The purpose of $\forall \delta.A$ is to name the future (or current) state using the TCV δ. This TCV can then be used in LTCs in A. The use of these TCVs must be in the LTCs in restricted quantifiers, which allows expressions to be derived from the future state.

Quantification over LTCs is similar to modal logic in the sense that it quantifies over

all future possible states which are now represented by the LTCs (through typing of the state), but differs from modal logic by assigning the future state to a variable TCV. The quantifier $\forall\delta.A$ is similar in essence to the hybrid logic formula "$\Box \downarrow_\delta A$" which quantifies over all future states (in time) using the $\Box$ modality and assigns that state to a variable δ using $\downarrow_\delta$ [9, 55]. The LTCs introduced here contain more information than the states used in hybrid logic, as they can be used to restrict quantification.

4.1.5 Notes On the Logical Syntax

In this logical language, names can never be referred to directly. Names can only be reasoned about through variables which represent those names. This is similar to how the ν_{GS}-calculus does not allow the programmer to state specific names, only specific variables that names are assigned to i.e. in $\mathsf{let}\ x = \mathsf{gensym}()\ \mathsf{in}\ M$ the fresh name cannot be referred to directly, instead the variable x is used to refer to the name.

As a motivation for LTCs, consider the example program $\mathsf{let}\ y = \mathsf{gensym}()\ \mathsf{in}\ \lambda x^{\mathsf{Nm}}.x = y$. The name assigned to the variable y will be fresh and can only ever be used in the returned function in the comparison to x. For example, every future possible application of the resulting function from $(\emptyset,\ \mathsf{let}\ y = \mathsf{gensym}()\ \mathsf{in}\ \lambda x^{\mathsf{Nm}}.x = y) \Downarrow (\{\mathsf{n}\},\ \lambda x^{\mathsf{Nm}}.x = \mathsf{n})$ will never be applied to n itself as it is in a sense "lost" or "hidden" under the equality and λ-binder. Hence the unrestricted universal quantification, if used naively, cannot suffice to reason about the ν_{GS}-calculus as the $[\textsc{Lam}]_\lambda$ -rule from Fig. 2.20 (seen below). fails to hold. This is because the post-condition $\{\forall x^\alpha.(B \to u \bullet x = m\{C\})\}$ would include quantification over all names including hidden names, such as n.

$$\frac{\{A^{-x} \wedge B\}\ M :_m \{C\}}{\{A\}\ \lambda x^\alpha.M :_u \{\forall x^\alpha.(B \to u \bullet x = m\{C\})\}}\ [\textsc{Lam}]_\lambda$$

Hence the restriction placed on quantifiers in this logic to allow us to reason about which values can be quantified over by introducing LTCs as a list of variables which can be used to derive a value.

It may be possible that a more abstract or complex notion of LTCs could be used with a more mathematically based set of operations to manipulate them, however this was not required in this case.

4.1.6 Shorthand Notations

The restricted existential quantification is defined using the standard definition as follows.

$$\exists x^\alpha \in (\mathbb{\Gamma}).A \stackrel{def}{=} \neg\forall x^\alpha \in (\mathbb{\Gamma}).\neg A$$

This reads as: there exists a value x derived from the LTC $\mathbb{\Gamma}$, such that A holds. This behaves similarly to the unrestricted existential quantification.

Fresh names are produced by gensym but freshness is not an absolute notion, instead a name is fresh with respect to something. In the case of programs a fresh name produced by gensym() is fresh from the nameset in the configuration (see Fig. 3.4). However, in the logic, freshness is with respect to an LTC (or the names derivable from an LTC). Thus freshness of the name x relative to the LTC $\mathbb{\Gamma}$ is written $e\#\mathbb{\Gamma}$ and defined below. Freshness is used pervasively, hence this shorthand notation.

$$e\#\mathbb{\Gamma} \stackrel{def}{=} \forall z^{\mathsf{Nm}} \in (\mathbb{\Gamma}).e \neq z.$$

Intuitively, $e\#\mathbb{\Gamma}$ states that the name denoted by e is not derivable, directly or indirectly, from the LTC $\mathbb{\Gamma}$. Freshness is a variant of a similar predicate in [72] seen in Sec. 2.2.3, which only states freshness from another expression i.e. $e\#e'$.

The typing in LTCs is often dropped for brevity such that where α is obvious, $\mathbb{\Gamma}+y:\alpha$ becomes $\mathbb{\Gamma}+y$. A further reduction in notation writes $\mathbb{\Gamma}+\delta+\mathbb{\Gamma}'$ as just $\delta+\mathbb{\Gamma}'$ because δ represents an LTC which is an extension of $\mathbb{\Gamma}$ and thus will contain all mappings in $\mathbb{\Gamma}$, but δ does not include any mapping in $\mathbb{\Gamma}'$ as $\mathbb{\Gamma}'$ is an extension of δ. If types are obvious in formulae they will often be dropped i.e. $e \bullet e' = m^{\alpha}\{A\} \stackrel{def}{=} e \bullet e' = m\{A\}$ and $\forall x^{\alpha} \in (\mathbb{\Gamma}).A \stackrel{def}{=} \forall x \in (\mathbb{\Gamma}).A$.

4.1.7 Triples

Triples are introduced for the ν-logic as they are for the λ-logic and Local-logic. The triple $\{A\}\ M :_u \{B\}$ has the standard meaning: if the *pre-condition* A holds and the value derived from program M is denoted by the *anchor* u, then the *post-condition* B holds. In this case the program M must now be static syntax, meaning the program contains no names ($\mathfrak{a}(M) = \emptyset$), and hence can only reference names via gensym. If names are included in the programs then they would also be required in the logic, which requires a more complicated logic.

4.2 Typing of Expressions, Formulae and Triples

New type judgments are introduced to type expressions, LTCs, formulae and triples using the LTCs as a basis and written $\mathbb{\Gamma} \Vdash e : \alpha$, $\mathbb{\Gamma} \Vdash \mathbb{\Gamma}'$, $\mathbb{\Gamma} \Vdash A$ and $\mathbb{\Gamma} \Vdash \{A\}\ M :_u \{B\}$ respectively. The LTC doing the typing (i.e. $\mathbb{\Gamma}$ in $\mathbb{\Gamma} \Vdash$) is referred to as the global LTC.

LTCs are required to be typed by a global LTC, to ensure all LTCs used in formulae (i.e. $\forall x \in (\mathbb{\Gamma}).A$ and $x\#\mathbb{\Gamma}$), are ordered subsets of the global LTC.

Definition 42 (Typing of LTCs, expressions, formulae)**.** *The rules for typing expressions, LTC and formulae can be seen in Fig. 4.2. In all cases the LTC doing the typing (in these cases $\mathbb{\Gamma}$) is referred to as the* global LTC.

Expressions:

$$\frac{b \in \{\mathsf{true}, \mathsf{false}\}}{\mathbb{\Gamma} \Vdash b : \mathsf{Bool}} \qquad \frac{-}{\mathbb{\Gamma} \Vdash () : \mathsf{Unit}} \qquad \frac{\mathbb{\Gamma}(x) = \alpha}{\mathbb{\Gamma} \Vdash x : \alpha}$$

$$\frac{\mathbb{\Gamma} \Vdash e : \alpha_1 \quad \mathbb{\Gamma} \Vdash e' : \alpha_2}{\mathbb{\Gamma} \Vdash \langle e, e' \rangle : \alpha_1 \times \alpha_2} \qquad \frac{\mathbb{\Gamma} \Vdash e : \alpha_1 \times \alpha_2}{\mathbb{\Gamma} \Vdash \pi_i(e) : \alpha_i}$$

LTCs:

$$\frac{-}{\mathbb{\Gamma} \Vdash \emptyset} \qquad \frac{\mathbb{\Gamma} \Vdash \mathbb{\Gamma}_0}{\mathbb{\Gamma}{+}x : \alpha \Vdash \mathbb{\Gamma}_0{+}x : \alpha} \qquad \frac{\mathbb{\Gamma} \Vdash \mathbb{\Gamma}_0}{\mathbb{\Gamma}{+}\delta \Vdash \mathbb{\Gamma}_0{+}\delta} \qquad \frac{\mathbb{\Gamma} \Vdash \mathbb{\Gamma}_0}{\mathbb{\Gamma}{+}\mathbb{\Gamma}' \Vdash \mathbb{\Gamma}_0}$$

Formulae:

$$\frac{\mathbb{\Gamma} \Vdash e_1 : \alpha \quad \mathbb{\Gamma} \Vdash e_2 : \alpha}{\mathbb{\Gamma} \Vdash e_1 = e_2} \qquad \frac{\mathbb{\Gamma} \Vdash A_1 \quad \mathbb{\Gamma} \Vdash A_2}{\mathbb{\Gamma} \Vdash A_1 \wedge A_2} \qquad \frac{\mathbb{\Gamma} \Vdash A}{\mathbb{\Gamma} \Vdash \neg A}$$

$$\frac{\mathbb{\Gamma} \Vdash e : \alpha_1 \to \alpha_2 \quad \mathbb{\Gamma} \Vdash e' : \alpha_1 \quad \mathbb{\Gamma}{+}x : \alpha_2 \Vdash A}{\mathbb{\Gamma} \Vdash e \bullet e' = x^{\alpha_2}\{A\}}$$

$$\frac{\mathbb{\Gamma} \Vdash \mathbb{\Gamma}' \quad \mathbb{\Gamma}{+}x : \alpha \Vdash A}{\mathbb{\Gamma} \Vdash \forall x^{\alpha} \in (\mathbb{\Gamma}').A} \qquad \frac{\mathbb{\Gamma} \Vdash e : \mathsf{Nm} \quad \mathbb{\Gamma} \Vdash \mathbb{\Gamma}'}{\mathbb{\Gamma} \Vdash e\#\mathbb{\Gamma}'} \qquad \frac{\mathbb{\Gamma}{+}\delta : \mathbb{TC} \Vdash A}{\mathbb{\Gamma} \Vdash \forall \delta.A}$$

Triples:

$$\frac{\mathbb{\Gamma} \Vdash A \quad \mathbb{\Gamma} \downarrow_{-TC} \vdash M : \alpha \quad \mathbb{\Gamma}{+}m : \alpha \Vdash B}{\mathbb{\Gamma} \Vdash \{A\}\ M :_m \{B\}}$$

Figure 4.2: Typing rules for expressions, LTCs, formulae and triples in the ν-logic. In the last rule for triples, M is static syntax.

The primary novelties are the typing rules for restricted quantification and quantification over LTCs. Each addition to the global LTC $\mathbb{\Gamma}$ must add the variable or TCV as an extension to the global LTC $\mathbb{\Gamma}$, ensuring the order is maintained. This extended LTC then types the sub-formula which may now use the variables or TCV in LTCs with the

assurance they occur in the correct order. For example the following type check fails if $\mathbb{\Gamma}_0 \equiv \mathbb{\Gamma}+x+\delta$, as this implies δ is an extension of $\mathbb{\Gamma}+x$, which is not the case as x is derived from δ. Hence the type check holds if $\mathbb{\Gamma}_0 \equiv \mathbb{\Gamma}+\delta+x$ or any LTC typed by this LTC (referred to as sub-LTCs).

$$\mathbb{\Gamma} \Vdash \forall\delta.\forall x \in (\delta).\forall y \in (\mathbb{\Gamma}_0).A$$

Definition 43 (Typing of triples)**.** *Typing rules for expressions, and LTC are used in the typing rules for formulae which in turn are used in the typing rules for triples written* $\mathbb{\Gamma} \Vdash \{A\}\ M :_u \{B\}$*. Programs are typed as in Fig. 3.2 which require an STC to type the programs, hence the use of* $\mathbb{\Gamma}\downarrow_{-TC}$*, which removes TCV and converts the LTC to an STC, in the typing rule for triples in Fig. 4.2.*

From now on it is assumed *all LTCs, expressions, formulae and triples are well-typed*, and typing will mostly be omitted unless explicitly required.

4.3 Advanced Substitutions

Reasoning with quantifiers requires quantifier instantiation. This is subtle with the two new logical quantifiers. Substitution of an expression e for a variable x in a formula A requires a global LTC $\mathbb{\Gamma}$ to ensure that substitutions of formulae with LTCs (which contain x) are well typed by the global LTC, this substitution is written $A[e/x]_{\mathbb{\Gamma}}$. Quantification over LTCs requires the substitution of an LTC $\mathbb{\Gamma}_0$ for TCV δ in a formula A written $A[\mathbb{\Gamma}_0/\delta]_{\mathbb{\Gamma}}$ where $\mathbb{\Gamma}$ is the global LTC required to maintain the order of any substitution. If the global LTC is obvious or not required then it is often dropped i.e. $[e/x]_{\mathbb{\Gamma}} \stackrel{def}{=} [e/x]$.

The free variables and TCV of expressions, LTCs and formulae are first defined as follows.

Definition 44 (Free variables of expressions)**.** *The free variables of expression* e*, written* $\mathsf{fv}(e)$*, are simply the variables occurring in the expression.*

$$\begin{array}{llll} \mathsf{fv}(()) \stackrel{\text{def}}{=} \emptyset & & \mathsf{fv}(x) \stackrel{\text{def}}{=} \{x\} & \\ \mathsf{fv}(\mathsf{true}) \stackrel{\text{def}}{=} \emptyset & & \mathsf{fv}(\langle e, e'\rangle) \stackrel{\text{def}}{=} \mathsf{fv}(e) \cup \mathsf{fv}(e') & \\ \mathsf{fv}(\mathsf{false}) \stackrel{\text{def}}{=} \emptyset & & \mathsf{fv}(\pi_i(e)) \stackrel{\text{def}}{=} \mathsf{fv}(e) & \end{array}$$

Definition 45 (Free variables of LTCs)**.** *Free variables of an LTC* $\mathbb{\Gamma}$ *written* $\mathsf{fv}(\mathbb{\Gamma})$*, are defined as all the variables of the domain i.e.* $\mathsf{fv}(\mathbb{\Gamma}) \stackrel{\text{def}}{=} \mathsf{dom}(\mathbb{\Gamma}\downarrow_{-TC}) \stackrel{\text{def}}{=} \mathsf{dom}(\mathbb{\Gamma}\backslash_{-TCV})$*. For brevity, if* $x \notin \mathsf{fv}(\mathbb{\Gamma})$ *then write* $\mathbb{\Gamma}^{\text{-}x}$*.*

Definition 46 (Free variables of Formulae)**.** *The free variables of formulae are defined as those occurring unbound by the anchors in evaluation formulae or quantifiers, with the formal definition as follows.*

$$\begin{array}{llll}
\mathsf{fv}(e = e') & \stackrel{\text{def}}{=} & \mathsf{fv}(e) \cup \mathsf{fv}(e') & \mathsf{fv}(e \bullet e' = m\{A\}) \stackrel{\text{def}}{=} \mathsf{fv}(e) \cup \mathsf{fv}(e') \cup (\mathsf{fv}(A) \setminus \{m\}) \\
\mathsf{fv}(\neg A) & \stackrel{\text{def}}{=} & \mathsf{fv}(A) & \mathsf{fv}(\forall x \in (\mathbb{\Gamma}).A) \stackrel{\text{def}}{=} (\mathsf{fv}(A) \backslash \{x\}) \cup \mathsf{fv}(\mathbb{\Gamma}) \\
\mathsf{fv}(A \wedge B) & \stackrel{\text{def}}{=} & \mathsf{fv}(A) \cup \mathsf{fv}(B) & \mathsf{fv}(x \# \mathbb{\Gamma}) \stackrel{\text{def}}{=} \mathsf{fv}(\mathbb{\Gamma}) \cup \{x\} \\
\mathsf{fv}(\forall \delta.A) & \stackrel{\text{def}}{=} & \mathsf{fv}(A) &
\end{array}$$

Similar to free variables, a function on LTC and formulae are defined on the free TCVs as follows.

Definition 47 (Free TCVs of LTCs)**.** *The free TCVs of an LTC* $\mathbb{\Gamma}$ *written* $\mathsf{ftcv}(\mathbb{\Gamma})$*, are simply the TCVs occurring in the LTC as follows.*

$$\begin{array}{lll}
\mathsf{ftcv}(\emptyset) & \stackrel{\text{def}}{=} & \emptyset \\
\mathsf{ftcv}(\mathbb{\Gamma} + x : \alpha) & \stackrel{\text{def}}{=} & \mathsf{ftcv}(\mathbb{\Gamma}) \\
\mathsf{ftcv}(\mathbb{\Gamma} + \delta : \mathbb{TC}) & \stackrel{\text{def}}{=} & \mathsf{ftcv}(\mathbb{\Gamma}) \cup \{\delta\} \\
\mathsf{ftcv}(\mathbb{\Gamma} + \mathbb{\Gamma}') & \stackrel{\text{def}}{=} & \mathsf{ftcv}(\mathbb{\Gamma}) \cup \mathsf{ftcv}(\mathbb{\Gamma}')
\end{array}$$

An LTC $\mathbb{\Gamma}$ *is* closed *or* TCV-free *if* $\mathsf{ftcv}(\mathbb{\Gamma}) = \emptyset$.

The same function is used to define free TCV of formulae.

Definition 48 (Free TCVs of formulae)**.** *The free TCVs of formulae are those occurring unbound by a quantifier over LTCs with the formal definition as follows.*

$$\begin{array}{lll}
\mathsf{ftcv}(e = e') & \stackrel{\text{def}}{=} & \emptyset \\
\mathsf{ftcv}(\neg A) & \stackrel{\text{def}}{=} & \mathsf{ftcv}(A) \\
\mathsf{ftcv}(A \wedge B) & \stackrel{\text{def}}{=} & \mathsf{ftcv}(A) \cup \mathsf{ftcv}(B) \\
\mathsf{ftcv}(e \bullet e' = m\{A\}) & \stackrel{\text{def}}{=} & \mathsf{ftcv}(A) \\
\mathsf{ftcv}(x \# \mathbb{\Gamma}) & \stackrel{\text{def}}{=} & \mathsf{ftcv}(\mathbb{\Gamma}) \\
\mathsf{ftcv}(\forall x \in (\mathbb{\Gamma}).A) & \stackrel{\text{def}}{=} & \mathsf{ftcv}(A) \cup \mathsf{ftcv}(\mathbb{\Gamma}) \\
\mathsf{ftcv}(\forall \delta.A) & \stackrel{\text{def}}{=} & \mathsf{ftcv}(A) \setminus \{\delta\}
\end{array}$$

A formula A *is* TCV-free *if* $\mathsf{ftcv}(A) = \emptyset$.

Before substitutions are defined, it is necessary to define when substitutions can occur safely. This extends the check in [38], which check that no variable being introduced is bound by a binder (assuming the typing holds) as follows.

Definition 49 (Expressions free for variables in formula)**.**

Define an expression e-free for x^{α} in A *in some LTC* $\mathbb{\Gamma}$ *if the following hold.*

– *e is of type* α *(*$\mathbb{\Gamma} \Vdash e : \alpha$*)*

– *For all occurrences of* x *in* A *all variables in* $\mathsf{fv}(e)$ *do* not *become bound in* A*.*

– *If* e *contains destructors i.e.* $\pi_i(\ )$ *or* $=$*, then all free occurrences of* x *occurring in any LTC (say* $\mathbb{\Gamma}_0$*) itself occurring in the formula* A *must imply* $\mathbb{\Gamma}_0 \Vdash e : \alpha$*.*

The extension to [38] is the final case above, which is introduced to ensure no complications arise from substituting the expressions containing $\pi_i(\ )$ or $=$ into terms with LTCs in restricted quantification.

Consider the substitution of the expression $\pi_1(y)$ for some variable x in a formula, such that the x occurs in an LTC $\mathbb{\Gamma}_0$ which occurs in the formula. If y occurs in $\mathbb{\Gamma}_0$ then no new variables are introduced by this substitution. However, if y does not occur in $\mathbb{\Gamma}_0$ then via the construction of the logic, substituting in the expression $\pi_1(y)$ is not permitted as it is not a variable, but substituting in the variable y may introduce the other side of y i.e. $\pi_2(y)$ to the LTC, which extends the LTC beyond its intended reach and has the potential to introduce previously inaccessible names. Hence the introduction of the definition above to restrict these forms of substitutions.

Similarly, consider the substitution of the expression $y = z$ for some variable x in the formula $\forall p \in (\mathbb{\Gamma}_0).A$, such that the x occurs in an LTC $\mathbb{\Gamma}_0$. Then two options for the substitution exist. Either y and z both already occur in $\mathbb{\Gamma}_0$ or not. In the former case then the substitution results in the same formula. In the latter case this cannot be a valid substitution as the expression $y = z$ cannot be introduced into the LTC $\mathbb{\Gamma}_0$ due to the construction of LTCs. However, adding both y and z to $\mathbb{\Gamma}_0$ then allows for names derivable from these variables to be quantified over for p but these same names are not accessible from the expression $y = z$. This substitution however is not permitted as $y = z$ is not free for x in $\forall p \in (\mathbb{\Gamma}_0).A$ as $\mathbb{\Gamma}_0 \nvdash y = z$ hence the justification for the new case is introduced in Def. 49.

The expressions of the form $\pi_i(e)$ and $e = e'$ are defined as destructors, as they take larger typed expressions e and e' and produce a smaller typed expression either α_i or Bool which means information (primarily about names) is lost. If further extensions to the programming language are considered the list of destructors may need to be altered.

Definition 50 (Logical substitution of expressions in expressions)**.** *The* logical substitution $e[e'/x]$*, of expression* e' *for variable* x *in expression* e*, is inductively defined on* e *as*

follows which has no restrictions, and no global LTC to type the substitution.

$$
\begin{array}{rcl@{\qquad}rcl}
()[e/x] & \stackrel{\text{def}}{=} & () & x[e/x] & \stackrel{\text{def}}{=} & e \\
\mathsf{true}[e/x] & \stackrel{\text{def}}{=} & \mathsf{true} & y[e/x] & \stackrel{\text{def}}{=} & y \\
\mathsf{false}[e/x] & \stackrel{\text{def}}{=} & \mathsf{false} & \langle e_1, e_2\rangle[e/x] & \stackrel{\text{def}}{=} & \langle e_1[e/x], e_2[e/x]\rangle \\
 & & & \pi_i(e')[e/x] & \stackrel{\text{def}}{=} & \pi_i(e'[e/x])
\end{array}
$$

Definition 51 (Logical substitution of expressions in formulae)**.** Logical substitution of e for x in A in the context of $\mathbb{\Gamma}$, *written* $A[e/x]_{\mathbb{\Gamma}}$*, assumes e-free for* x *in* A *, and is defined as follows. The first line introduces* logical substitution of e for x in $\mathbb{\Gamma}'$ in context of $\mathbb{\Gamma}$.

$$
\mathbb{\Gamma}'[e/x]_{\mathbb{\Gamma}} \stackrel{\text{def}}{=} \begin{cases} \mathbb{\Gamma}'_e \ s.t.\ \mathsf{dom}(\mathbb{\Gamma}'_e) = \mathsf{fv}(e) \cup \mathsf{dom}(\mathbb{\Gamma}'\backslash x), \mathbb{\Gamma} \Vdash \mathbb{\Gamma}'_e & x \in \mathsf{fv}(\mathbb{\Gamma}') \\ \mathbb{\Gamma}' & x \notin \mathsf{fv}(\mathbb{\Gamma}') \end{cases}
$$

$$
\begin{array}{rcll}
\mathsf{T}[e/x]_{\mathbb{\Gamma}} & \stackrel{\text{def}}{=} & \mathsf{T} & \\
\mathsf{F}[e/x]_{\mathbb{\Gamma}} & \stackrel{\text{def}}{=} & \mathsf{F} & \\
e_1 = e_2[e/x]_{\mathbb{\Gamma}} & \stackrel{\text{def}}{=} & e_1[e/x] = e_2[e/x] & \\
(\neg A)[e/x]_{\mathbb{\Gamma}} & \stackrel{\text{def}}{=} & \neg(A[e/x]_{\mathbb{\Gamma}}) & \\
(A \wedge B)[e/x]_{\mathbb{\Gamma}} & \stackrel{\text{def}}{=} & (A[e/x]_{\mathbb{\Gamma}}) \wedge (B[e/x]_{\mathbb{\Gamma}}) & \\
(e_1 \bullet e_2 = m\{A\})[e/x]_{\mathbb{\Gamma}} & \stackrel{\text{def}}{=} & e_1[e/x] \bullet e_2[e/x] = m\{A[e/x]_{\mathbb{\Gamma}+m}\} & (x \neq m,\ m \notin \mathsf{fv}(\mathbb{\Gamma})) \\
(e'\#\mathbb{\Gamma}')[e/x]_{\mathbb{\Gamma}} & \stackrel{\text{def}}{=} & e'[e/x]\#(\mathbb{\Gamma}'[e/x]_{\mathbb{\Gamma}}) & \\
(\forall m \in (\mathbb{\Gamma}').A)[e/x]_{\mathbb{\Gamma}} & \stackrel{\text{def}}{=} & \forall m \in (\mathbb{\Gamma}'[e/x]_{\mathbb{\Gamma}}).(A[e/x]_{\mathbb{\Gamma}+m}) & (x \neq m,\ m \notin \mathsf{fv}(\mathbb{\Gamma})) \\
(\forall \delta.A)[e/x]_{\mathbb{\Gamma}} & \stackrel{\text{def}}{=} & \forall \delta.(A[e/x]_{\mathbb{\Gamma}+\delta}) &
\end{array}
$$

For $\mathbb{\Gamma}'[e/x]_{\mathbb{\Gamma}}$, Def. 49 guarantees that if x is in $\mathbb{\Gamma}$ and e contains a destructor such as $\pi_i(\cdot)$ or $=$ then every free variable in e must occur in LTC $\mathbb{\Gamma}'$. So this substitution returns $\mathbb{\Gamma}'\backslash x$ as $\mathsf{fv}(e) \subseteq \mathsf{fv}(\mathbb{\Gamma}'\backslash x)$ is implied by the assumption. However, if e is destructor-free and x occurs in $\mathbb{\Gamma}'$ then a new LTC is returned, consisting of the original LTC $\mathbb{\Gamma}'$ with x removed and the addition of all free variables in e mapped to their respective types defined and ordered by $\mathbb{\Gamma}$. Clearly if x does not occur in $\mathbb{\Gamma}'$ then no substitution is required.

Cases which use LTCs i.e. freshness and restricted quantification both require substitution of expressions in LTCs as described above, whilst all other substitutions are standard.

The definition of *type context substitution* of LTCs for TCVs in both LTCs and formulae are introduced in Def. 53, but first requires first defining when an LTC is free for an LTC in a formula similar to Def. 49.

Definition 52 (LTCs free for TCVs in formula)**.** *The LTC* $\mathbb{\Gamma}_0$ is free for δ *in* A *in a global LTC* $\mathbb{\Gamma}$ *if no free occurrence of any TCV in* $\mathbb{\Gamma}_0$ *becomes a bound occurrence of the TCV if* $\mathbb{\Gamma}_0$ *is substituted for* δ *in* A*. This requires* $\mathbb{\Gamma}_0$ *to be a sub-LTC of* $\mathbb{\Gamma}$*.*

Definition 53 (Type context substitutions)**.** *The substitution of* $\mathbb{\Gamma}_0$ *for* δ *in* $\mathbb{\Gamma}'$ *in the global LTC* $\mathbb{\Gamma}$ *requires that* $\mathbb{\Gamma}_0$ *is free for* δ *in* $\mathbb{\Gamma}'$*. This requires* $\mathbb{\Gamma}_0$ *and* $\mathbb{\Gamma}'$ *to be sub-LTCs of* $\mathbb{\Gamma}$*. The LTC substitutions are as expected where TCVs can only appear in freshness and restricted quantification formulae, and are defined inductively for the other logical constructors as expected. The first line defines the substitution on LTCs.*

$$\mathbb{\Gamma}'[\mathbb{\Gamma}_0/\delta]_{\mathbb{\Gamma}} \stackrel{\text{def}}{=} \begin{cases} \mathbb{\Gamma}_1 \ s.t. \ \mathsf{dom}(\mathbb{\Gamma}_1) = \mathsf{dom}(\mathbb{\Gamma}_0, \mathbb{\Gamma}'\backslash\delta), \mathbb{\Gamma} \Vdash \mathbb{\Gamma}_1 & \delta \in \mathsf{ftcv}(\mathbb{\Gamma}') \\ \mathbb{\Gamma}' & \delta \notin \mathsf{ftcv}(\mathbb{\Gamma}') \end{cases}$$

$$\begin{array}{rcll}
\mathsf{T}[\mathbb{\Gamma}_0/\delta]_{\mathbb{\Gamma}} & \stackrel{\text{def}}{=} & \mathsf{T} & \\
\mathsf{F}[\mathbb{\Gamma}_0/\delta]_{\mathbb{\Gamma}} & \stackrel{\text{def}}{=} & \mathsf{F} & \\
e = e'[\mathbb{\Gamma}_0/\delta]_{\mathbb{\Gamma}} & \stackrel{\text{def}}{=} & e = e' & \\
(\neg A)[\mathbb{\Gamma}_0/\delta]_{\mathbb{\Gamma}} & \stackrel{\text{def}}{=} & \neg(A[\mathbb{\Gamma}_0/\delta]_{\mathbb{\Gamma}}) & \\
(A \wedge B)[\mathbb{\Gamma}_0/\delta]_{\mathbb{\Gamma}} & \stackrel{\text{def}}{=} & A[\mathbb{\Gamma}_0/\delta]_{\mathbb{\Gamma}} \wedge B[\mathbb{\Gamma}_0/\delta]_{\mathbb{\Gamma}} & \\
(e_1 \bullet e_2 = m\{A\})[\mathbb{\Gamma}_0/\delta]_{\mathbb{\Gamma}} & \stackrel{\text{def}}{=} & e_1 \bullet e_2 = m\{A[\mathbb{\Gamma}_0/\delta]_{\mathbb{\Gamma}+m}\} & (m \notin \mathsf{fv}(\mathbb{\Gamma}_0)) \\
(e\#\mathbb{\Gamma}')[\mathbb{\Gamma}_0/\delta]_{\mathbb{\Gamma}} & \stackrel{\text{def}}{=} & e\#(\mathbb{\Gamma}'[\mathbb{\Gamma}_0/\delta]_{\mathbb{\Gamma}}) & \\
(\forall m \in (\mathbb{\Gamma}').A)[\mathbb{\Gamma}_0/\delta]_{\mathbb{\Gamma}} & \stackrel{\text{def}}{=} & \forall m \in (\mathbb{\Gamma}'[\mathbb{\Gamma}_0/\delta]_{\mathbb{\Gamma}}).(A[\mathbb{\Gamma}_0/\delta]_{\mathbb{\Gamma}+x}) & (m \notin \mathsf{fv}(\mathbb{\Gamma}_0)) \\
(\forall \delta'.A)[\mathbb{\Gamma}_0/\delta]_{\mathbb{\Gamma}} & \stackrel{\text{def}}{=} & \begin{cases} (\forall \delta'.A[\mathbb{\Gamma}_0/\delta]_{\mathbb{\Gamma}+\delta'}) & \delta \neq \delta' \quad (\delta' \notin \mathsf{ftcv}(\mathbb{\Gamma}_0)) \\ \forall \delta.A & \textit{otherwise} \end{cases} &
\end{array}$$

In $\mathbb{\Gamma}'[\mathbb{\Gamma}_0/\delta]_{\mathbb{\Gamma}}$ if δ occurs in $\mathbb{\Gamma}'$ then the result is the union of $\mathbb{\Gamma}'\backslash\delta$ and $\mathbb{\Gamma}_0$ with the order defined by $\mathbb{\Gamma}$. If δ does not occur in $\mathbb{\Gamma}'$ then that LTC is unaffected by the change in δ and hence the same LTC is returned. The $\forall\delta.A$ case requires the cases to guarantee capture avoiding substitution for LTCs. If $\delta = \delta'$, then δ will never occur free in A, hence can never be substituted for. Otherwise, the substitution occurs as expected with the condition that δ' does not occur in $\mathbb{\Gamma}_0$, meaning this is a capture avoiding substitution for type contexts.

In both substitutions, the evaluation formulae and the two new quantifiers (i.e. freshness and restricted quantification) require the addition of a binding variable/TCV to the LTC $\mathbb{\Gamma}$ to ensure the correct order is maintained in subsequent substitutions i.e. $A[e/x]_{\mathbb{\Gamma}+m}$ and $A[e/x]_{\mathbb{\Gamma}+\delta}$. These additions to the LTC unsurprisingly match the addi-

tions in the typing rules seen in Fig. 3.2. The global LTC is often dropped for brevity i.e. $A[e/x]_{\mathbb{\Gamma}} \stackrel{def}{=} A[e/x]$ and $[\mathbb{\Gamma}_0/\delta]_{\mathbb{\Gamma}} \stackrel{def}{=} [\mathbb{\Gamma}_0/\delta]$, as the LTC $\mathbb{\Gamma}$ is only used for ordering and types, which can be obtained from the global LTC that types the triple/formula.

4.4 Properties of Logical Formulae

In first-order logic, if a formula is satisfied by a model, then it is also satisfied by extensions of that model, and vice-versa (assuming all free variables of the formula remain in the model). This can no longer be taken for granted in this logic.

Consider the formula $\forall\delta.\exists z \in (\delta).(z\#\mathbb{\Gamma} \wedge \neg z\#\delta)$ typed by $\mathbb{\Gamma}_0 \equiv \mathbb{\Gamma}+\mathbb{\Gamma}'_0$. This states that there exists a name z which is fresh from $\mathbb{\Gamma}$ but not fresh from any extension i.e. δ, essentially meaning there is a reachable name in the $\mathbb{\Gamma}'_0$ part which is not in $\mathbb{\Gamma}$. The quantification over LTCs requires this must hold for δ representing the current state $\mathbb{\Gamma}_0$ (and all its extensions). Hence if $\mathbb{\Gamma}'_0$ has a new variable mapped to a fresh name then this holds, however $\mathbb{\Gamma}'_0 \equiv \emptyset$ is a clear contradiction as no name exists in $\mathbb{\Gamma}'_0$. Hence the formula may hold for $\mathbb{\Gamma}'_0 \not\equiv \emptyset$, but fail if $\mathbb{\Gamma}'_0 \equiv \emptyset$. The negation of the formula fails for LTC extensions and this shows how some formulae may become invalid under contracting or extension of the LTC in which it is typed.

Fortunately, such formulae are rarely needed when reasoning about programs. However, in order to satisfy the soundness proofs, a restriction on formulae referred to as *extension independence* is introduced, ensuring the formulae are stable under LTC extension and contractions. This is first introduced as a syntactic definition however the formal definition is introduced later using models.

Sometimes a weaker property is required to ensure formulae preserve their validity when a specific variable is removed from the general typing LTC. The example $\mathbb{\Gamma}+x : \mathsf{Nm}+\mathbb{\Gamma}_0 \Vdash \forall\delta.\exists z \in (\delta).(z\#\mathbb{\Gamma} \wedge \neg z\#\delta)$ is an example where the x is required if $\mathbb{\Gamma}_0$ is all Nm-free for the same reason as above. This property is referred to as *thinness* with respect to a variable.

Syntactic definitions of sets of formulae which obtain these respective properties are introduced as follows.

Definition 54 (Syntactic classification of extension independent formulae)**.** *Define a set* $\textsc{ExtInd}_{Syn}$*, of formulae that are syntactically extension independent, using the following set of inductive rules. A formula A is defined as* $\textsc{ExtInd}_{Syn}$*, written A-*$\textsc{ExtInd}_{Syn}$*, if it is in* $\textsc{ExtInd}_{Syn}$*.*

1. T, F, $e = e'$ *are all* $\textsc{ExtInd}_{Syn}$.

2. *If* A_1-$\textsc{ExtInd}_{Syn}$ *and* A_2-$\textsc{ExtInd}_{Syn}$
   then $\neg A_1$, $A_1 \wedge A_2$, $A_1 \vee A_2$, $A_1 \rightarrow A_2$, $u \bullet e = m^{\alpha}\{A_1\}$, $\exists x \in (\mathbb{\Gamma}').A_1$, $\forall x \in (\mathbb{\Gamma}').A_1$
   are all in $\textsc{ExtInd}_{Syn}$.

3. *If* A_1-$\textsc{ExtInd}_{Syn}$ *and* $\delta \notin \mathsf{ftcv}(A_1)$
   then $\forall \delta.A_1$-$\textsc{ExtInd}_{Syn}$
   and $x \notin \mathsf{fv}(A_1)$ *implies* $\forall \delta. \forall x \in (\delta).A_1$-$\textsc{ExtInd}_{Syn}$

4. *Two extra specific cases are:*
   $\forall \delta. f \bullet () = b\{b\#\delta\}$-$\textsc{ExtInd}_{Syn}$
   and $\forall \delta. \forall x \in (\delta). f \bullet x = b\{b\#\delta + x\}$-$\textsc{ExtInd}_{Syn}$

This can be used to show $e\#\mathbb{\Gamma}'(\stackrel{def}{=} \forall z \in (\mathbb{\Gamma}').z \neq e)$-$\textsc{ExtInd}_{Syn}$ as follows:
(1.) implies $z = e'$-$\textsc{ExtInd}_{Syn}$
(2.) and the above implies $\neg z = e'$-$\textsc{ExtInd}_{Syn}$
(3.) and the above line implies $\forall z \in (\mathbb{\Gamma}').\neg z = e'$-$\textsc{ExtInd}_{Syn}$

This is an incomplete characterisation of all extension independent formulae, but provides a simple method for checking whether a formula is $\textsc{ExtInd}_{Syn}$ and hence this will later be shown to prove it is also $\textsc{ExtInd}_{Sem}$. Given the incomplete characterisation, this cannot be used to prove non-extension independence. Case 3. could be generalised further but was not required in any of the reasoning examples.

The $\textsc{ExtInd}_{Syn}$ definition is similar to the syntactic definition for monotonicity in the Local-logic in [72], however the meaning is more similar to statelessness from the same paper. Statelessness implies $A \equiv \Box A$ which only ever looks forward however $\textsc{ExtInd}_{Syn}$ requires a more complex version of $A \equiv \Box A$ to look at previous states.

In Sec. 5.3 extension independent formulae are given a semantic formalisation, and a proof that $\textsc{ExtInd}_{Syn}$ formulae are indeed extension independent is given.

Definition 55 (Syntactic classification of thin formulae)**.** *Define a set* $\textsc{Thin}_{Syn}(x)$*, of formulae that are syntactically thin with respect to the variable* x*, using the following set of inductive rules. A formula* A *is defined as* $\textsc{Thin}_{Syn}(x)$*, written* A-$\textsc{Thin}_{Syn}(x)$*, if it is in* $\textsc{Thin}_{Syn}(x)$.

1. *If* $\mathbb{\Gamma} \Vdash A$ *and* $\mathbb{\Gamma} \Vdash x : \alpha_{\_(\mathsf{Nm},\rightarrow)}$ *then* A-$\textsc{Thin}_{Syn}(x)$

2. *If* $A \equiv \mathsf{T}, \mathsf{F}, e = e'$, $e \neq e'$ *and* $x \notin \mathsf{fv}(A)$ *then* A-$\textsc{Thin}_{Syn}(x)$

3. *If* A_1-$\text{THIN}_{Syn}(x)$ *and* A_2-$\text{THIN}_{Syn}(x)$ *then* $A_1 \wedge A_2$, $A_1 \vee A_2$, $e \bullet e' = m\{A_1\}$, $\forall y \in (\mathbb{\Gamma}_1).A_1$, $\exists y^{\alpha_{\text{-}(\mathsf{Nm},\rightarrow)}} \in (\mathbb{\Gamma}_1).A_1$, $\exists y \in (\mathbb{\Gamma}_1 \downarrow_{-TC}).A_1$ *are all* $\text{THIN}_{Syn}(x)$.

4. *If* A_1-$\text{THIN}_{Syn}(x)$ *and* $\delta \notin \mathsf{ftcv}(A_1)$ *then* $\forall \delta.A_1$-$\text{THIN}_{Syn}(x)$

5. *If* A_1-$\text{THIN}_{Syn}(x)$ *then* $\forall \delta.\forall y^{\alpha_y} \in (\mathbb{\Gamma}+\delta).A_1^{-\delta}$-$\text{THIN}_{Syn}(x)$

Thus $e\#\mathbb{\Gamma}_0$-$\text{THIN}_{Syn}(x)$ given $e\#\mathbb{\Gamma}_0 \stackrel{def}{=} \forall z \in (\mathbb{\Gamma}_0).z \neq e$-$\text{THIN}_{Syn}(x)$ as follows:
(2.) proves $z \neq e$-$\text{THIN}_{Syn}(x)$ given $x \notin \mathsf{fv}(e)$,
(2.) again and the above implies $\forall z \in (\mathbb{\Gamma}_0).z \neq e$-$\text{THIN}_{Syn}(x)$

This is an incomplete characterisation of all thin formulae and hence cannot be used to prove non-thinness, but covers all formulae required for the proofs. This definition follows that of the thinness of formulae in Local-logic in Sec. 2.2.3, adapted to the new logic [5, 72].

In Sec. 5.3.2 the thinness property is formally defined in the model and a proof that $\text{THIN}_{Syn}(x)$ formulae are indeed thin with respect to the semantics is given.

The following lemma is required in the soundness proofs, but included here for convenience (proximity to the previous definition). For variables $x : \alpha_1 \rightarrow \alpha_2$ and $y : \alpha_1$, the types ensure x is not of $\alpha_{\text{-}(\mathsf{Nm},\rightarrow)}$ type as it contains a function and hence cannot be formed from the first case of Def. 55.

Lemma 56 ($\text{THIN}_{Syn}(x)$ implies $\text{THIN}_{Syn}(y)$ under certain type conditions)**.**

$$\forall\, \mathbb{\Gamma}, x^{\alpha_1 \rightarrow \alpha_2}, y^{\alpha_1}, A.\ \mathbb{\Gamma}\backslash x, y \Vdash A \rightarrow A\text{-}\text{THIN}_{Syn}(x) \rightarrow A\text{-}\text{THIN}_{Syn}(y)$$

Proof. If $\alpha_1 \in \alpha_{\text{-}(\mathsf{Nm},\rightarrow)}$ then this clearly holds as A-$\text{THIN}_{Syn}(y)$ clearly holds by Def. 55. Otherwise, $\alpha_1 \rightarrow \alpha_2 \notin \alpha_{\text{-}(\mathsf{Nm},\rightarrow)}$ implies the first case in Def. 55 cannot be used and thus the definitions are identical if $x^{\alpha_1 \rightarrow \alpha_2}$ or y^{α_1} are used as $\mathbb{\Gamma}\backslash x, y \Vdash A$ meaning clearly A $\text{THIN}_{Syn}(x) \rightarrow A$ $\text{THIN}_{Syn}(y)$ holds.

□

This fails to hold in the opposite direction as $\alpha_1 \in \alpha_{\text{-}(\mathsf{Nm},\rightarrow)}$ causes issues in the first case of Def. 55 in proving A $\text{THIN}_{Syn}(x)$, but does hold if $\alpha_1 \notin \alpha_{\text{-}(\mathsf{Nm},\rightarrow)}$.

4.5 Logic of Axioms

Axioms and axiom schemas are similar in intention to those of the λ-logic seen in Sec. 2.2.2, but expressed within the constraints of the new logic. Axiom schemas are indexed by the LTC that types them and the explicit types where noted. The axioms and axiom schemas are referred to simply as axioms and are introduced here. This is neither a maximal nor

minimal list, but contains the axioms which are interesting and useful, i.e. those used in the reasoning examples seen in Chapt. 7.

Some axioms only hold for the types in the STLC, i.e. Nm-free types, written $\alpha_{\text{-Nm}}$. Other axioms use the type $\alpha_{\text{-(Nm,}\to\text{)}}$ which are the function free types of the STLC. These two types are defined as follows.

Definition 57 (Nm free types and Nm and function free types)**.** *Define two subsets of types as follows.*

$$\begin{array}{lcl} \alpha_{\text{-(Nm,}\to\text{)}} & ::= & \mathsf{Unit} \mid \mathsf{Bool} \mid \alpha_{\text{-(Nm,}\to\text{)}} \times \alpha_{\text{-(Nm,}\to\text{)}} \\ \alpha_{\text{-Nm}} & ::= & \mathsf{Unit} \mid \mathsf{Bool} \mid \alpha_{\text{-Nm}} \times \alpha_{\text{-Nm}} \mid \alpha_{\text{-Nm}} \to \alpha_{\text{-Nm}} \end{array}$$

A term which is of type $\alpha_{\text{-Nm}}$ may contain names for instance $\lambda x^{\mathsf{Bool}}.n = n'$, however it will be shown that all $\alpha_{\text{-Nm}}$ typed terms are equivalent to a nameless term.

4.5.1 Axioms for Equality

Axioms for **eq**uality are standard and found in Fig. 4.3, the three axioms $(eq1)$, $(eq2)$ and $(eq3)$ are reflexivity, symmetry and transitivity respectively. Axiom $(eq4)$ allows for substitution, however there are constraints that e-free for x in A must hold.

$$\begin{array}{lrcl} (eq1) & \mathbb{\Gamma} \Vdash e = e & & \\ (eq2) & \mathbb{\Gamma} \Vdash e = e' & \leftrightarrow & e' = e \\ (eq3) & \mathbb{\Gamma} \Vdash e = e' \wedge e' = e'' & \leftrightarrow & e = e'' \\ (eq4) & \mathbb{\Gamma} \Vdash A \wedge x = e & \leftrightarrow & A[e/x]_{\mathbb{\Gamma}} \end{array}$$

Figure 4.3: Axioms for equality of the ν-logic.

4.5.2 Axioms for Restricted Quantification

The axioms for **u**niversal restricted quantification alongside the axioms for existential restricted quantification are found in Fig. 4.4.

The axioms $(u1)$, $(u2)$ and $(u3)$ are instantiation, vacuous generalisation (and instantiation), and distribution respectively. These are inspired by $(u1)_\lambda$, $(u2)_\lambda$, $(u3)_\lambda$ which are axioms of first order logic.

Axiom $(u4)$ uses $\mathbb{\Gamma}_0 \Vdash \mathbb{\Gamma}_1$ which ensures that $\mathbb{\Gamma}_1$ is a subset of $\mathbb{\Gamma}_0$. Hence, if y quantifies over all expressions derived from $\mathbb{\Gamma}_0$, then clearly this implies quantification over

$$
\begin{array}{llrcll}
(u1) & \mathbb{\Gamma} \Vdash & \forall x^{\alpha} \in (\mathbb{\Gamma}_0).A & \rightarrow & A[e/x]_{\mathbb{\Gamma}} & \mathbb{\Gamma}_0 \vdash e : \alpha \\
(u2) & & A^{-x} & \leftrightarrow & \forall x \in (\mathbb{\Gamma}_0).A^{-x} & A\text{-}\textsc{ExtInd}_{Syn} \\
(u3) & & \forall x \in (\mathbb{\Gamma}_0).(A \wedge B) & \leftrightarrow & (\forall x \in (\mathbb{\Gamma}_0).A) \wedge (\forall x \in (\mathbb{\Gamma}_0).B) & \\
(u4) & & \forall x \in (\mathbb{\Gamma}_0).A & \rightarrow & \forall x \in (\mathbb{\Gamma}_1).A & \mathbb{\Gamma}_0 \Vdash \mathbb{\Gamma}_1 \\
(u5) & & \forall x^{\alpha} \in (\mathbb{\Gamma}_0).A & \leftrightarrow & \forall x^{\alpha} \in (\emptyset).A & \alpha \in \alpha_{\text{-}\mathsf{Nm}} \\
(u6) & \mathbb{\Gamma}{+}x{+}y \Vdash & x\#\mathbb{\Gamma} \wedge y\#\mathbb{\Gamma}{+}x & \rightarrow & \forall f^{\mathsf{Nm}\rightarrow\mathsf{Nm}} \in (\mathbb{\Gamma}).f \bullet x = m\{m \neq y\} & \\
(u7) & \mathbb{\Gamma}{+}x{+}y \Vdash & x\#\mathbb{\Gamma} \wedge y\#\mathbb{\Gamma}{+}x & \rightarrow & \forall f^{\mathsf{Nm}\rightarrow\mathsf{Bool}} \in (\mathbb{\Gamma}).f \bullet x = m\{f \bullet y = n\{m = n\}\} & \\
(u8) & \mathbb{\Gamma} \Vdash & \forall x^{\mathsf{Nm}} \in (\emptyset).\ x\#\mathbb{\Gamma} \wedge A & \leftrightarrow & \forall x^{\mathsf{Nm}} \in (\emptyset).\ A & \\
(u9) & \mathbb{\Gamma} \Vdash & \forall x^{\mathsf{Nm}} \in (\mathbb{\Gamma}_1).a \bullet x = e & \rightarrow & \forall x^{\mathsf{Nm}} \in (\mathbb{\Gamma}_1{+}a).a \bullet x = e & \mathbb{\Gamma}_1 \Vdash e : \mathsf{Nm} \\
(u10) & \mathbb{\Gamma}{+}a : \alpha \rightarrow \mathsf{Bool} \Vdash & \forall f^{\alpha} \in (\mathbb{\Gamma}).a \bullet f = c & \rightarrow & \forall f^{\alpha} \in (\mathbb{\Gamma}{+}a).a \bullet f = c & \\
 & & & & & \\
(ex1) & \mathbb{\Gamma} \Vdash & A[e/x]_{\mathbb{\Gamma}} & \rightarrow & \exists x \in (\mathbb{\Gamma}_0).x = e \wedge A & \mathbb{\Gamma} \Vdash \mathbb{\Gamma}_0 \text{ and } \mathbb{\Gamma}_0 \vdash e : \alpha \\
(ex2) & & A \wedge \exists x \in (\mathbb{\Gamma}_0).B & \leftrightarrow & \exists x \in (\mathbb{\Gamma}_0).(A \wedge B) & A\text{-}\textsc{ExtInd}_{Syn} \wedge x \notin \mathsf{fv}(A) \\
(ex3) & \mathbb{\Gamma}{+}x{+}\mathbb{\Gamma}_0 \Vdash & a \bullet b = c\{c = x\} & \rightarrow & \exists x' \in (\mathbb{\Gamma}_0).x = x' & \{a, b\} \subseteq \mathsf{fv}(\mathbb{\Gamma}_0) \\
(ex4) & \mathbb{\Gamma}{+}x \Vdash & \forall y \in (\emptyset).\exists z^{\mathsf{Nm}} \in (\mathbb{\Gamma}_0{+}y).x = z & \rightarrow & \exists z \in (\mathbb{\Gamma}_0).x = z &
\end{array}
$$

Figure 4.4: Axioms for universal and existential restricted quantification of the ν-logic.

all expression derived from $\mathbb{\Gamma}_1$ given $\mathbb{\Gamma}_1$ is a subset of $\mathbb{\Gamma}_0$. This allows for the reduction of LTCs in the restricted quantifier. Critically, $(u4)$ is *not* $\leftrightarrow$ as $\mathbb{\Gamma}_0$ may quantify over more expressions than $\mathbb{\Gamma}_1$, hence the $\leftarrow$ direction fails.

Axiom $(u5)$ shows how $\alpha_{\text{-}\mathsf{Nm}}$ types can be derived from any LTC as the values of $\alpha_{\text{-}\mathsf{Nm}}$ type are always equivalent to an STLC value.

Axioms $(u6)$ and $(u7)$ state that if x and y are fresh from $\mathbb{\Gamma}$ and each other, then any function derived from $\mathbb{\Gamma}$ cannot contain x or y and hence the right hand side holds, given these restrictions on f. The axiom $(u8)$ states that the x in $\forall x^{\mathsf{Nm}} \in (\emptyset).A$ is indeed always fresh from any global LTC which types the formula.

Axioms $(u9)$ and $(u10)$ are both used in specific cases where the quantifying LTC needs to be expanded. The axioms hold in specific circumstances. Axiom $(u9)$ requires that every output of a is some name e allowing the quantifying LTC to be extended with a. Axiom $(u10)$ requires a to be of type $\alpha \rightarrow \mathsf{Bool}$ meaning it can never be used to output a name, at best a can be used to compare names, however because the output is always the Boolean constant c, a can be added to the quantifying LTC.

Axioms for **ex**istential restricted quantification contain two standard axioms $(ex1)$ (the dual of $(u1)$) and $(ex2)$ which are inspired by $(ex1)_\lambda$ and $(ex2)_\lambda$ respectively with the additional requirement that A-ExtInd$_{Syn}$ in $(ex2)$ to ensure it holds when x is added/removed from the model satisfying A.

The new axiom $(ex3)$ allows for the introduction of the existential quantifier restricted to $\mathbb{\Gamma}_0$ given the evaluation of ab evaluates to some constant x, then if a and b are in $\mathbb{\Gamma}_0$ an existential quantifier can be introduced. It makes sense, that if a substitution of e for x in A holds and $\mathbb{\Gamma}_0$ types e then there exists an expression derived from the LTC $\mathbb{\Gamma}_0$ such that A holds. Axiom $(ex3)$ is a specific axiom, introducing the existential quantification from evaluation formulae of two variables in $\mathbb{\Gamma}_0$ which evaluate to a fixed result x which cannot be a fresh name.

Reducing $\mathbb{\Gamma}$ in $\exists x \in (\mathbb{\Gamma}).A$ is possible via $(ex4)$ for a specific structure where y is any expression derived from an empty LTC which equates to either a constant or a fresh name, however knowing that $z = x$ and x is a previously generated name means even if y is a fresh name then it is not required in the existential quantifier for z.

4.5.3 Axioms for Freshness

Axioms for the derived **f**reshness constructor are found in Fig. 4.5. The $(f1)$ axiom states when freshness implies equality, whereas $(f2)$ show instances LTCs can be reduced. The axioms $(f3)$ and $(f4)$ show instances LTCs can be extended under specific circumstances.

$$
\begin{array}{llrcll}
(f1) & & e\#\mathbb{\Gamma}_0 & \rightarrow & e \neq e' & \mathbb{\Gamma}_0 \vdash e' : \mathsf{Nm} \\
(f2) & & e\#\mathbb{\Gamma}_0 & \rightarrow & e\#\mathbb{\Gamma}_1 & \mathbb{\Gamma}_0 \Vdash \mathbb{\Gamma}_1 \\
(f3) & \mathbb{\Gamma}+x+f : \alpha \rightarrow \alpha_{\text{-}(\mathsf{Nm},\rightarrow)} \Vdash & x\#\mathbb{\Gamma} & \rightarrow & x\#\mathbb{\Gamma}+f : \alpha \rightarrow \alpha_{\text{-}(\mathsf{Nm},\rightarrow)} & \\
(f4) & \mathbb{\Gamma} \Vdash & e\#\mathbb{\Gamma}_0 \wedge \forall y^\alpha \in (\mathbb{\Gamma}_0).A & \leftrightarrow & \forall y^\alpha \in (\mathbb{\Gamma}_0).(e\#(\mathbb{\Gamma}_0+y) \wedge A) & y \notin \mathsf{fv}(e)
\end{array}
$$

Figure 4.5: Axioms for freshness of the ν-logic.

To deconstruct freshness, axiom $(f1)$ states that any expression e', which can be typed by the LTC $\mathbb{\Gamma}_0$ to be of type Nm, is guaranteed not to be equivalent to the name at e. This can be derived from the syntactic definition of $e\#\mathbb{\Gamma}_0 \stackrel{def}{=} \forall z^{\mathsf{Nm}} \in (\mathbb{\Gamma}_0).z \neq e$ and the axiom $(u1)$ instantiating z with e'.

Axiom $(u4)$ and the syntactic definition of freshness gives rise to the Axiom $(f2)$ i.e. if e cannot be derived from $\mathbb{\Gamma}_0$, then clearly any sub-LTC $\mathbb{\Gamma}_1$ of $\mathbb{\Gamma}_0$ cannot derive e either.

Clearly $(f2)$ fails to be $\leftrightarrow$ as $\mathbb{\Gamma}_0$ is larger than $\mathbb{\Gamma}_1$ and hence may reveal the name e.

Axiom $(f3)$ holds due to f being derived from $\mathbb{\Gamma}{+}x$, meaning no name can be derived using f and hence neither can the name x. Interactions between freshness and universal restricted quantification in axiom $(f4)$ states for an LTC $\mathbb{\Gamma}_0$ which the name x is fresh from, then any y derived from $\mathbb{\Gamma}_0$ by definition cannot reveal the name at e, hence the right hand side of $(f4)$ is obtained. The $\leftarrow$ direction of $(f4)$ is derivable from the axioms $(u3)$ and $(u2)$.

4.5.4 Axioms for Quantification Over LTCs

Axioms for **u**niversal quantification over logical **t**ype **c**ontexts are also similar to those for the classical universal quantifier except $(utc4)$ and $(utc5)$ which extends the restricted quantifier to any future LTC in two specific cases.

$(utc1)$		$\mathbb{\Gamma} \Vdash \quad \forall\delta.A$	$\rightarrow$	$A[\mathbb{\Gamma}/\delta]_{\mathbb{\Gamma}}$	
$(utc2)$		$A^{-\delta}$	$\leftrightarrow$	$\forall\delta.A^{-\delta}$	A-ExtInd$_{Syn}$
$(utc3)$		$\forall\delta.(A \wedge B)$	$\leftrightarrow$	$(\forall\delta.A) \wedge (\forall\delta.B)$	
$(utc4)$	$\mathbb{\Gamma} \Vdash$	$\forall x^{\mathsf{Nm}} \in (\mathbb{\Gamma}).A^{-\delta}$	$\leftrightarrow$	$\forall\delta.\forall x^{\mathsf{Nm}} \in (\mathbb{\Gamma}{+}\delta).A$	A-ExtInd$_{Syn}$
$(utc5)$	$\mathbb{\Gamma} \Vdash$	$\forall f^{\alpha} \in (\mathbb{\Gamma}).a \bullet f = c$	$\rightarrow$	$\forall\delta.\forall f^{\alpha} \in (\delta).a \bullet f = c$	$c \in \{\mathsf{true}, \mathsf{false}\}$

Figure 4.6: Axioms for universal quantification over LTCs of the ν-logic.

Axioms $(utc1)$, $(utc2)$ and $(utc3)$ are instantiation, vacuous generalisation (and instantiation), and distribution, respectively, inspired by $(u1)_{\lambda}$, $(u2)_{\lambda}$, $(u3)_{\lambda}$ which are those of first order logic.

Instantiation is given by $(utc1)$ taking the global LTC $\mathbb{\Gamma}$, which types the formula, as the LTC to replace the TCV δ in A. Axiom $(utc2)$ mimics $(u2)$ where δ does not occur in $\mathsf{ftcv}(A)$ (written $A^{-\delta}$) hence the quantifier can be introduced/removed assuming A-ExtInd$_{Syn}$. The axiom $(utc3)$ allows the quantifier over LTCs to be merged and split over conjunction $\wedge$, similar to $(u3)$ but for quantification over LTC.

In the special case where a name x is derived from the global LTC $\mathbb{\Gamma}$ then this is equivalent to introducing a "$\forall\delta.$" and deriving a name from δ as can be seen in axiom $(utc4)$. The reasoning behind $(utc4)$, is that on the left hand side, the name x can either be derivable from $\mathbb{\Gamma}$ or be completely fresh and hence on the right hand side, x will be derivable from δ which includes $\mathbb{\Gamma}$ and any new name which will be fresh from the extension

to $\mathbb{\Gamma}$ i.e. δ.

The axiom $(utc5)$ is similar to $(utc4)$. It states that if all terms f, derived from the global LTC, satisfy $a \bullet f = c$ for some Boolean c, then any f derived from a future state also satisfies $a \bullet f = c$. This axiom is not dependent on the actual Boolean result c, only the fact that af always returns the same Boolean constant. The axiom $(utc5)$ allows for the LTC in a universal quantification to be extended to all future LTCs, if the formula is of this specific form.

4.5.5 Axioms for Evaluation Formulae

The axioms for the **e**valuation formulae are similar to those of the λ-logic and are introduced in Fig. 4.7. The axiom (ext) maintains extensionality in this logic for the $\alpha_{\text{-Nm}}$ typed expressions and requires the following definition for $\mathsf{Ext}(e_1, e_2)$.

$$\mathsf{Ext}(e_1, e_2) \stackrel{def}{=} \forall x \in (\emptyset).e_1 \bullet x = m_1\{e_2 \bullet x = m_2\{m_1 = m_2\}\}$$

The (ext) axiom fails on non-$\alpha_{\text{-Nm}}$ types. The simple counter example with e_1 and e_2 being the gensym function, then clearly $\mathsf{gensym} = \mathsf{gensym}$, however $e_1 \bullet () = m_1\{e_2 \bullet () = m_2\{m_1 \neq m_2\}\}$ holds by definition as m_1 and m_2 are fresh from each other.

$$
\begin{array}{lrcll}
(ext) & \mathsf{Ext}(e_1, e_2) & \leftrightarrow & e_1 =^{\alpha_1 \to \alpha_2} e_2 & (\alpha_1 \to \alpha_2) \in \alpha_{\text{-Nm}} \\
(e_\alpha) & e \bullet e' = m\{e \bullet e' = a\{a = m \wedge A\}\} & \leftrightarrow & e \bullet e' = m^\alpha\{A\} & \alpha \in \alpha_{\text{-Nm}},\ a \notin \mathsf{fv}(e, e', A) \\
(e1) & e_1 \bullet e_2 = m^\alpha\{A\} \wedge e_1 \bullet e_2 = m^\alpha\{B\} & \leftrightarrow & e_1 \bullet e_2 = m^\alpha\{A \wedge B\} & \\
(e2) & e_1 \bullet e_2 = m^\alpha\{\neg A\} & \leftrightarrow & \neg e_1 \bullet e_2 = m^\alpha\{A\} & \\
(e3) & e_1 \bullet e_2 = m\{A \wedge B\} & \leftrightarrow & A \wedge e_1 \bullet e_2 = m\{B\} & A\text{-ExtInd}_{Syn},\ m \notin \mathsf{fv}(A) \\
(e4) & e_1 \bullet e_2 = m\{\forall x \in (\mathbb{\Gamma}).A\} & \leftrightarrow & \forall x \in (\mathbb{\Gamma}).e_1 \bullet e_2 = m\{A\} & m \notin \mathsf{fv}(\mathbb{\Gamma}),\ x \notin \mathsf{fv}(e_1, e_2, m) \\
(e5) & e_1 \bullet e_2 = m^{\alpha\text{-}(\mathsf{Nm},\to)}\{\forall \delta.A\} & \leftrightarrow & \forall \delta.e_1 \bullet e_2 = m^{\alpha\text{-}(\mathsf{Nm},\to)}\{A\} & A\text{-ExtInd}_{Syn}
\end{array}
$$

Figure 4.7: Axioms for evaluation formulae of the ν-logic.

All STLC values are included in the variables of $\alpha_{\text{-Nm}}$ type as stated previously. The axioms (ext) and (e_α) require $\alpha_{\text{-Nm}}$ types as these are direct copies of λ-logic axioms, thus ensuring the logic is a conservative extension of the λ-logic(See Sec. 6.5). These axioms may hold for other types but are not required in the reasoning examples.

Axioms $(e1)$, $(e2)$ and $(e3)$ are standard axioms from the λ-logic with the constraint that A-ExtInd$_{Syn}$ in $(e3)$ to ensure the formula holds if m is added or removed from the global LTC (or model).

The interaction between evaluation formulae and restricted quantification, and quantification over LTCs, are shown in $(e4)$ and $(e5)$ respectively. These behave as the λ-logic axiom $(e4)_\lambda$, but with more constraints due to the LTC. A clear comparison of $(e5)$ and $(e4)_\lambda$ is apparent, with a swap in variable/TCV. In $(e4)$ the restrictions on $\mathbb{\Gamma}$ not containing m ensures the LTC is not interfering with the anchor in the evaluation formula, similarly x not occurring in e_1, e_2 or m ensures no interference with the quantifying variable with the evaluation formula.

The additional constraints of $\textsc{ExtInd}_{Syn}$ in $(e3)$ and $(e5)$ ensures A holds when variables/TCVs are added and removed from the global LTC (or model) which is a book keeping requirement in the soundness proof.

4.6 Logic of Rules

The logic of rules is now introduced for the ν-logic. This allows for the reasoning about static-syntax ν_{GS}-calculus programs using the triples introduced in Sec. 4.1.7. This requires the logic of axioms introduced in the previous section.

The use of $\vdash_\nu \{A\}\ M :_m \{B\}$ indicates that $\{A\}\ M :_m \{B\}$ can be derived from these rules and the previously introduced axioms, however this notation is dropped where obvious (i.e. in the rules themselves). These rules are similar to those of the λ-logic and the Local-logic, but suitably adapted to the effectful nature of the ν-logic.

All rules are typed following the corresponding typing of the programs occurring in the triples, but with additions to account for auxiliary variables. These types can be included or dropped where needed.

4.6.1 Core Rules

The core rules of inference can be found in Fig. 4.8.

One primary difference with these ν-logic rules is the logical substitution introduced in Sec. 4.3 which has no effect on the rules $[\textsc{Var}]_\nu$, $[\textsc{Const}]_\nu$, $[\textsc{If}]_\nu$ and $[\textsc{Pair}]_\nu$, and a minor effect on $[\textsc{Eq}]_\nu$ and $[\textsc{Proj}(i)]_\nu$. The latter two rules substitute for a variable u, the expressions $m = n$ and $\pi_i(m)$, both of which cannot add their constituent variables (m, n) to an LTC without potentially introducing hidden names. Hence, e-free for u in A ensures all LTCs containing u must also type $m = n$ or $\pi_i(u)$ respectively.

The other difference from the λ-logic is the need for thinness to replace the standard "free from" in the λ-logic often denoted $A^{\text{-}x}$. Thinness is introduced in the Local-logic and is also required here for the soundness proof. For example $[\textsc{App}]_\nu$, which produces u from

$$\frac{-}{\{A[x/m]\}\ x :_m \{A\}}\ [\textsc{Var}]_\nu \qquad \frac{-}{\{\mathsf{T}\}\ \mathsf{gensym} :_u \{\forall\delta.u \bullet () = m\{m\#\delta\}\}}\ [\textsc{Gensym}]_\nu$$

$$\frac{-}{\{A[\mathsf{c}/m]\}\ \mathsf{c} :_m \{A\}}\ [\textsc{Const}]_\nu \qquad \frac{\{A\}\ M :_m \{B\} \quad \{B\}\ N :_n \{C[m = n/u]\}}{\{A\}\ M = N :_u \{C\}}\ [\textsc{Eq}]_\nu$$

$$\frac{\mathbb{\Gamma}{+}\delta{+}x : \alpha \Vdash \{A \wedge B\}\ M :_m \{C\} \quad A\text{-}\textsc{ExtInd}_{Syn}}{\mathbb{\Gamma} \Vdash \{A\}\ \lambda x^\alpha.M :_u \{\forall\delta.\forall x^\alpha \in (\delta).(B \to u \bullet x = m\{C\})\}}\ [\textsc{Lam}]_\nu$$

$$\frac{\{A\}\ M :_m \{B\} \quad \{B\}\ N :_n \{m \bullet n = u\{C\}\}}{\{A\}\ MN :_u \{C\}}\ [\textsc{App}]_\nu \qquad \frac{\{A\}\ M :_m \{C[\neg m/u]\}}{\{A\}\ \neg M :_u \{C\}}\ [\textsc{Neg}]_\nu$$

$$\frac{\{A\}\ M :_m \{B\} \quad \{B[\mathsf{true}/m]\}\ N_1 :_u \{C\} \quad \{B[\mathsf{false}/m]\}\ N_2 :_u \{C\}}{\{A\}\ \mathsf{if}\ M\ \mathsf{then}\ N_1\ \mathsf{else}\ N_2 :_u \{C\}}\ [\textsc{If}]_\nu$$

$$\frac{\{A\}\ M :_m \{B\} \quad \{B\}\ N :_n \{C[\langle m, n\rangle/u]\}}{\{A\}\ \langle M, N\rangle :_u \{C\}}\ [\textsc{Pair}]_\nu \qquad \frac{\{A\}\ M :_m \{C[\pi_i(m)/u]\}}{\{A\}\ \pi_i(M) :_u \{C\}}\ [\textsc{Proj}(i)]_\nu$$

Figure 4.8: Inference rules of the ν-logic. The rule $[\textsc{Proj}(i)]_\nu$ requires $C\text{-}\textsc{Thin}_{Syn}(m)$ and $[\textsc{Eq}]_\nu$, $[\textsc{App}]_\nu$, $[\textsc{Pair}]_\nu$ require $C\text{-}\textsc{Thin}_{Syn}(m, n)$. The non-essential LTCs are omitted in typing and substitutions.

m and n, hence $\mathbb{\Gamma}{+}m{+}n{+}u \Vdash C$ implies that $\textsc{ExtInd}_{Syn}$ is insufficient, given u is still required and $\mathbb{\Gamma}{+}u$ is not a contraction of $\mathbb{\Gamma}{+}m{+}n{+}u$ as u introduced after m and n. This will be discussed more when the model has been introduced in Sec. 5.3.2 and the soundness proofs in Sec. 6.3.

The rules $[\textsc{Var}]_\nu$, $[\textsc{Const}]_\nu$, $[\textsc{Eq}]_\nu$, $[\textsc{App}]_\nu$, $[\textsc{Neg}]_\nu$, $[\textsc{If}]_\nu$, $[\textsc{Pair}]_\nu$ and $[\textsc{Proj}(i)]_\nu$ are those of the Local-logic with the changes discussed above regarding logical substitution and syntactic-thinness and are not discussed further.

In the post-condition of the $[\textsc{Gensym}]_\nu$ rule, $u \bullet () = m\{m\#\delta\}$ indicates that the name produced by $u()$ and stored at m is not derivable from the LTC δ. If there were no quantifications over LTCs prior to the evaluation, m could at most, only be fresh from the global LTC which types the formula. However, the freshness of the name produced by $u()$ must hold in every future typing context in which the evaluation occurs. To ensure freshness in future LTCs (or states) quantification over LTCs, "$\forall\delta$.", is introduced, thus quantifying over all future names derivable from the future state. Placing he quantification over LTCs prior to the evaluation of $u()$ ensures that instantiating the δ cannot include

the anchor of $u()$ which ensures $m\#m$ cannot be derived. Elsewhere in reasoning it is key that the post-condition of $[\text{GENSYM}]_\nu$ is EXTIND_{Syn} and hence holds in all future and past LTCs assuming the anchor for gensym is present. This ensures application of gensym in any context produces a fresh name each time it is applied.

Rules for λ-abstraction $[\text{LAM}]_\lambda$ in the λ-logic universally quantifies over all possible arguments. This needs to be restricted in this logic as hidden names should not be quantified over. The corresponding ν-logic rule $[\text{LAM}]_\nu$, refines this and quantifies only over current or future values that do not reveal hidden names. A key example for this is the term $\mathsf{let}\ x = \mathsf{gensym}()\ \mathsf{in}\ \lambda y.x = y$, which contains the name stored at x, but can never use the name as it is hidden under an equality. Hence reasoning about this program (See Ex. 29), initially quantifies over all future LTCs which includes x. However, in removing the quantification over x, this allows for the derivation that the function always outputs false.

Comparing the two LTCs typing the assumption and conclusion of the $[\text{LAM}]_\lambda$ rule, implies δ is an extension of $\mathbb{\Gamma}$ and x is derived from $\mathbb{\Gamma}+\delta$. Hence the typing implies precisely what is conveyed in the post-condition of the conclusion: "$\forall\delta.\forall x \in (\delta)$.". The introduction of the formula B allows for the introduction of constraints on δ and x. Requiring A-EXTIND_{Syn} implies A still holds in all extensions of $\mathbb{\Gamma}$ including $\mathbb{\Gamma}+\delta+x$. The evaluation formula part is trivial when $((\lambda x.M)x) \cong^G_\alpha M$ is considered.

4.6.2 Structural Rules

The structural rules are found in Fig. 4.9.

$$\frac{A \to A' \quad \{A'\}\ M :_m \{B'\} \quad B' \to B}{\{A\}\ M :_m \{B\}}\ [\textsc{Conseq}]_\nu \qquad \frac{\{A\}\ M :_m \{B\} \quad C\text{-}\textsc{ExtInd}_{Syn}}{\{A \wedge C\}\ M :_m \{B \wedge C\}}\ [\textsc{Invar}]_\nu$$

$$\frac{\{A \wedge B\}\ M :_m \{C\} \quad B\text{-}\textsc{ExtInd}_{Syn}}{\{A\}\ M :_m \{B \to C\}}\ [\wedge \to]_\nu \qquad \frac{\{A\}\ M :_m \{B \to C\} \quad B\text{-}\textsc{ExtInd}_{Syn}}{\{A \wedge B\}\ M :_m \{C\}}\ [\to \wedge]_\nu$$

$$\frac{\{A\}\ M :_m \{B\} \quad \{A'\}\ M :_m \{B\}}{\{A \vee A'\}\ M :_m \{B\}}\ [\vee\text{-}\textsc{Pre}]_\nu \qquad \frac{\{A\}\ M :_m \{B\} \quad \{A\}\ M :_m \{B'\}}{\{A\}\ M :_m \{B \wedge B'\}}\ [\wedge\text{-}\textsc{Post}]_\nu$$

$$\frac{\mathbb{\Gamma} \Vdash \{A\}\ M :_m \{B\} \quad A, B\text{-}\textsc{ExtInd}_{Syn}}{\mathbb{\Gamma}{+}x \Vdash \{A\}\ M^{-x} :_m \{B\}}\ [\textsc{Weak}(x)]_\nu$$

$$\frac{\mathbb{\Gamma}{+}\mathbb{\Gamma}' \Vdash \{A\}\ M :_m \{B\} \quad A, B\text{-}\textsc{ExtInd}_{Syn}}{\mathbb{\Gamma}{+}\delta{+}\mathbb{\Gamma}' \Vdash \{A\}\ M :_m \{B\}}\ [\textsc{Weak}(\delta)]_\nu$$

Figure 4.9: Structural inference rules of the ν-logic.

The $[\textsc{Conseq}]_\nu$ rule introduces the logic of axioms to the logic of rules. This rule is identical to $[\textsc{Invar}]_\lambda$.

The $[\textsc{Invar}]_\nu$ rule extends the $[\textsc{Invar}]_\lambda$ rule with the constraint that C-$\textsc{ExtInd}_{Syn}$ to ensure C holds in the extension where m has been assigned. The $[\wedge \to]_\nu$, $[\to \wedge]_\nu$, $[\vee\text{-}\textsc{Pre}]_\nu$ and $[\wedge\text{-}\textsc{Post}]_\nu$ rules are lifted directly from the λ-logic, with the addition of B-$\textsc{ExtInd}_{Syn}$ required in the first two. These are standard rules and allow for the manipulation of standard predicate logic constructors via rules.

The $[\textsc{Weak}(x)]_\nu$ and $[\textsc{Weak}(\delta)]_\nu$ rules are only used in the soundness proof of the derived $[\textsc{Let}]_\nu$ rule and hence also the $[\textsc{LetFresh}]_\nu$ rule introduced in the next section. The $[\textsc{Weak}(x)]_\nu$ rule allows for the addition (or removal depending on perspective) of variables as an extension to the LTC that types the triple assuming both pre- and post-conditions are $\textsc{ExtInd}_{Syn}$ and do not contain the variable being added. The $[\textsc{Weak}(\delta)]_\nu$ rule allows for the addition (or removal) of a TCV to the LTC that types the triple in any location, assuming the pre- and post-conditions are $\textsc{ExtInd}_{Syn}$ and do not contain the TCV in an unbound state. Both rules are introduced specifically for the derived rules in the following section and no applications in any reasoning examples have been found yet.

4.6.3 Derived Rules

Two rules are introduced that are useful in reasoning about examples but are derived from the previous rules, these are introduced in Fig. 4.10.

$$\frac{\{A\}\ M :_m \{B\} \quad \{B\}\ N :_u \{C\} \quad A, B, C - \textsc{ExtInd}_{Syn}}{\{A\}\ \mathsf{let}\ m = M\ \mathsf{in}\ N :_u \{C\}}\ [\textsc{Let}]_\nu$$

$$\frac{\{A \wedge m\#\mathbb{\Gamma}\}\ M :_u \{C\} \quad A, C\text{-}\textsc{ExtInd}_{Syn}}{\{A\}\ \mathsf{let}\ m = \mathsf{gensym}()\ \mathsf{in}\ M :_u \{C\}}\ [\textsc{LetFresh}]_\nu$$

Figure 4.10: Derived inference rules of the ν-logic. In both $[\textsc{Let}]_\nu$ and $[\textsc{LetFresh}]_\nu$ the requirement that $C\text{-}\textsc{Thin}_{Syn}(m)$ is required.

The STLC's $[\textsc{Let}]_\lambda$ rule can be written to introduce x in the post-condition by means of an "$\exists x.C$". This fails here as x may be unreachable, hence not derivable from any extending or contracting LTC. The requirement that $C\text{-}\textsc{Thin}_{Syn}(x)$ ensures x is not critical to C so can either be derived from the current LTC or is hidden. Thinness ensures no reference to the variable m is somehow hidden under quantification over LTCs.

The $[\textsc{LetFresh}]_\nu$ rule is commonly used and hence included for convenience, but it is entirely derivable from the other rules ($[\textsc{Let}]_\nu$ $[\textsc{Gensym}]_\nu$, $[\textsc{Const}]_\nu$ and $[\textsc{App}]_\nu$) given the triple $\mathbb{\Gamma} \Vdash \{\mathsf{T}\}\ \mathsf{gensym}() :_m \{m\#\mathbb{\Gamma}\}$ proven in Ex. 24 in Chapt. 7.

Given the translation in Sec. 3.3, $\langle\!\langle \nu n.M \rangle\!\rangle_{Pitts \to GS} \stackrel{def}{=} \mathsf{let}\ n = \mathsf{gensym}()\ \mathsf{in}\ \langle\!\langle M \rangle\!\rangle_{Pitts \to GS}$, then $[\textsc{LetFresh}]_\nu$ is essentially the inference rule for $\nu\mathsf{n}.M$.

4.7 Alternative Design Choices

Some alternative design choices for the construction of the program logic are introduced here and discussed briefly.

4.7.1 gensym as a Constant in the Logic

When designing a program logic there is a balance between what is placed in the logic of axioms and what is placed in the logic of rules. For instance the expressions $\pi_i(e)$ and $\langle e, e' \rangle$ could be placed into the formulae in a form of pair constructors and pair destructors which would require axioms and rules which allow for manipulation of these formulae.

Similarly the evaluation formulae could be expressed as an expression with a different, more complicated, yet equally powerful, set of axioms.

When considering the construction of this logic, various alternatives were considered. The primary alternative involved including gensym as a constant in the expressions and then letting the derivation of freshness stem from the axioms. Primarily the axiom of the form "$\forall\delta.\mathsf{gensym}\bullet() = m\{m\#\delta\}$" would allow for the derivation of $[\textsc{Gensym}]_\nu$, harnessing the power of the $[\textsc{App}]_\nu$ rule. However, this made the other definitions more complex such as substitution and requirements on some of the other axioms to be more restricted to account for the expression gensym. Using gensym as a constant is not expected to strengthen or weaken the logic, but it is expected to make the applications and proofs less intuitive.

4.7.2 The Use of LTCs

The reason for the introduction of LTCs is not obvious. A list of names in place of LTCs is insufficient, as it fails to quantify over hidden names. A list of expressions in place of LTCs contains no structure, hence reasoning is difficult, in particular substitution and instantiation. In both these cases, the concept of extensions also becomes less apparent, as extensions cannot be referenced within them so clearly. The need to quantify over future states, means the LTC must be able to refer to the future states by name (or TCV). This ensures LTCs must include TCVs and the ordering of the LTC ensures this happens in a formulaic manner. This accumulates to a single form of LTC with a dual purpose, both typing the logic and used within the logical formulae.

4.7.3 Separating "Derived" and "Quantification"

The common use of the mathematical symbol "$\in$" is used to mean an element of a set. Universal quantification often uses this by default as "$\forall x \in S.A$" to mean for all elements x in the set S then A holds. This can be separated to mean "$\forall x.x \in S \to A$" where "$x \in S$" checks whether x is in the set S. A similar separation could be introduced in the ν-logic such that $\forall x \in (\mathbb{\Gamma}).A$ becomes "$\forall x.x \in \mathbb{\Gamma} \to A$" where "$x \in \mathbb{\Gamma}$" does not represent "element of" but instead "derived from". It is not known if the logic for splitting the "derived from" can be generalised in a succinct form. However, the full power of this generalisation was not required for reasoning about names in this logic.

4.7.4 Syntactic Characterisations of Properties of Formulae

The definition of $\textsc{Thin}_{Syn}(x)$ is not a complete list of all thin formulae however it covers all cases required in the examples reasoned about. A more complete definition would be useful such that the syntactically thin and semantically thin formulae, introduced in the next chapter in Def. 78, match precisely. However, this seemed to be a complicated task as the non-thin formulae are not common and no pattern was found when constructing these non-thin formulae.

4.8 Summary

This chapter introduced ν-logic. The logical syntax is introduced with the primary novelty being the introduction of the ordered LTC, with a new mapping of TCVs representing extensions of type contexts. LTCs are used in the logical syntax to restrict the expressions (and thus names) which can be quantified over in the new restricted quantification constructor. This new constructor builds on the inability for the language to reintroduce hidden names, which do not have a reference, out of thin air. The quantification needs to derive values from a set of other values, i.e. the LTCs. A new quantifier over all future type contexts is introduced, with the ability to name the future state using TCVs.

Relevant changes are made to substitution to satisfy the new constructors. Syntactic properties of formulae are introduced to ensure the certain requirements are held in the soundness proofs in later sections. The logic of axioms and rules are introduced to allow for the reasoning about ν_{GS}-calculus programs.

Certain choices made throughout the construction of the logic are introduced and discussed to show the motivation behind certain aspects of the logic.

Chapter 5

Model

The soundness proof for the λ-logic requires a model which maps variables from the typing context to closed values, as seen in Sec. 2.2.2. This approach is built upon here for the program logic for the ν_{GS}-calculus, introduced in Chapt. 4, which will be used for the proof of soundness in Chapt. 6.

The model is defined in Sec. 5.1 with the interpretation of expressions introduced in Sec. 5.2 and the semantics of formulae and triples introduced in Sec. 5.2. The definitions of syntactic extension independence and syntactic thinness of formulae introduced in Sec. 4.4 are given semantic definitions in Sec. 5.3. These are specifically required to ensure that the soundness proofs hold.

Using these definitions, core lemmas are introduced in Sec. 5.4 which are used throughout the soundness proof in the next chapter.

5.1 Defining the Model

The model required to prove the λ-logic sound is typed by the standard type contexts (STC), which are unordered maps from variables to closed values, of the type given by the STC. Whereas any closed value in the STLC can be written by the programmer, the ν_{GS}-calculus does not have this luxury. Consider the simple example of a particular fresh name n, the programmer cannot mention this name directly as the only tool to reference names is the gensym operator which by definition will not produce the desired fresh name n if it is in the nameset. If the name n appears in the model then using the relevant variable could allow this name to be accessed. However the name may appear in the model in an unreachable form, for instance if the name only appears in the term $\lambda x.x = \mathsf{n}$, then without any other access to the name n, the name n cannot be reproduced. This means

this function will never be applied to n, meaning the function is equivalent to $\lambda x.\mathsf{false}$, this is not the case if n is accessible as $(\lambda x.x = \mathsf{n})\mathsf{n}$ returns true.

In the ν-logic, future states are mentioned using $\forall\delta.A$ which could be modelled naively as the addition of any mappings to the model. However, consider the example above: $(\lambda x.x = \mathsf{n})$. Where n does not appear elsewhere in the model, this means the function is equivalent to $\lambda x.\mathsf{false}$. If the name n is added to the model in some extension, then this equivalence no longer holds. For this reason the model is now based on the Logical Type Contexts which now contain Type Context Variables (TCVs) and an inherent order. This now places a restriction on the possible model extensions, by stating that for any value added to a model, it must have been derived from the model it is being added to. By definition, this restricts the example above where n cannot be added to a model if it only appears hidden in the model, for example in $\lambda x.x = \mathsf{n}$. This is formally defined as follows.

Models now map variables to closed values, and TCVs to closed LTCs, i.e. TCV-free LTCs. If TCV were to map to an open LTC, then a secondary(or more) call to the model would be required to close the LTC. This is similar to the requirement of models mapping to closed values

Definition 58 (Model)**.** *A model is a map from variables to closed values and TCVs to closed LTCs.*

$$\xi \quad ::= \quad \emptyset \;\mid\; \xi \cdot x : V \;\mid\; \xi \cdot \delta : \mathbb{\Gamma}$$
$$\text{where } V \text{ and } \mathbb{\Gamma} \text{ are closed}$$

Key functions on models are as expected and defined as follows. The domain of the model ξ written $\mathsf{dom}(\xi)$ contains all variables and TCVs that are mapped by the model.

Definition 59 (Model assignment)**.** *Use $\xi \cdot x : V$ as the* assigning *of a value V to variable x in the model ξ, where $x \notin \mathsf{dom}(\xi)$. Similarly $\xi \cdot \delta : \mathbb{\Gamma}_0$ means the assignment of LTC $\mathbb{\Gamma}_0$ to TCV δ in the model ξ assuming δ doesn't occur in ξ*

Definition 60 (Model mapping)**.** *Define the value obtained by variable x in a model, ξ, as $\xi(x)$ or by a TCV, δ, in model ξ as $\xi(\delta)$. The result is either a closed value (i.e. contains no free variables), or a closed LTC, $\mathbb{\Gamma}$ (i.e. contains no TCVs).*

Whereas in most formalisations, the removal of a variable, x, from a model, ξ, is simply "the removal of that variable", this needs clarification in this logic due to the TCVs.

Definition 61 (Model variable removal)**.** *Removing the variable x from the model ξ written $\xi\backslash x$: removes x from the mapping of variables, and from any LTC $\mathbb{\Gamma}$, mapped to by a TCV*

in the model, written $\mathbb{\Gamma}\backslash x$*. Both* $\xi\backslash x$ *and* $\mathbb{\Gamma}\backslash x$ *are formally defined as follows.*

$$\begin{array}{rcl@{\qquad}rcl}
\emptyset\backslash x & \stackrel{\text{def}}{=} & \emptyset & \emptyset\backslash x & \stackrel{\text{def}}{=} & \emptyset \\
(\xi \cdot x : V_x)\backslash x & \stackrel{\text{def}}{=} & \xi\backslash x & (\mathbb{\Gamma}_1{+}x : \alpha_x)\backslash x & \stackrel{\text{def}}{=} & \mathbb{\Gamma}_1 \\
(\xi \cdot y : V_y)\backslash x & \stackrel{\text{def}}{=} & (\xi\backslash x) \cdot y : V_y & (\mathbb{\Gamma}_1{+}y : \alpha_y)\backslash x & \stackrel{\text{def}}{=} & (\mathbb{\Gamma}_1\backslash x){+}y : \alpha_y \\
(\xi \cdot \delta : \mathbb{\Gamma}_1)\backslash x & \stackrel{\text{def}}{=} & (\xi\backslash x) \cdot \delta : (\mathbb{\Gamma}_1\backslash x) & (\mathbb{\Gamma}_1{+}\delta : \mathbb{TC})\backslash x & \stackrel{\text{def}}{=} & (\mathbb{\Gamma}_1\backslash x){+}\delta : \mathbb{TC}
\end{array}$$

The final case above, $(\mathbb{\Gamma}_1{+}\delta : \mathbb{TC})\backslash x$ should never be used as $\xi \cdot \delta : \mathbb{\Gamma}_1$ requires $\mathbb{\Gamma}_1$ to be closed and thus TCV-free, however it is included here for completeness.

The standard $\Gamma\backslash x$ holds as the standard removal of the variable x from the STC Γ.

Definition 62 (Term closure)**.**
The closure of a term M*, by a model* ξ*, written* $M\xi$*, assuming* $\mathsf{fv}(M) \subseteq \mathsf{dom}(\xi)$ *is defined as standard with the additions,* $\mathsf{gensym}\xi \stackrel{\text{def}}{=} \mathsf{gensym}$ *and* $\mathsf{n}\xi \stackrel{\text{def}}{=} \mathsf{n}$ *. This is defined in full as follows.*

$$\begin{array}{rcll}
x\xi & \stackrel{\text{def}}{=} & \xi(x) & \\
c\xi & \stackrel{\text{def}}{=} & c & c \in \{(), \mathsf{true}, \mathsf{false}\} \\
(\lambda x.M)\xi & \stackrel{\text{def}}{=} & \lambda x.(M(\xi\backslash x)) & \\
(MN)\xi & \stackrel{\text{def}}{=} & (M\xi)(N\xi) & \\
(\mathsf{let}\ x = M\ \mathsf{in}\ N)\xi & \stackrel{\text{def}}{=} & \mathsf{let}\ x = M\xi\ \mathsf{in}\ N\xi\backslash x & \\
(M = N)\xi & \stackrel{\text{def}}{=} & M\xi = N\xi & \\
(\pi_i(M))\xi & \stackrel{\text{def}}{=} & \pi_i(M\xi) & \\
(\langle M, N\rangle)\xi & \stackrel{\text{def}}{=} & \langle M\xi, N\xi\rangle & \\
(\mathsf{if}\ M\ \mathsf{then}\ N_1\ \mathsf{else}\ N_2)\xi & \stackrel{\text{def}}{=} & \mathsf{if}\ M\xi\ \mathsf{then}\ N_1\xi\ \mathsf{else}\ N_2\xi & \\
(\mathsf{gensym})\xi & \stackrel{\text{def}}{=} & \mathsf{gensym} & \\
\mathsf{n}\xi & \stackrel{\text{def}}{=} & \mathsf{n} \qquad \textit{where } \mathsf{n} \textit{ is a name.} &
\end{array}$$

Clearly TCV are not required for term closure hence: $M\xi\backslash_{-TCV} \equiv M\xi \equiv M(\xi \cdot \delta : \mathbb{\Gamma}')$ *holds for all* δ *and* $\mathbb{\Gamma}'$*.*

Term closure can be thought of as the environment in which terms are run. Term closure replaces the free variables by values that have previously been produced and stored in the model, hence the following lemma holds trivially.

Lemma 63 (Closure equivalent to substitution)**.** *Closure can be interchanged with substitution:* $M(\xi \cdot x : V) \equiv (M[V/x])\xi$
Proof by induction on the structure of M*.*

Similar to the operation on LTCs to remove all TCVs $\backslash_{-TCV}$ a similar operation on models is defined.

Definition 64 (Removing TCVs from a model). *The removal of all TCVs from a model ξ, written $\xi\backslash_{-TCV}$ is defined as follows.*

$$\begin{aligned}\emptyset\backslash_{-TCV} &\stackrel{\text{def}}{=} \emptyset\\ (\xi \cdot x : V)\backslash_{-TCV} &\stackrel{\text{def}}{=} (\xi\backslash_{-TCV}) \cdot x : V\\ (\xi \cdot \delta : \mathbb{\Gamma}')\backslash_{-TCV} &\stackrel{\text{def}}{=} \xi\backslash_{-TCV}\end{aligned}$$

The $\mathring{\mathsf{a}}(\cdot)$ function for terms in Def. 20, is extended to models.

Definition 65 (All names in an environment). *All names that appear in a model ξ written $\mathring{a}(\xi)$, is a function defined as the union of all names in the codomain of the model as follows. Here X ranges over variables and TCVs.*

$$\mathring{a}(\xi) \stackrel{\text{def}}{=} \bigcup_{X \in \mathsf{dom}(X)} \mathring{a}(\xi(X))$$

For all LTCs, $\mathring{a}(\mathbb{\Gamma}) = \emptyset$ ensures $\mathring{a}(\xi\backslash_{-TCV}) \stackrel{\text{def}}{=} \mathring{a}(\xi)$.

In the λ-logic in Sec. 2.2.2 a model was typed by a type context. This required every variable in the type context to be mapped by the model to a closed value of the type given by the type context for that variable, i.e. $\xi^{\Gamma} \stackrel{def}{=} \mathsf{dom}(\Gamma) = \mathsf{dom}(\xi) \wedge \forall x \in \mathsf{dom}(\Gamma).\xi(x) : \Gamma(x)$. This new model for the ν_{GS}-calculus extends this idea to use LTCs as a basis. Conceptually a model is *typed* by an LTC if the model values match the LTC types for each variable and for any TCV in the LTC which appears as $\mathbb{\Gamma}_0 + \delta$, then the model maps δ to an LTC, which is a subset of $\mathbb{\Gamma}_0$. This if formally defined as follows.

Definition 66 (Typed model). *A model ξ is typed by an LTC $\mathbb{\Gamma}$, written $\mathbb{\Gamma} \Vdash \xi$ if it can be typed using the following rules. If $\mathbb{\Gamma} \Vdash \xi$ then for brevity this is written as $\xi^{\mathbb{\Gamma}}$.*

$$\frac{-}{\emptyset \Vdash \emptyset} \qquad \frac{\mathbb{\Gamma} \Vdash \xi \quad \emptyset \vdash V : \alpha}{\mathbb{\Gamma} + x : \alpha \Vdash \xi \cdot x : V} \qquad \frac{\mathbb{\Gamma} \Vdash \xi \quad \mathbb{\Gamma} \Vdash \mathbb{\Gamma}_d}{\mathbb{\Gamma} + \delta \Vdash \xi \cdot \delta : \mathbb{\Gamma}_d}$$

It is important to note, in a model $\xi^{\mathbb{\Gamma}}$, that all variables in $\mathsf{dom}(\mathbb{\Gamma})$ are mapped in the model. This definition of typing ensures that for any model $\xi^{\mathbb{\Gamma}+\delta+x:\alpha}$ then $x \notin \mathsf{dom}(\xi(\delta))$ which guarantees that the order of the LTC is "maintained" in the model.

In a similar fashion to the interpretation of expressions by a model in the λ-logic (Def. 13), the LTCs in the ν-logic are interpreted by a model.

Definition 67 (Interpretation of LTCs). *The* interpretation *of an LTC, $\mathbb{\Gamma}_0$ in a model $\xi^{\mathbb{\Gamma}}$, written $[\![\mathbb{\Gamma}_0]\!]_\xi$, outputs an (unordered) STC which is used to type programs. This is defined as follows, assuming $\mathbb{\Gamma} \Vdash \mathbb{\Gamma}_0$.*

$$\begin{aligned}
[\![\emptyset]\!]_\xi &\stackrel{\text{def}}{=} \emptyset \\
[\![x:\alpha]\!]_\xi &\stackrel{\text{def}}{=} x:\alpha \\
[\![\mathbb{\Gamma}_0 + x:\alpha]\!]_\xi &\stackrel{\text{def}}{=} [\![\mathbb{\Gamma}_0]\!]_\xi, x:\alpha \\
[\![\mathbb{\Gamma}_0 + \delta:\mathbb{TC}]\!]_\xi &\stackrel{\text{def}}{=} [\![\mathbb{\Gamma}_0]\!]_\xi \cup [\![\xi(\delta)]\!]_\xi
\end{aligned}$$

Interpretations of LTCs produce STCs which can be used to type terms, hence the typing judgment $[\![\mathbb{\Gamma}]\!]_\xi \vdash M : \alpha$ is often used.

Given a particular LTC and a model, it is desirable to consider which values can be derived from these without the external introduction of names, in particular which names are present and from those which are reachable or hidden. Consider the model $\xi_1 \equiv x : \lambda z.z = \mathsf{n}$, and the values which can be derived from ξ_1. Clearly the derived values from ξ_1 cannot contain the value n directly, as any use of x will never return n. However, it is possible that a value $\lambda f.(f\mathsf{n}' = (\lambda z.z = \mathsf{n})\mathsf{n}')$ is derived from ξ_1. This value uses n but only under the $\lambda z.z = \cdot$, i.e. the manner in which it occurs in ξ_1. It is precisely for this reason the concept of derivation is introduced. A more complicated example is the model $\xi_2 \equiv x : \lambda z.\mathsf{if}\ z = \mathsf{n}\ \mathsf{then}\ \mathsf{n}_1\ \mathsf{else}\ \mathsf{n}_2 \cdot y : V_y^{\mathsf{Nm}}$, which has two possibilities when it comes to the use of name n_1. If $V_y = \mathsf{n}$ then clearly the term xy evaluates to n_1, however if $V_y \neq \mathsf{n}$ then clearly no combination of x and y can possibly return n_1 and hence n_1 is hidden. Hence it is the combination of the LTC and the model which allows us to define derivation of a term as follows.

Definition 68 (Value derived from an LTC and model)**.** *Deriving a value V from a term M typed by an LTC $\mathbb{\Gamma}$ and model ξ is written $M \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V$*

$$M \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V \stackrel{\text{def}}{=} \mathfrak{a}(M) = \emptyset \ \wedge\ [\![\mathbb{\Gamma}]\!]_\xi \vdash M : \alpha \ \wedge\ (\mathfrak{a}(\xi), M\xi) \Downarrow (\mathfrak{a}(\xi), G',\ V)$$

The values which an LTC can derive are called the reach *of the LTC in a given model.*

Derivations of values from LTCs take inspiration from programs which cannot access names without prior generation and using only the variables and names which are accessible to them and not those which are hidden. For example, $\langle M_0, \mathsf{let}\ x = M_x\ \mathsf{in}\ \mathsf{let}\ y = M_y\ \mathsf{in}\ M\rangle$ can use the M_x and M_y using x and y in M, but cannot use the M_0 in M, as there is no connection between the two elements of the pair. This idea is extended to models in the next definition. Consider the program $\mathsf{let}\ x = M_x\ \mathsf{in}\ \mathsf{let}\ y = M_y\ \mathsf{in}\ M$, where $\mathsf{let}\ y = M_y\ \mathsf{in}\ M$ may use x in the term, then consider that M may now use both x and y, where now it is clear that M_y is derived from x.

Definition 69 (Singleton model extension)**.** *The model ξ' is a single point extension to the model $\xi^{\mathbb{\Gamma}}$, written $\xi \preccurlyeq \xi'$, if it is an addition to ξ at a single: variable mapped to a value derived from ξ, or TCV mapped to an LTC, which is a subset or equal to $\mathbb{\Gamma}$. Formally $\xi^{\mathbb{\Gamma}} \preccurlyeq \xi'$ is defined as follows.*

$$\begin{aligned}\xi^{\mathbb{\Gamma}} \preccurlyeq \xi' \stackrel{\text{def}}{=}\quad & \exists\, y, M^{\alpha}_{y}.\ M_y \overset{[\mathbb{\Gamma},\xi]}{\rightsquigarrow} V_y \ \wedge\ \xi'^{\mathbb{\Gamma}+y:\alpha} \equiv \xi \cdot y : V_y \\ & \vee\ \exists\, \delta, \mathbb{\Gamma}_0.\ \mathbb{\Gamma} \Vdash \mathbb{\Gamma}_0 \ \wedge\ \mathbb{\Gamma}_0 \equiv \mathbb{\Gamma}_0 \backslash_{-TCV} \ \wedge\ \xi' \equiv \xi \cdot \delta : \mathbb{\Gamma}_0\end{aligned}$$

In the definition above, the first case ensures that values are derived from the previous model, ensuring that no names which appear hidden in ξ appear reachable in ξ'. The latter case ensures the LTC added is an ordered subset of the initial type context meaning no new mappings are introduced and ensures the LTC is closed so cannot contain TCVs. By definition $\delta \notin \mathsf{dom}(\mathbb{\Gamma})$ is guaranteed.

Definition 70 (Model extension)**.** *The transitive reflexive closure of the singleton model extension $\preccurlyeq$ is written $\preccurlyeq^{\star}$ such that the model $\xi_0^{\mathbb{\Gamma}_0}$ is extended to the model $\xi_k^{\mathbb{\Gamma}_k}$ is written $\xi_0 \preccurlyeq^{\star} \xi_k$ and is defined as follows.*

$$\xi_0 \preccurlyeq^{\star} \xi_k \stackrel{\text{def}}{=} \xi_0 \equiv \xi_k \ \vee\ \xi_0 \preccurlyeq \xi_k \ \vee\ \exists\, \xi_{k-1}.\ \xi_0 \preccurlyeq^{\star} \xi_{k-1} \preccurlyeq \xi_k$$

The model ξ is defined as a contraction *of ξ' if $\xi \preccurlyeq^{\star} \xi'$. Contractions are the dual of extensions.*

Although a typed model $\xi^{\mathbb{\Gamma}}$ means the domain of both the model and the LTC are identical, the LTCs used in the logic of the form $\mathbb{\Gamma}_0 + \delta$ are intended to express that δ is any future extension of $\mathbb{\Gamma}_0$. This motivates the next definition which is introduced to construct a model from the LTCs used in the logic.

Definition 71 (Well constructed model)**.** *A model ξ is* constructed by *an LTC $\mathbb{\Gamma}$, written $\mathbb{\Gamma} \triangleright \xi$, if any TCV represents a model extension, formally defined by the rules that follow.*

$$\frac{-}{\emptyset \triangleright \emptyset} \qquad \frac{\mathbb{\Gamma} \triangleright \xi \quad \exists\, M^{\alpha}.M \overset{[\mathbb{\Gamma},\xi]}{\rightsquigarrow} V}{\mathbb{\Gamma}+x : \alpha \triangleright \xi \cdot x : V} \qquad \frac{\exists\, \xi_0.\mathbb{\Gamma} \triangleright \xi_0 \ \wedge\ \xi_0 \preccurlyeq^{\star} \xi^{\mathbb{\Gamma}_2} \ \wedge\ \mathbb{\Gamma}_1 \equiv \mathbb{\Gamma}_2 \backslash_{-TCV}}{\mathbb{\Gamma}+\delta \triangleright \xi \cdot \delta : \mathbb{\Gamma}_1}$$

A model $\xi^{\mathbb{\Gamma}}$ is defined as well constructed *if there exists an LTC $\mathbb{\Gamma}'$ such that $\mathbb{\Gamma}' \triangleright \xi$.*

Well constructed models relate the LTCs used in the logic to a model and ensure that all TCVs in the LTC (from the logic) map to type contexts which represent a future LTC.

Well constructed models ensure all names in values are either fresh, or occur in the form in which they were accessible in the previous model. This restricts the reintroduction of a previously hidden name, yet allows them to be used in their hidden form. Consider

the example above in the context of models: let $\xi_1 \equiv x : \lambda z.\mathsf{if}\ z = \mathsf{n}\ \mathsf{then}\ \mathsf{n}_1\ \mathsf{else}\ \mathsf{n}_2$ then n_1 would become reachable if $y : \mathsf{n}$ were a valid extension to ξ_1 however the requirement of well constructed models ensures that $\xi_1 \not\preccurlyeq^\star \xi_1 \cdot x : \mathsf{n}$ in this case. It is worth noting that even though $\xi_1 \not\preccurlyeq^\star \xi_1 \cdot x : \mathsf{n}$, it is the case that $y : \mathsf{n} \preccurlyeq^\star y : \mathsf{n} \cdot x : \lambda z.\mathsf{if}\ z = \mathsf{n}\ \mathsf{then}\ \mathsf{n}_1\ \mathsf{else}\ \mathsf{n}_2$.

For any model ξ, constructed by an LTC $\mathbb{\Gamma}$, the type of the model always types the LTC which constructed it, hence the following lemma is introduced.

Lemma 72 (LTC model construction implies sub-LTC)**.**

$$\forall\, \mathbb{\Gamma}, \xi^{\mathbb{\Gamma}_d}.\ \mathbb{\Gamma} \triangleright \xi \ \rightarrow\ \mathbb{\Gamma}_d \Vdash \mathbb{\Gamma}$$

Proof. Trivial given the manner in which models are constructed in Def. 71 □

All future models in this thesis are assumed to be well constructed unless otherwise stated.

5.2 Semantics

To interpret the triples in the model it is first necessary to interpret the formulae in the model which in turn requires the interpretation of LTCs defined in Def. 67 and expressions. These are all defined in the following definitions.

The expressions in the ν-logic are a subset of the terms of the ν-calculus in Fig. 3.1. As such, expressions could be interpreted by the model simply by the closure of the expression by the model as in Def. 62 (i.e. $[\![e]\!]_\xi \equiv e\xi$). However, to maintain the style presented in the literature, interpretation of expressions is formally defined below.

Definition 73 (Interpretation of expressions)**.** *The* interpretation *of expression e in a model $\xi^{\mathbb{\Gamma}}$, written $[\![e]\!]_\xi$, is given by the following clauses, assuming $\mathbb{\Gamma} \Vdash e : \alpha$.*

$$\begin{aligned}
[\![c]\!]_\xi &\stackrel{\text{def}}{=} c && c \in \{(), \mathsf{true}, \mathsf{false}\} \\
[\![x]\!]_\xi &\stackrel{\text{def}}{=} \xi(x) \\
[\![\langle e, e'\rangle]\!]_\xi &\stackrel{\text{def}}{=} \langle [\![e]\!]_\xi, [\![e']\!]_\xi\rangle \\
[\![\pi_i(e)]\!]_\xi &\stackrel{\text{def}}{=} \pi_i([\![e]\!]_\xi)
\end{aligned}$$

Formulae are given semantics using the well constructed models based on the semantics of the λ-logic and the new concept of models.

Definition 74 (Semantics of formulae)**.**

The semantics *of a formula A in a well constructed model $\xi^{\mathbb{\Gamma}}$, written $\xi \models A$, is defined inductively on the structure of A as follows. The derived formulae are also defined for convenience.*

- $\xi \models e = e' \stackrel{\text{def}}{=} [\![e]\!]_\xi \cong_\alpha^{\mathring{a}(\xi)} [\![e']\!]_\xi$
- $\xi \models \neg A \stackrel{\text{def}}{=} \xi \not\models A.$
- $\xi \models A \wedge B \stackrel{\text{def}}{=} \xi \models A \ \wedge \ \xi \models B.$
- $\xi \models e \bullet e' = m\{A\} \stackrel{\text{def}}{=} \exists V.\ ee' \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V \ \wedge \ \xi \cdot m : V \models A$
- $\xi^{\mathbb{\Gamma}} \models \forall x^\alpha \in (\mathbb{\Gamma}').A \stackrel{\text{def}}{=} \forall M^\alpha, V^\alpha.\ M \overset{[\mathbb{\Gamma}',\ \xi]}{\rightsquigarrow} V \ \rightarrow \ \xi \cdot x : V \models A$
- $\xi^{\mathbb{\Gamma}} \models \forall \delta.A \stackrel{\text{def}}{=} \forall \xi'^{\mathbb{\Gamma}'}.\ \xi \preccurlyeq^\star \xi' \ \rightarrow \ \xi' \cdot \delta : (\mathbb{\Gamma}' \backslash_{-TCV}) \models A$

Derived semantics:

- $\xi \models x \# \mathbb{\Gamma}_0 \stackrel{\text{def}}{=} \neg \exists M_x.\ M_x \overset{[\mathbb{\Gamma}_0,\ \xi]}{\rightsquigarrow} [\![x]\!]_\xi$
- $\xi^{\mathbb{\Gamma}} \models \exists x^\alpha \in (\mathbb{\Gamma}').A \stackrel{\text{def}}{=} \exists M^\alpha, V^\alpha.\ M \overset{[\mathbb{\Gamma}',\ \xi]}{\rightsquigarrow} V \ \wedge \ \xi \cdot x : V \models A$
- $\xi \models A^{-x}(x)[e/x]_{\mathbb{\Gamma}} \stackrel{\text{def}}{=} x \notin \mathsf{dom}(\xi) \ \wedge \ \exists V.\ e \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V \ \wedge \ \xi \cdot x : V \models A(x)$
- $\xi \models A^{-x}(x)[e/x]_{\mathbb{\Gamma}} \stackrel{\text{def}}{=} x \in \mathsf{dom}(\xi) \wedge \forall x'.x' \notin \mathsf{dom}(\xi) \rightarrow \xi \models A(x')[e/x']_{\mathbb{\Gamma}}$

By definition, all models on the right hand side of Def. 74 are well constructed models, as all additions are closed values or closed LTCs derived from the initial model, and extensions of the model ξ, as defined in Def. 70.

The semantics of the previous definition are discussed in more detail here.

$e = e'$ Uses the contextual congruence from Def. 29, to equate the interpreted expressions using any names in the model, including the hidden names.

$\neg A$ and $A \wedge B$ These are both standard definitions.

$e \bullet e' = m\{A\}$ Given $\mathring{a}(ee') = \emptyset$ (see Def. 20) and $\mathbb{\Gamma} \Vdash ee' : \alpha$ by the typing rules in Fig. 2.16 then $ee' \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V \ \leftrightarrow \ (\mathring{a}(\xi), [\![e]\!]_\xi [\![e']\!]_\xi) \Downarrow (G', V)$ hence this is the standard semantics for evaluation formulae. Termination is guaranteed, hence the existence of such a V is guaranteed, meaning this condition is often dropped.

$\forall x^\alpha \in (\mathbb{\Gamma}').A$ This quantifies over all values of type α which are derived from the LTC $\mathbb{\Gamma}'$ and the model used to satisfy the formula. This definition ensures $\xi \cdot x : V$ is a well constructed model of ξ by definition.

$\forall \delta.A$ This quantifies over all possible extensions of the initial model. The LTC which types the extension is then assigned to the TCV. Models must map TCVs to closed LTCs hence the TCV-mappings are removed prior to assigning the LTC in the model.

The derived semantics:

$x\#\mathbb{\Gamma}_0$ The freshness formula is syntactic sugar for $\forall z^{\mathsf{Nm}} \in (\mathbb{\Gamma}_0).z \neq x$, and the semantics of the two formulae are equivalent.

$\exists x^{\alpha} \in (\mathbb{\Gamma}').A$ This is dual to $\forall x^{\alpha} \in (\mathbb{\Gamma}').A$ i.e. equivalent to $\neg\forall x^{\alpha} \in (\mathbb{\Gamma}').\neg A$ and this can be proven using the semantics using the F.O.L. axioms in the meta-logic.

$A^{-x}(x)[e/x]_{\mathbb{\Gamma}}$ This case splits into two, dependant on whether x occurs in $\mathsf{dom}(\xi)$ or not.
If $x \notin \mathsf{dom}(\xi)$, then evaluating e and assigning the value to the substituting variable must satisfy A.
If $x \in \mathsf{dom}(\xi)$, then the x variable is α-converted to an unused variable then proceed as above.

Finally the semantics of triples can be defined.

Definition 75 (Semantics of triples)**.** *A triple is modelled by a well constructed model $\xi^{\mathbb{\Gamma}}$ as follows.*

$$\xi^{\mathbb{\Gamma}} \models \{A\}\ M :_m \{B\} \quad \stackrel{\text{def}}{=} \quad \xi \models A \ \rightarrow\ \exists V.\ (M \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V \ \wedge\ \xi \cdot m : V \models B)$$

Programs in Hoare triples must be static syntax, meaning for any triple $\{A\}\ M :_u \{B\}$ then $\mathfrak{a}(M) = \emptyset$ and the typing requirements ensure $[\![\mathbb{\Gamma}]\!]_{\xi} \vdash M : \alpha$ hence by Def. 68 then $M \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V \ \leftrightarrow\ (\mathfrak{a}(\xi),\ M\xi) \Downarrow (\mathfrak{a}(\xi), G,\ V)$. All well typed terms are terminating in the ν_{GS}-calculus so the existence of such a value V is guaranteed and is often dropped in the soundness proofs for brevity.

Triples must be well typed by an LTC which can be used to construct a model via Def. 71 and if the triple is satisfied under all such models then this is defined as being a valid triple.

Definition 76 (Valid triple)**.** *Let $\mathbb{\Gamma} \Vdash \{A\}\ M :_m \{B\}$, then the triple $\{A\}\ M :_m \{B\}$ is* valid*, written $\models \{A\}\ M :_m \{B\}$ if the following holds.*

$$\begin{aligned}&(\mathbb{\Gamma} \Vdash \{A\}\ M :_m \{B\}\ \rightarrow)\\ &\quad \models \{A\}\ M :_m \{B\} \ \stackrel{\text{def}}{=}\ \forall\, \mathbb{\Gamma}_0, \xi_0^{\mathbb{\Gamma}_0}.\ \mathbb{\Gamma} \triangleright \xi_0 \ \rightarrow\ \xi_0 \models \{A\}\ M :_m \{B\}\end{aligned}$$

In the definition above, variables occurring in $\mathsf{dom}(\xi_0) - \mathsf{dom}(\mathbb{\Gamma})$ cannot occur directly in the triple, but they may still be used in the semantics of the formula via the $\forall\delta$.-constructor.

5.3 Semantics of Extension Independence and Thinness

The definitions of syntactic extension independence (ExtInd_{Syn}), and syntactic thinness with respect to a variable ($\text{Thin}_{Syn}(x)$), are introduced in Sec. 4.4. Semantic versions of these definitions are now introduced with the intention of representing these properties within the semantics. It will later be required to prove that the syntactic definition implies the semantic definition for each. These properties are clearer when defined using the model as they originate specifically from the requirements of the soundness proofs. The semantics and idea of extension independence and thinness are similar respectively to statelessness and thinness found in the Local-logic in Def. 18 with slight differences to be discussed.

5.3.1 Semantic Extension Independence

To prove the ν-logic sound it is often required that certain formulae that are satisfied by one model are satisfied under any extension or contraction to that model, defined below as *semantic extension independence*.

Definition 77 (Semantic extension-independent formulae)**.** *Formulae that are semantically independent of extensions, written* ExtInd_{Sem}*, are defined as follows.*

$$A\text{-ExtInd}_{Sem} \stackrel{\text{def}}{=} \forall \mathbb{\Gamma}.\ \mathbb{\Gamma} \Vdash A \rightarrow \forall \xi, \xi'.\ \mathbb{\Gamma} \triangleright \xi \wedge \xi \preccurlyeq^{\star} \xi' \rightarrow (\xi \models A \leftrightarrow \xi' \models A)$$

The dual of ExtInd_{Sem} *is that the semantics of the formulae do depend on the extensions.*

The requirement that ξ and ξ' are well constructed models by virtue of being extensions of one another, ensures that if a formula holds for a model, it also holds for all extensions and contractions, assuming the typing holds. This differs from the statelessness property in Def. 18 for the Local-logic which only proves the formula must hold in future states. The contraction part of the definition is required in the soundness of rules such as $[\text{Weak}(x)]_{\nu}$ and $[\text{Weak}(\delta)]_{\nu}$.

Although all formulae used in the reasoning proof are ExtInd_{Sem}, there are formulae which are not. An example of a formula that does not maintain its validity under model extensions/contractions is one which depends on how many names exist in the current model, e.g. $\forall \delta. \exists x^{\mathsf{Nm}} \in (\mathbb{\Gamma} + \delta).(x \# \mathbb{\Gamma} \wedge \neg x \# \mathbb{\Gamma} + \delta)$ for some $\mathbb{\Gamma}$. This example intends to capture the existence of a name at x which is derived from the extension δ but fresh from $\mathbb{\Gamma}$ but not fresh from δ itself. Consider the simple model $\xi' \equiv z : n$ then the following

holds as x can always be instantiated as $z : \mathsf{n}$.

$$\xi' \models \forall\delta.\exists x^{\mathsf{Nm}} \in (\delta).(x\#\emptyset \wedge \neg x\#\delta)$$

However, $\emptyset^{\emptyset} \preccurlyeq^{\star} \xi'$ holds, as do the type checks, but the model fails to satisfy the formula.

$$\emptyset \not\models \forall\delta.\exists x^{\mathsf{Nm}} \in (\delta).(x\#\emptyset \wedge \neg x\#\delta)$$

This is because the instantiation of $\forall\delta.$ via $(utc1)$, with the empty LTC $\emptyset$, results in the formula $\exists x^{\mathsf{Nm}} \in (\emptyset).(x\#\emptyset \wedge \neg x\#\emptyset)$ which is clearly a contradiction.

More complicated examples of non-$\textsc{ExtInd}_{Sem}$ formulae exist, however it is not clear if these formulae can be constructed from the axioms and rules hence the requirement that all formulae are indeed $\textsc{ExtInd}_{Sem}$ in the soundness proof. To prove $\textsc{ExtInd}_{Sem}$ of formulae, a subset of all possible $\textsc{ExtInd}_{Sem}$ formulae are defined in Def. 55, and in Lem. 112 it is proven that these $\textsc{ExtInd}_{Syn}$ formulae are indeed $\textsc{ExtInd}_{Sem}$.

5.3.2 Semantic Thinness with Respect to a Variable

Certain proofs in Chapt. 6 require the ability to remove a variable from the model and still satisfy a formula. In this case it is assumed that the model is well constructed both before and after the variable is removed. The syntactically thin formulae defined in Def. 55 are a subset of all thin formulae if thinness is considered with respect to a specific variable defined as follows.

Definition 78 (Thin formulae with respect to a variable)**.**
Define a formula A, as thin with respect to the variable x, written A-$\textsc{Thin}_{Sem}(x)$, as the potential to remove the variable x from the model and still satisfy A. Formally defined as follows.

$$\forall\, \mathbb{\Gamma}.\ \mathbb{\Gamma}\backslash x \Vdash A \ \wedge\ x^{\alpha} \in \mathsf{dom}(\mathbb{\Gamma})$$
$$\rightarrow$$
$$A\ \textsc{Thin}_{Sem}(x^{\alpha}) \ \leftrightarrow\ \forall\, \xi.\ \mathbb{\Gamma} \triangleright \xi \ \rightarrow\ \xi \models A \ \rightarrow\ \xi\backslash x \models A$$
(Assuming $\xi\backslash x$ is a well constructed model)

Lem. 113 proves syntactic thinness implies thinness from Def. 55 and Def. 78 respectively with respect to the same variable.

5.4 General Lemmas Used in Soundness Proofs

General lemmas are introduced here to simplify the later proofs in Chapt. 6. For brevity, sometime $M \Downarrow V$ is used in place of $(\mathfrak{a}(M),\ M) \Downarrow (G',\ V)$.

5.4.1 Lemmas Regarding Function and Nm-Free Types

The following lemmas are regarding $\alpha_{\text{-}(\mathsf{Nm},\to)}$ (name and function free) types introduced in Def. 57.

The following lemma confirms $\alpha_{\text{-}(\mathsf{Nm},\to)}$ typed values contain no names.

Lemma 79 (Function and name free typed values are name free)**.**

$$\forall\, V\text{-}value.\ \emptyset \vdash V : \alpha_{\text{-}(\mathsf{Nm},\to)} \ \to\ \mathring{a}(V) = \emptyset$$

Proof. Assume some value V such that $\emptyset \vdash V : \alpha_{\text{-}(\mathsf{Nm},\to)}$ and prove that $\mathring{a}(V) = \emptyset$ by induction on the structure of values of type $\alpha_{\text{-}(\mathsf{Nm},\to)}$, as follows.

- $\alpha = \mathsf{Unit}$ implies $V = \mathsf{Unit}$ so this clearly holds.
- $\alpha = \mathsf{Bool}$ implies $V = \mathsf{true}$ or $V = \mathsf{false}$ so this clearly holds.
- $\alpha = \alpha_{\text{-}(\mathsf{Nm},\to)_1} \times \alpha_{\text{-}(\mathsf{Nm},\to)_2}$ then $V = \langle V_1, V_2\rangle$ and by I.H. assuming
  $\forall\, V_1\text{-value}.\emptyset \vdash V_1 : \alpha_{\text{-}(\mathsf{Nm},\to)_1} \ \to\ \mathring{a}(V_1) = \emptyset$
  $\forall\, V_2\text{-value}.\emptyset \vdash V_2 : \alpha_{\text{-}(\mathsf{Nm},\to)_2} \ \to\ \mathring{a}(V_2) = \emptyset$
  then clearly $\mathring{a}(\langle V_1, V_2\rangle) = \mathring{a}(V_1) \cup \mathring{a}(V_2) = \emptyset$

□

The previous lemma implies that the closed function and name free type values are also terms which can be typed by any type context, hence the following lemma. Let $\equiv_{sy}$ mean syntactically equivalent (which by definition implies contextual congruence).

Lemma 80 (Function and name free typed values can be derived equally from any LTC)**.**

$$\forall\, \mathbb{\Gamma}, \xi^{\mathbb{\Gamma}}, \mathbb{\Gamma}_1, V^{\alpha\text{-}(\mathsf{Nm},\to)}.\ \mathbb{\Gamma} \Vdash \mathbb{\Gamma}_1 \ \to\ (\exists\, M_0^{\alpha\text{-}(\mathsf{Nm},\to)}.\ M_0 \overset{[\emptyset,\ \xi]}{\rightsquigarrow} V \ \leftrightarrow\ \exists\, M_1^{\alpha\text{-}(\mathsf{Nm},\to)}.\ M_1 \overset{[\mathbb{\Gamma}_1,\ \xi]}{\rightsquigarrow} V)$$

Proof. Holds as V is always a suitable name free closed term i.e. $M_0 \equiv_{sy} V \equiv_{sy} M_1$ always holds as $\mathring{a}(V) = \emptyset \ \wedge\ \emptyset \vdash V : \alpha_{\text{-}(\mathsf{Nm},\to)}$. □

From the previous lemma it is clear that when deriving a value from an LTC, that function and name free typed values can be included or excluded from an LTC as the variable in the term can be replaced by the function and name free typed value mapped by the model to produce the same value.

Lemma 81 (Function and name free typed values don't extend reach of an LTC)**.**

$$\forall\, \mathbb{\Gamma}_0, \xi^{\mathbb{\Gamma}_0}, \mathsf{n}.\ \mathbb{\Gamma}_0 \Vdash \mathbb{\Gamma}{+}y : \alpha_{\text{-}(\mathsf{Nm},\to)} \ \to\ (\exists\, M^{\mathsf{Nm}}.M \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} \mathsf{n} \ \leftrightarrow\ \exists\, M^{\mathsf{Nm}}.M \overset{[\mathbb{\Gamma}+y:\alpha\text{-}(\mathsf{Nm},\to),\ \xi]}{\rightsquigarrow} \mathsf{n})$$

Proof. Clearly holds from Lem. 79 and Lem. 80 so no names are added to the LTC. □

Lemma 82 (Function and name free typed values can be added and removed from the TCV mappings and model equivalent formulae)**.**

$$\forall\, \mathbb{\Gamma},\ \xi^{\mathbb{\Gamma}},\ A.\ \mathbb{\Gamma}(x) = \alpha_{\text{-}(\mathsf{Nm},\to)} \ \rightarrow\ (\xi \cdot \delta : \mathbb{\Gamma}' \models A \ \leftrightarrow\ \xi \cdot \delta : (\mathbb{\Gamma}' \backslash x) \models A)$$

Proof. Assume some $\mathbb{\Gamma}$, $\xi^{\mathbb{\Gamma}}$, A such that $x \in \mathsf{dom}(\mathbb{\Gamma})$ and $\mathbb{\Gamma}(x) = \alpha_{\text{-}(\mathsf{Nm},\to)}$ and prove $(\xi \cdot \delta : \mathbb{\Gamma}' \models A \ \leftrightarrow\ \xi \cdot \delta : (\mathbb{\Gamma}' \backslash x) \models A)$ by induction on the structure of A considering the possible occurrences of TCVs in formulae as follows.

The only two occurrences of δ possible in assertions are in $\mathbb{\Gamma}_0$ in the constructors $\forall x \in (\mathbb{\Gamma}_0).$ and $x \# \mathbb{\Gamma}_0$.

If A contains $\forall x \in (\mathbb{\Gamma}_0).A'$ and $\mathbb{\Gamma}_0$ contains δ then it is unaffected by the addition (or removal) of $x : \alpha_{\text{-}(\mathsf{Nm},\to)}$ as $M \overset{[\mathbb{\Gamma}_0,\ \xi]}{\rightsquigarrow} V \leftrightarrow M \overset{[\mathbb{\Gamma}_0 + y:\alpha_{\text{-}(\mathsf{Nm},\to)},\ \xi]}{\rightsquigarrow} V$ from Lem. 81

If A contains $x \# \mathbb{\Gamma}_0$ and $\mathbb{\Gamma}_0$ contains δ then it is unaffected by the addition (or removal) of $x : \alpha_{\text{-}(\mathsf{Nm},\to)}$ as $M \overset{[\mathbb{\Gamma}_0,\ \xi]}{\rightsquigarrow} V \leftrightarrow M \overset{[\mathbb{\Gamma}_0 + y:\alpha_{\text{-}(\mathsf{Nm},\to)},\ \xi]}{\rightsquigarrow} V$ from Lem. 81.

Any occurrence of $\forall \delta'.A'$ in A extends some model equivalently if x is present in δ' or not, and all other cases are trivial, hence the Lemma holds. □

Lemma 83 (Function and name free typed values can be added or removed and maintain extensions)**.**

$$\forall\, \mathbb{\Gamma},\ \xi^{\mathbb{\Gamma}},\ V_m^{\alpha\text{-}(\mathsf{Nm},\to)},\ \mathbb{\Gamma}', \xi'^{\mathbb{\Gamma}'}.\ \xi \preccurlyeq^{\star} \xi' \ \leftrightarrow\ \xi \cdot m : V_m \preccurlyeq^{\star} \xi' \cdot m : V_m$$

Proof. Assume some $\mathbb{\Gamma}$, $\xi^{\mathbb{\Gamma}}$, $V_m^{\alpha\text{-}(\mathsf{Nm},\to)}$, $\mathbb{\Gamma}'$, $\xi'^{\mathbb{\Gamma}'}$ and prove that $\xi \preccurlyeq^{\star} \xi' \ \leftrightarrow\ \xi \cdot m : V_m \preccurlyeq^{\star} \xi' \cdot m : V_m$ by proving the two directions individually as follows.

$\rightarrow$: Clearly holds given Lem. 80 and Lem. 82 ensure V_m is name free and does not affect the derivation of terms in the extension.

$\leftarrow$: The proof by induction on the structure of $\preccurlyeq^{\star}$ is satisfied by the following cases.

If $\xi \equiv \xi'$ then this clearly holds.

Lem. 80 ensures any value derived from $\mathbb{\Gamma}_x$, ξ_x can equally be derived from $\mathbb{\Gamma}_x \backslash m$, $\xi_x \backslash m$ for any LTC $\mathbb{\Gamma}_x$ and model ξ_x.

Lem. 82 ensures all TCV mapping extensions can add/remove function and name free typed values with equivalent results.

□

The next lemma is used to prove axiom $(f3)$ which is restricted to only functions of type $\alpha \to \alpha_{\text{-}(\mathsf{Nm},\to)}$ where α can be any type. A more general axiom than $(f3)$ may exist,

but a use hasn't been required yet. The lemma ensures that any function of this type can never reveal any names in the function in any context which does not contain the name itself.

Lemma 84 (Functions that map to function and name free types cannot reveal names in that function)**.**

$$\forall\, M^{\mathsf{Nm}}, V^{(\alpha\to\alpha_{\text{-}(\mathsf{Nm},\to)})}, \mathsf{n}_x^{\mathsf{Nm}}.\ (G \equiv \mathring{a}(M) \cup \mathring{a}(V)) \left(\begin{array}{l} \mathsf{n}_x \notin \mathring{a}(M) \\ \wedge\ f : \alpha \to \alpha_{\text{-}(\mathsf{Nm},\to)} \vdash M : \mathsf{Nm} \\ \wedge\ \mathsf{n}_x \in \mathring{a}(V) \\ \wedge\ \emptyset \vdash V : (\alpha \to \alpha_{\text{-}(\mathsf{Nm},\to)}) \end{array}\right) \to \neg\,(G,\ M[V/f]) \Downarrow (G, G',\ \mathsf{n}_x)$$

Proof. Assuming some M^{Nm}, $V^{(\alpha\to\alpha_{\text{-}(\mathsf{Nm},\to)})}$, and some $\mathsf{n}_x^{\mathsf{Nm}}$ such that $\mathsf{n}_x \notin \mathring{a}(M)$ and $f : \alpha \to \alpha_{\text{-}(\mathsf{Nm},\to)} \vdash M : \mathsf{Nm}$ and $\mathsf{n}_x \in \mathring{a}(V)$ and $\emptyset \vdash V : (\alpha \to \alpha_{\text{-}(\mathsf{Nm},\to)})$. Then prove that $\neg\,(G,\ M[V/f]) \Downarrow (G, G',\ \mathsf{n}_x)$ by contradiction by first assuming a smallest such term M for which this holds, then showing a smaller case must exist as follows.

Given $\mathsf{n}_x \notin \mathring{a}(M)$ then it is clear $M \neq \mathsf{n}_x$ and given $\emptyset \vdash V : (\alpha \to \alpha_{\text{-}(\mathsf{Nm},\to)})$ then it is clear $V \neq \mathsf{n}_x$, hence n_x must be derived from terms M and V. Assume there exists at least one such M for which this holds, then take the smallest one M, then by definition there exists an M_k such that

$(G,\ M[V/f]) \Downarrow (G, G',\ \mathsf{n}_x) \leftrightarrow (G,\ M[V/f]) \to^* (G, G',\ M_k) \to (G, G',\ \mathsf{n}_x)$

Then each possible case of M_k is accounted for as follows:

- $M_k \equiv \pi_1(\langle \mathsf{n}_x, V' \rangle)$: hence $M[V/f] \equiv \mathcal{E}[\pi_1(\langle M_1, M_2 \rangle)][V/f]$ where $\mathcal{E}[M_2][V/f] \Downarrow \mathsf{n}_x$ so this $\mathcal{E}[M_2]$ is smaller than M hence contradiction.
- $M_k \equiv \pi_2(\langle V', \mathsf{n}_x \rangle)$: hence $M[V/f] \equiv \mathcal{E}[\pi_2(\langle M_1, M_2 \rangle)][V/f]$ where $\mathcal{E}[M_2][V/f] \Downarrow \mathsf{n}_x$ so this $\mathcal{E}[M_2]$ is smaller than M hence contradiction.
- $M_k \equiv \mathsf{if\ true\ then\ n}_x\ \mathsf{else}\ V'$: hence $M[V/f] \equiv \mathcal{E}[\mathsf{if}\ M_b\ \mathsf{then}\ M_1\ \mathsf{else}\ M_2][V/f]$ where $\mathcal{E}[M_1][V/f] \Downarrow \mathsf{n}_x$ so this $\mathcal{E}[M_1]$ is smaller than M hence contradiction.
- $M_k \equiv \mathsf{if\ false\ then}\ V'\ \mathsf{else\ n}_x$: hence $M[V/f] \equiv \mathcal{E}[\mathsf{if}\ M_b\ \mathsf{then}\ M_1\ \mathsf{else}\ M_2][V/f]$ where $\mathcal{E}[M_2][V/f] \Downarrow \mathsf{n}_x$ so this $\mathcal{E}[M_2]$ is smaller than M hence contradiction.
- $M_k \equiv (\lambda a.\mathsf{n}_x)V'$: hence $M[V/f] \equiv \mathcal{E}[(\lambda a.M_1)M_2][V/f]$ where $\mathcal{E}[M_1][V/f] \Downarrow \mathsf{n}_x$ so this $\mathcal{E}[M_1]$ is smaller than M hence contradiction.
- $M_k \equiv (\lambda a.a)\mathsf{n}_x$: hence $M[V/f] \equiv \mathcal{E}[(\lambda a.M_1)M_2][V/f]$ where $\mathcal{E}[M_2][V/f] \Downarrow \mathsf{n}_x$ so this $\mathcal{E}[M_2]$ is smaller than M hence contradiction.

– $M_k \equiv \mathsf{let}\ x = V'\ \mathsf{in}\ M$: Similar for $\mathsf{let}\ x = V'\ \mathsf{in}\ M$

– There are no other possible terms that reduce to n_x.

– $M_k \equiv \mathsf{gensym}()$: fails to produce n_x as $\mathsf{n}_x \in G$

i.e. Assuming a smallest M such that $M \Downarrow n_x$ (which is assumed to exist), for each term M_k which is just one $\rightarrow$-step away from n_x then it can be proven that a smaller M can be found which produces n_x hence a contradiction exists. □

5.4.2 Lemmas Regarding Nm-Free Types

The following lemmas will be defined for types which are $\alpha_{\text{-Nm}}$. These represent the STLC types although there are ν_{GS}-calculus programs that are of this $\alpha_{\text{-Nm}}$ type which do contain names e.g. $\lambda x.(\mathsf{n} = \mathsf{n})$. It is shown that $\alpha_{\text{-Nm}}$ typed terms can be equated to a name-free term i.e. a STLC term. One of the key assumptions to make the proof simple is that the initial STLC is simple enough to not contain infinite values of a single type i.e. there are no integers and no recursion, however proving these harder extensions would be harder.

To prove a finite number of values of any particular $\alpha_{\text{-Nm}}$ type (up to equivalence), it is first proven that it is possible to define when two functions of $\alpha_{\text{-Nm}}$ type are equal, this is essentially through comparing each of the finite values of the input type.

Definition 85 (The equating formula)**.** *For $\alpha \in \alpha_{\text{-Nm}}$, inductively define $\mathsf{EQ}^{\alpha}(M, N)$ on the type α as the program that equates two functions of type α as follows.*

$$
\begin{array}{rcl}
\mathsf{EQ}^{\mathsf{Unit}}(M,N) & \stackrel{\text{def}}{=} & \mathsf{true} \\
\mathsf{EQ}^{\mathsf{Bool}}(M,N) & \stackrel{\text{def}}{=} & M = N \\
\mathsf{EQ}^{\alpha\to\mathsf{Unit}}(M,N) & \stackrel{\text{def}}{=} & \mathsf{true} \\
\mathsf{EQ}^{\alpha_1\times\alpha_2}(M,N) & \stackrel{\text{def}}{=} & \mathsf{if}\ \neg\mathsf{EQ}^{\alpha_1}(\pi_1(M),\pi_1(N))\ \mathsf{then\ false\ else} \\
 & & \mathsf{if}\ \neg\mathsf{EQ}^{\alpha_2}(\pi_2(M),\pi_2(N))\ \mathsf{then\ false\ else\ true} \\
\mathsf{EQ}^{\alpha_1\to\alpha_2}(M,N) & \stackrel{\text{def}}{=} & \mathsf{if}\ \neg\mathsf{EQ}^{\alpha_2}(M\tilde{V}_0,N\tilde{V}_0)\ \mathsf{then\ false\ else} \\
 & & \mathsf{if}\ \neg\mathsf{EQ}^{\alpha_2}(M\tilde{V}_1,N\tilde{V}_1)\ \mathsf{then\ false\ else} \\
 & & \ldots \\
 & & \mathsf{if}\ \neg\mathsf{EQ}^{\alpha_2}(M\tilde{V}_k,N\tilde{V}_k)\ \mathsf{then\ false\ else\ true} \\
 & & \textit{s.t. } \tilde{V}_k \textit{ are all finite-k number of values of type } \alpha_1 \textit{ (Lem. 86)}
\end{array}
$$

Note the program $\neg M \stackrel{\text{def}}{=} \mathsf{if}\ M\ \mathsf{then\ false\ else\ true}$ *is used for brevity.*

In the following lemmas M_ν represents standard ν_{GS}-calculus terms and M_λ represents a ν_{GS}-calculus term constructed only from STLC terms (i.e. no names and no gensym). The let $x = M$ in N term constructor is ignored for simplicity.

Lemma 86 (Finite STLC values for each $\alpha_{\text{-Nm}}$ type)**.** *Write a list of STLC values of type* α *as* $\tilde{W}^{\alpha}_{\lambda finite}$.

$$\forall\, \alpha \in \alpha_{\text{-Nm}}.\ \exists\, \tilde{W}^{\alpha}_{\lambda finite}.\ \forall\, M^{\alpha}_{\lambda}.\ \exists\, V^{\alpha}_{\lambda} \in \tilde{W}_{\lambda}.M_{\lambda} \cong V_{\lambda}$$

Proof. Assume some $\alpha \in \alpha_{\text{-Nm}}$, then it must be shown that there exists a finite list of STLC values of that type, written $\tilde{W}^{\alpha}_{\lambda\text{finite}}$. Then prove for any ν-term there exists an equivalent value in $\tilde{W}^{\alpha}_{\lambda\text{finite}}$. This is proven by induction on the structure of α and by creating a complete list of values of type α written $\tilde{W}^{\alpha} \equiv V[\alpha]$ as follows.

- $\alpha \equiv \mathsf{Unit}$ then this clearly holds as $(G,\ M) \Downarrow (G',\ ())$ must always hold and so $V[\mathsf{Unit}] \equiv ()$ holds.
- $\alpha \equiv \mathsf{Bool}$ then this clearly holds as $(G,\ M) \Downarrow (G',\ \mathsf{true})$ must always hold or $(G,\ M) \Downarrow (G',\ \mathsf{false})$ must always hold and so $V[\mathsf{Bool}] \equiv \mathsf{true} \mid \mathsf{false}$ holds.
- $\alpha \equiv \alpha_1 \times \alpha_2$ then clearly $V[\alpha_1 \times \alpha_1] \equiv \langle V[\alpha_1], V[\alpha_2] \rangle$ where the right hand side $V[\alpha_1]$ represents every possible value of type α_1, hence the right hand side is the list of every possible combination between $V[\alpha_1]$ and $V[\alpha_2]$.
- $\alpha \equiv \alpha_1 \to \alpha_2$ then by induction on α_1 it can be assumed there are finite values of this type i.e. let $\tilde{W} \equiv V[\alpha_1]$, this is used to state the values of type $\alpha_1 \to \alpha_2$ as

$$\begin{aligned} V[\alpha_1 \to \alpha_2] \equiv \lambda x^{\alpha_1}.\ & \mathsf{if}\ \mathsf{EQ}^{\alpha_1}(x, \tilde{W}_0)\ \mathsf{then}\ V[\alpha_2]\ \mathsf{else} \\ & \mathsf{if}\ \mathsf{EQ}^{\alpha_1}(x, \tilde{W}_1)\ \mathsf{then}\ V[\alpha_2]\ \mathsf{else} \\ & \ldots \\ & \mathsf{if}\ \mathsf{EQ}^{\alpha_1}(x, \tilde{W}_k)\ \mathsf{then}\ V[\alpha_2]\ \mathsf{else}\ V[\alpha_2] \end{aligned}$$

  hence the right hand side is the list of every possible combination between $\tilde{W}$, and $V[\alpha_2]$ in each instance.

The number of values grow exponentially with the size of the type but there are always finite number of values for each type. These values cover all possible cases by definition as no other possible inputs or outputs exist to a function of the given type. □

The next lemma proves that for any term of type $\alpha_{\text{-Nm}}$, there is an equivalent term which contains no names and is a subset of the STLC syntax. The proof relies on the

language containing a finite number of values of any particular $\alpha_{\text{-Nm}}$ type as seen in the previous lemma.

Lemma 87 ($\alpha_{\text{-Nm}}$ typed terms are equivalent to a name free STLC term)**.** *Let* $\alpha \in \alpha_{\text{-Nm}}$

$$\forall\, M_\nu^\alpha.\ \exists\, N_\lambda . M_\nu \cong_\alpha^{\mathring{a}(M_\nu)} N_\lambda$$

I.e. for each term in the ν_{GS}*-calculus of a type which in* $\alpha_{\text{-Nm}}$ *then there exists a contextually congruent term constructed only of core STLC constructors.*

Proof. This is proven by induction on the type $\alpha \in \alpha_{\text{-Nm}}$ as follows.
All ν-terms that are of type Unit and Bool are equivalent to a constant of that type.
The case for $\alpha \equiv \alpha_1 \times \alpha_2$ holds trivially by induction on $\pi_1(M_\nu)^{\alpha_1}$ and $\pi_2(M_\nu)^{\alpha_2}$.
The case for $\alpha \equiv \alpha_1 \to \alpha_2$ holds as follows:

1	$I.H.(\alpha) \stackrel{def}{=} \forall\, M^\alpha.\ \exists\, N_\lambda^\alpha.\ M \cong_\alpha^{\mathring{a}(M)} N$	$I.H.(\alpha)$
2	Assume $I.H.(\alpha_1) \wedge I.H.(\alpha_2)$	*for* α_1, α_2 *in* $\alpha_{\text{-Nm}}$
3	Prove: $I.H.(\alpha_1 \to \alpha_2)$	*for* $\alpha_1 \to \alpha_2$ *in* $\alpha_{\text{-Nm}}$
4	$\forall\, \alpha.\ \exists\, \tilde{W}^\alpha_{\text{finite}}.\ \forall\, M^\alpha_\lambda.\ \exists\, V^\alpha \in \tilde{W}.M \cong V$	$\alpha \in \alpha_{\text{-Nm}}$ *Lem. 86*
5	$I.H.(\alpha_1)$ implies for each ν-term of type α_1 there exists an equivalent λ-term(which must mean there are finite ones of these). Let $\tilde{W}$ be this term	$I.H.(\alpha_1)$
6	$\forall\, M^{\alpha_1}_{1\nu}.\ \exists\, N^{\alpha_1}_{1\lambda}.\ M_{1\nu} \cong_{\alpha_1}^{\mathring{a}(M_{1\nu})} N_{1\lambda}$	$I.H.(\alpha_1)$
7	For each value in $\tilde{W}$, then $M\tilde{W}_i$ is a term of type α_2 which by $I.H.(\alpha_2)$ implies there is an equivalent λ-term. Let $\tilde{U}_i$ be this term i.e. $M\tilde{V}_i \cong \tilde{U}_i$	$I.H.(\alpha_2)$
8	$\forall\, \tilde{W}_i^{\alpha_1} \in \tilde{W}^{\alpha_1}.\ \forall\, M_\nu^{\alpha_1 \to \alpha_2}.\ \exists\, N_\lambda^{\alpha_2}.\ M\tilde{W}_i \cong_{\alpha_2}^{\mathring{a}(M)} N$	$I.H.(\alpha_2)$
9	Using lines 5-8 an equivalent formula to M can be constructed by brute force such that for each input case there is an equivalent output case i.e. $N_\lambda \equiv \lambda x^{\alpha_1}.$ if $\mathsf{EQ}^{\alpha_1}(x, \tilde{V}_0)$ then $\tilde{U}_0$ else ... if $\mathsf{EQ}^{\alpha_1}(x, \tilde{V}_{k-1})$ then $\tilde{U}_{k-1}$ else $\tilde{U}_k$	
10	By definition $M_\nu \cong_{\alpha 1 \to \alpha_2}^{\mathring{a}(M_\nu)} N_\lambda$ as any use of N_λ behaves identically to M_ν in any application it is used in.	

□

5.4.3 Lemmas Regarding Expressions

As discussed in Sec. 5.2, the following lemmas treat expressions as terms in the language. Each expression constructor, c, x, $\pi_i(e)$, and $\langle e, e\rangle$ is also a syntactic constructor in the language, hence $[\![e]\!]_\xi \equiv e\xi$ (closure) implies the expression e is treated as a term.

The next lemma guarantees that no new names are ever generated by evaluation of expressions (represented by the lack of new names in the final configuration). Expressions contain no gensym and no application of functions, hence the subset of the language is guaranteed to produce no fresh names.

Lemma 88 (Expressions cannot create new names)**.**

$$\forall\, \mathbb{\Gamma}, \xi^{\mathbb{\Gamma}}, e.\ \mathbb{\Gamma} \Vdash e : \alpha \ \rightarrow\ \exists\, V.\ (\mathring{a}(\xi),\ [\![e]\!]_\xi) \Downarrow (\mathring{a}(\xi),\ V)$$

Clearly guaranteed termination implies $\exists\, V'.(\mathring{a}(\xi),\ [\![e]\!]_\xi) \Downarrow (\mathring{a}(\xi), G',\ V')$*, however this lemma proves that no new names are produced in such an evaluation.*

Proof. Trivial by induction on the structure of e noting that gensym cannot occur naturally, and can only occur under a λ-binder which cannot be applied. □

The next two lemmas prove that expressions are name free when considered as terms and when evaluating expressions closed by a model then all names in the value produced are contained within the model.

Lemma 89 (Expressions are name free)**.**

$$\forall\, \mathbb{\Gamma}, e.\ \mathbb{\Gamma} \Vdash e : \alpha \ \rightarrow\ \mathring{a}(e) = \emptyset$$

Proof. Assuming some $\mathbb{\Gamma}$ and some e such that $\mathbb{\Gamma} \Vdash e : \alpha$, then $\mathring{a}(e) = \emptyset$ is proven by induction on the structure of e, knowing that $e \in \{c, x, \pi_i(e), \langle e_1, e_2\rangle\}$, i.e. no built-in names. Hence names in e only come from closing with a model via the semantics of $[\![e]\!]_\xi$. □

Lemma 90 (Expressions are fresh name free)**.**

$$\forall\, \mathbb{\Gamma}, \xi^{\mathbb{\Gamma}}, e, V_e.\ \mathbb{\Gamma} \Vdash e : \alpha\ \wedge\ e \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_e\ \rightarrow\ \mathring{a}(V_e) \subseteq \mathring{a}(\xi)$$

Proof. Assuming some $\mathbb{\Gamma}$, $\xi^{\mathbb{\Gamma}}$, e and some V_e such that $\mathbb{\Gamma} \Vdash e : \alpha$ and $e \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_e$ then $\mathring{a}(V_e) \subseteq \mathring{a}(\xi)$ is proven by induction on the structure of e. Given $e \in \{c, x, \pi_i(e), \langle e_1, e_2\rangle\}$ contains no built in names, names only come from closing with a model via the semantics of closure $e\xi$, and hence no evaluation of gensym() can occur, and only old names occur in the value produced. □

The previous lemma ensures that any evaluation of an expression closed by a model will be contextually congruent to the value it evaluated to, due to no fresh names being introduced. This is formalized in the next lemma.

Lemma 91 (Expressions are congruent to their evaluation)**.**

$$\forall\, \mathbb{\Gamma}, \xi^{\mathbb{\Gamma}}, e, V_e. \mathbb{\Gamma} \Vdash e : \alpha \;\wedge\; e \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_e \rightarrow e\xi \cong_{\alpha}^{\mathring{a}(\xi)} V_e$$

Proof. Lem. 90 implies all values V_e only contain names in the model.
Assuming some $\mathbb{\Gamma}$, $\xi^{\mathbb{\Gamma}}$, e and some V_e such that $\mathbb{\Gamma} \Vdash e : \alpha$ and $e \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_e$ then $e\xi \cong_{\alpha}^{\mathring{a}(\xi)} V_e$ can be proven by induction on the structure of e as follows.

$e \equiv c$: clearly holds for $c \in \{(), \mathsf{true}, \mathsf{false}\}$.

$e \equiv x$: clearly holds as $x\xi \equiv \xi(x) \equiv V_e$

$e \equiv \pi_i(e')$: holds as by induction e',
i.e. $\mathbb{\Gamma} \Vdash e' : \alpha_1 \times \alpha_2 \;\wedge\; e' \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} \langle V_1, V_2\rangle \rightarrow e'\xi \cong_{\alpha_1\times\alpha_2}^{\mathring{a}(\xi)} \langle V_1, V_2\rangle$
hence $\pi_i(e')\xi \cong_{\alpha_i}^{\mathring{a}(\xi)} V_i$

$e \equiv \langle e_1, e_2\rangle$: holds by I.H. on e_1 and e_2
i.e. $\mathbb{\Gamma} \Vdash e_1 : \alpha_1 \;\wedge\; e_1 \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_1 \rightarrow e_1\xi \cong_{\alpha_1}^{\mathring{a}(\xi)} V_1$
and $\mathbb{\Gamma} \Vdash e_2 : \alpha_2 \;\wedge\; e_2 \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_2 \rightarrow e_2\xi \cong_{\alpha_2}^{\mathring{a}(\xi)} V_2$
implies $\langle e_1, e_2\rangle\xi \cong_{\alpha_1\times\alpha_2}^{\mathring{a}(\xi)} \langle V_1, V_2\rangle$

□

5.4.4 Lemmas Regarding Derivations

LTCs typing other LTCs can be seen as an ordered subset, as the order is ensured by the typing and the subset is seen when the LTC is interpreted into an STC by a model, as follows.

Lemma 92 (Typed LTC's implies sub-type-contexts)**.**

$$\forall\, \mathbb{\Gamma}, \mathbb{\Gamma}_0, \xi^{\mathbb{\Gamma}}.\ \mathbb{\Gamma} \Vdash \mathbb{\Gamma}_0 \;\rightarrow\; [\![\mathbb{\Gamma}_0]\!]_\xi \subseteq [\![\mathbb{\Gamma}]\!]_\xi$$

Proof: clearly holds through induction on structure of $\mathbb{\Gamma}$ *and* $\mathbb{\Gamma}_0$ *and using the typing rules for LTCs in Fig. 3.2.*

An LTC derived value can always be derived from any extension to the LTC as it will always contain the original LTC, as seen in the following lemma.

Lemma 93 (Values derivable from one LTC are also derivable from an extension of the LTC)**.**

$$\forall\, \mathbb{\Gamma}, \xi^{\mathbb{\Gamma}}, \mathbb{\Gamma}_0 + \mathbb{\Gamma}_1, M, V.\ \mathbb{\Gamma} \Vdash \mathbb{\Gamma}_0 \ \wedge\ \mathbb{\Gamma}_0 \Vdash \mathbb{\Gamma}_1 \ \wedge\ M \overset{[\mathbb{\Gamma}_1,\ \xi]}{\rightsquigarrow} V \ \rightarrow\ M \overset{[\mathbb{\Gamma}_0,\ \xi]}{\rightsquigarrow} V$$

Proof. Assuming some $\mathbb{\Gamma}$, $\xi^{\mathbb{\Gamma}}$, $\mathbb{\Gamma}_0$, $\mathbb{\Gamma}_1$, M, V such that $\mathbb{\Gamma} \Vdash \mathbb{\Gamma}_0$ and $\mathbb{\Gamma}_0 \Vdash \mathbb{\Gamma}_1$ and $M \overset{[\mathbb{\Gamma}_1,\ \xi]}{\rightsquigarrow} V$ then prove $M \overset{[\mathbb{\Gamma}_0,\ \xi]}{\rightsquigarrow} V$ using Lem. 92 i.e. $[\![\mathbb{\Gamma}_1]\!]_\xi \subseteq [\![\mathbb{\Gamma}_1]\!]_\xi$. This then holds trivially as all requirements of LTC derived values are satisfied trivially. □

5.4.5 Lemmas Regarding the Revealing of Names

If a name can be derived from a model but when added to the model it cannot be derived, then this name must be fresh, which is proven in the following lemma.

Lemma 94 (Adding a name to the model but not to the context means it is fresh)**.**

$$\forall\, \mathbb{\Gamma}, \xi^{\mathbb{\Gamma}}, \mathsf{n}_x.\ (\exists\, M_x.M_x \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} \mathsf{n}_x \ \wedge\ \neg\ \exists\, N_x.N_x \overset{[\mathbb{\Gamma},\ \xi\cdot x:\mathsf{n}_x]}{\rightsquigarrow} \mathsf{n}_x) \ \rightarrow\ \mathsf{n}_x \notin \mathring{\mathsf{a}}(\xi)$$

Proof. Assuming some $\mathbb{\Gamma}$, $\xi^{\mathbb{\Gamma}}$ and some $\mathsf{n}_x^{\mathsf{Nm}}$ such that $\exists\, M_x.M_x \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} \mathsf{n}_x$ and $\neg\ \exists\, N_x.N_x \overset{[\mathbb{\Gamma},\ \xi\cdot x:\mathsf{n}_x]}{\rightsquigarrow} \mathsf{n}_x$ then $\mathsf{n}_x \notin \mathring{\mathsf{a}}(\xi)$ is proven by contradiction. Assume $\mathsf{n}_x \in \mathring{\mathsf{a}}(\xi)$ then clearly there would be a direct contradiction in the assumption as $\exists\, N_x.N_x \overset{[\mathbb{\Gamma},\ \xi\cdot x:\mathsf{n}_x]}{\rightsquigarrow} n_x$ as $\mathring{\mathsf{a}}(\xi \cdot x : \mathsf{n}_x) \equiv \mathring{\mathsf{a}}(\xi)$. Hence $\mathsf{n}_x \notin \mathring{\mathsf{a}}(\xi)$. □

Any value derived from an LTC and model cannot reveal previously accessible names from that same LTC and model when added. Hence any name occurring in a model derived from an LTC and the model can also be derived from the LTC and model with the addition of a value derived from said model. This is formally defined by the following lemma.

Lemma 95 (LTC derived terms cannot reveal old names)**.**

$$\forall\, \mathbb{\Gamma}, \xi^{\mathbb{\Gamma}}, \mathbb{\Gamma}_0, M_y^{\alpha_y}, \mathsf{n}^{\mathsf{Nm}}.\ \left(\begin{array}{l} \mathbb{\Gamma} \Vdash \mathbb{\Gamma}_0 \\ \wedge\ M_y \overset{[\mathbb{\Gamma}_0,\ \xi]}{\rightsquigarrow} V_y \\ \wedge\ \mathsf{n} \in \mathring{\mathsf{a}}(\xi) \end{array}\right) \ \rightarrow\ \left(\begin{array}{l} \exists\, M_1.M_1 \overset{[\mathbb{\Gamma}_0 + y,\ \xi\cdot y:V_y]}{\rightsquigarrow} \mathsf{n} \\ \leftrightarrow \\ \exists\, M_0.M_0 \overset{[\mathbb{\Gamma}_0,\ \xi]}{\rightsquigarrow} \mathsf{n} \end{array}\right)$$

Essentially V_y cannot reveal any names in ξ that are not already available to $\mathbb{\Gamma}_0$.

Proof.

1	Proof by contradiction:
2	Assume some $\mathbb{\Gamma}, \xi^{\mathbb{\Gamma}}, \mathbb{\Gamma}_0, M_y^{\alpha_y}, \mathsf{n}^{\mathsf{Nm}}$ with $\mathbb{\Gamma} \Vdash \mathbb{\Gamma}_0 \ \wedge \ M_y \overset{[\mathbb{\Gamma}_0,\ \xi]}{\rightsquigarrow} V_y \ \wedge \ \mathsf{n} \in \mathring{\mathsf{a}}(\xi)$
3	Clearly a contradiction $\neg \ \exists \, M_1.M_1 \overset{[\mathbb{\Gamma}_0 + y,\ \xi \cdot y : V_y]}{\rightsquigarrow} \mathsf{n} \ \wedge \ \exists \, M_0.M_0 \overset{[\mathbb{\Gamma}_0,\ \xi]}{\rightsquigarrow} \mathsf{n}$ *Def. 68*
4	Letting $\mathbb{\Gamma}_{0y} \equiv \mathbb{\Gamma}_0 + y$ and $\xi_y \equiv \xi \cdot y : V_y$ then:
5	Prove the following is a contradiction as follows:
6	$\exists \, \mathsf{n}. \exists \, M_1.M_1 \overset{[\mathbb{\Gamma}_{0y},\ \xi \cdot y : V_y]}{\rightsquigarrow} \mathsf{n} \ \wedge \ \neg \ \exists \, M_0.M_0 \overset{[\mathbb{\Gamma}_0,\ \xi]}{\rightsquigarrow} \mathsf{n}$
7	$\leftrightarrow \ \exists \, \mathsf{n}. \exists \, M_1(y).M_1(y) \overset{[\mathbb{\Gamma}_{0y},\ \xi \cdot y : V_y]}{\rightsquigarrow} \mathsf{n} \ \wedge \ \neg \ \exists \, M_0.M_0 \overset{[\mathbb{\Gamma}_0,\ \xi]}{\rightsquigarrow} \mathsf{n}$
8	$\leftrightarrow \ \exists \, \mathsf{n}. \exists \, M_1(y).M_1(y) \overset{[\mathbb{\Gamma}_{0y},\ \xi \cdot y : V_y]}{\rightsquigarrow} \mathsf{n} \ \wedge \ \neg \ \exists \, M_0 \equiv \mathsf{let}\ y = M_y\ \mathsf{in}\ M_1(y).M_0 \overset{[\mathbb{\Gamma}_0,\ \xi]}{\rightsquigarrow} \mathsf{n}$
9	Contradiction given the name derived by $(M_1(y))\xi_y$ can also be derived by $(\mathsf{let}\ y = M_y\ \mathsf{in}\ M_1(y))\xi$ given the semantics of the evaluation of $(\mathsf{let}\ y = M_y\ \mathsf{in}\ M_1(y))\xi \rightarrow (\mathsf{let}\ y = V_y\ \mathsf{in}\ M_1(y))\xi \rightarrow M_1(y)(\xi \cdot y : V_y)$. As $M_y \overset{[\mathbb{\Gamma}_0,\ \xi]}{\rightsquigarrow} V_y$ then this holds given any fresh names in V_y has no affect on the name produced by M_0 and M_1 as $\mathsf{n} \in \mathring{\mathsf{a}}(\xi)$.
10	This works because $\mathsf{n} \in \mathring{\mathsf{a}}(\xi)$ hence the name cannot be fresh (derived from V_y and any fresh names in V_y can be replicated by a new generation via M_y and will be treated equally.

□

Similar to the previous lemma, any name that occurs in a model can be derived from said model if and only if it can be derived from any extension of said model. This is defined in the next lemma and uses the previous lemma for one of the extension cases.

Lemma 96 (Extensions cannot reveal old names)**.**

$$\forall \, \mathbb{\Gamma}, \xi^{\mathbb{\Gamma}}, \mathbb{\Gamma}', \xi^{\mathbb{\Gamma}'}, \mathsf{n}^{\mathsf{Nm}}. \left(\begin{array}{l} \xi \preccurlyeq^{\star} \xi' \\ \wedge \ \mathsf{n} \in \mathring{\mathsf{a}}(\xi) \end{array} \right) \rightarrow \begin{array}{l} \exists \, M'.M' \overset{[\mathbb{\Gamma}',\ \xi']}{\rightsquigarrow} \mathsf{n} \\ \leftrightarrow \\ \exists \, M.M \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} \mathsf{n} \end{array}$$

Essentially ξ' cannot reveal any names in ξ that are not already available to ξ.

Proof. Assuming some $\mathbb{\Gamma}$, $\xi^{\mathbb{\Gamma}}$, $\mathbb{\Gamma}'$, ξ' and some n^{Nm} such that $\xi \preccurlyeq^{\star} \xi'$ and $\mathsf{n} \in \mathring{\mathsf{a}}(\xi)$ then $\exists \, M'.M' \overset{[\mathbb{\Gamma}',\ \xi']}{\rightsquigarrow} \mathsf{n} \ \leftrightarrow \ \exists \, M.M \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} \mathsf{n}$ holds by proving both directions of the $\leftrightarrow$ individually as follows.

$\leftarrow$: this clearly holds as the same M can be used i.e. $\exists\, M.M \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} \mathsf{n} \rightarrow M \overset{[\mathbb{\Gamma}',\ \xi']}{\rightsquigarrow} \mathsf{n}$

$\rightarrow$: Proof by induction on the structure of ξ':

- $\xi' \equiv \xi$: This clearly holds.
- $\xi' \equiv \xi'_0 \cdot \delta : \mathbb{\Gamma}_0 \backslash_{-TCV}$: then this holds as no new names are reachable by the introduction of δ as $[\![\mathbb{\Gamma}']\!]_{\xi'} \equiv [\![\mathbb{\Gamma}' + \delta]\!]_{\xi' \cdot \delta : \mathbb{\Gamma}'}$ and I.H. on ξ'_0.
- $\xi' \equiv \xi_0^{\mathbb{\Gamma}'_0} \cdot y : V_y$: then $\exists\, M_y.M_y \overset{[\mathbb{\Gamma}'_0,\ \xi'_0]}{\rightsquigarrow} V_y$ and using Lem. 95 and I.H. on ξ'_0 this clearly holds.

□

In a reduction evaluation, if a name occurs in the initial nameset but not in the term then it can never occur in the produced value, as proven in the following lemma.

Lemma 97 (Names fresh in term imply name fresh in value)**.**

$$\forall\, \mathbb{\Gamma}, \xi^{\mathbb{\Gamma}}, M, \mathsf{n}, V.\ \mathsf{n} \notin \mathring{a}(M)\ \wedge\ (\mathring{a}(M), \mathsf{n}, G_0,\ M) \Downarrow (\mathring{a}(\xi), \mathsf{n}, G_0, G',\ V)\ \rightarrow\ \mathsf{n} \notin \mathring{a}(V)$$

Proof. Assume some $\mathbb{\Gamma}$, $\xi^{\mathbb{\Gamma}}$, M, n, V such that $\mathsf{n} \notin \mathring{a}(M)$ and $(\mathring{a}(M), \mathsf{n}, G_0,\ M) \Downarrow (\mathring{a}(\xi), \mathsf{n}, G_0, G',\ V)$ and prove $\mathsf{n} \notin \mathring{a}(V)$ by induction on the structure of M. Most cases are trivial, the only non-trivial case is as follows.

$M \equiv M_1 M_2$:

By I.H. $(G,\ M_1) \Downarrow (G, G_1,\ V_1)$ and $(G, G_1,\ M_2) \Downarrow (G, G_1, G_2,\ V_2)$ with $\mathsf{n} \notin V_1 V_2$ hence 2 cases occur:

$V_1 \equiv \lambda x.M'_1$ then this evaluates to $M'_1[V_2/x]$ which by assumptions and the operational semantics implies $\mathsf{n} \notin M'_1[V_2/x]$, hence by induction $\mathsf{n} \notin \mathring{a}(V)$.

$V_1 \equiv \mathsf{gensym}$ and $V_2 \equiv ()$ then by the operational semantics this generates a fresh name $(\neq \mathsf{n})$ hence $\mathsf{n} \notin \mathring{a}(V)$. □

The previous lemma extends to LTC derived values as the impossibility of deriving a fresh name from a model with the fresh name but the LTC without the name as follows.

Lemma 98 (Fresh names are not derivable from an LTC of a model where the name is added to the model)**.**

$$\forall\, \mathbb{\Gamma}, \xi^{\mathbb{\Gamma}}, \mathsf{n}.\ \mathsf{n} \notin \mathring{a}(\xi)\ \rightarrow\ \neg\ \exists\, M^{\mathsf{Nm}}.M \overset{[\mathbb{\Gamma},\ \xi \cdot m : \mathsf{n}]}{\rightsquigarrow} \mathsf{n}$$

Proof. This holds as n must be a freshly generated name not appearing in ξ. Hence, no term can be derived from the model ξ to generate such a name.

1	Assume $\mathbb{\Gamma}$, $\xi^{\mathbb{\Gamma}}$ then prove by contradiction:	
2	i.e. assume $\neg\,(\mathsf{n} \notin \mathring{a}(\xi) \rightarrow \neg\, \exists\, M^{\mathsf{Nm}}.M \overset{[\mathbb{\Gamma},\ \xi\cdot m:\mathsf{n}]}{\rightsquigarrow} \mathsf{n})$	
3	$\leftrightarrow\ (\mathsf{n} \notin \mathring{a}(\xi) \wedge \exists\, M^{\mathsf{Nm}}.M \overset{[\mathbb{\Gamma},\ \xi\cdot m:\mathsf{n}]}{\rightsquigarrow} \mathsf{n})$	*F.O.L.*
4	$\leftrightarrow\ \mathsf{n} \notin \mathring{a}(\xi) \wedge \exists\, M^{\mathsf{Nm}}.\ \mathring{a}(M) = \emptyset \wedge \mathbb{\Gamma} \vdash M : \alpha$ $\wedge\,(\mathring{a}(\xi), \mathsf{n},\ M(\xi \cdot m : \mathsf{n})) \Downarrow (\mathring{a}(\xi), \mathsf{n}, G',\ \mathsf{n})$	*Sem.* $\overset{[,]}{\rightsquigarrow}$
5	$\leftrightarrow\ \mathsf{n} \notin \mathring{a}(\xi) \wedge \exists\, M^{\mathsf{Nm}}.\ \mathring{a}(M) = \emptyset \wedge \mathbb{\Gamma} \vdash M : \alpha$ $\wedge\,(\mathring{a}(\xi), \mathsf{n},\ M\xi) \Downarrow (\mathring{a}(\xi), \mathsf{n}, G',\ \mathsf{n})$	$m \notin \mathsf{dom}(\mathbb{\Gamma})$ $\wedge\ \mathbb{\Gamma} \vdash M : \alpha$
6	$\rightarrow\ \mathsf{n} \notin \mathring{a}(\xi) \wedge \exists\, M^{\mathsf{Nm}}.\ \mathring{a}(M) = \emptyset \wedge \mathbb{\Gamma} \vdash M : \alpha$ $\wedge\ \mathsf{n} \notin \mathring{a}(M\xi) \wedge \mathsf{n} \notin \mathsf{n}$ $\wedge\,(\mathring{a}(\xi), \mathsf{n},\ M\xi) \Downarrow (\mathring{a}(\xi), \mathsf{n}, G',\ \mathsf{n})$	$\mathsf{n} \notin \mathring{a}(M\xi)$ *and* *Lem. 97* *implies contradiction*
7	$\rightarrow\quad \mathsf{n} \notin \mathring{a}(\xi) \rightarrow \neg\, \exists\, M^{\mathsf{Nm}}.M \overset{[\mathbb{\Gamma},\ \xi\cdot m:\mathsf{n}]}{\rightsquigarrow} \mathsf{n}$	*by contradiction*

□

5.4.6 Lemmas Regarding Model Extensions

The following two lemmas prove interpretations of LTCs and closure of terms are equivalent under model extensions/contractions.

Lemma 99 (Semantics of LTC is equal in model extensions)**.**

$$\forall\, \mathbb{\Gamma}_1, \mathbb{\Gamma}_2, \xi_1^{\mathbb{\Gamma}_1}, \xi_2^{\mathbb{\Gamma}_2}.\ (\xi_1 \preccurlyeq^\star \xi_2 \wedge \mathbb{\Gamma}_1 \Vdash \mathbb{\Gamma}_0) \rightarrow [\![\mathbb{\Gamma}_0]\!]_{\xi_1} = [\![\mathbb{\Gamma}_0]\!]_{\xi_2}$$

Proof. Assume some $\mathbb{\Gamma}_1, \xi_1^{\mathbb{\Gamma}_1}, \mathbb{\Gamma}_2, \xi_2^{\mathbb{\Gamma}_2}$ such that $\xi_1 \preccurlyeq^\star \xi_2$ and $\mathbb{\Gamma}_1 \Vdash \mathbb{\Gamma}_0$ and prove $[\![\mathbb{\Gamma}_0]\!]_{\xi_1} = [\![\mathbb{\Gamma}_0]\!]_{\xi_2}$ by induction on the structure of $\mathbb{\Gamma}_0$ as follows.

$\mathbb{\Gamma}_0 \equiv \emptyset$: Clearly $[\![\emptyset]\!]_{\xi_1} = \emptyset = [\![\emptyset]\!]_{\xi_2}$.

$\mathbb{\Gamma}_0 \equiv \mathbb{\Gamma}_0' + x : \alpha$: By I.H. $[\![\mathbb{\Gamma}_0']\!]_{\xi_1} = [\![\mathbb{\Gamma}_0']\!]_{\xi_2}$ and also by Def.$\preccurlyeq^\star$, $\mathbb{\Gamma}_1(x) = \alpha = \mathbb{\Gamma}_2(x)$.

$\mathbb{\Gamma}_0 \equiv \mathbb{\Gamma}_0' + \delta : \mathbb{TC}$: By I.H. $[\![\mathbb{\Gamma}_0']\!]_{\xi_1} = [\![\mathbb{\Gamma}_0']\!]_{\xi_2}$ and also by Def.$\preccurlyeq^\star$, $\mathbb{\Gamma}_1(\delta) = \mathbb{TC} = \mathbb{\Gamma}_2(\delta)$ and $\xi_1(\delta) = \xi_2(\delta)$ this case holds.

$\mathbb{\Gamma}_0 \equiv \mathbb{\Gamma}_0' + \mathbb{\Gamma}_0''$: by I.H. on both $\mathbb{\Gamma}_0'$ and $\mathbb{\Gamma}_0''$ this clearly holds.

□

A model extension adds variable and TCV mappings to the base model and hence interpretations of expressions are equivalent under both base and extension models.

Lemma 100 (Extensions give equal semantics of expressions)**.**

$$\forall\, \mathbb{\Gamma}_1, \xi_1^{\mathbb{\Gamma}_1}, \mathbb{\Gamma}_2, \xi_2^{\mathbb{\Gamma}_2}, e.\ \xi_1 \preccurlyeq^\star \xi_2 \ \wedge\ \mathbb{\Gamma}_1 \Vdash e : \alpha \ \rightarrow\ [\![e]\!]_{\xi_1} \equiv_{sy} [\![e]\!]_{\xi_2}$$

Proof. Trivial given $\mathbb{\Gamma}_1 \Vdash e : \alpha$ and $\forall\, x \in \mathsf{dom}(\mathbb{\Gamma}_1).\ \xi_1(x) \equiv_{sy} \xi_2(x)$. □

Lemma 101 (Model extensions close terms equally)**.**

$$\forall\, \mathbb{\Gamma}_1, \xi_1^{\mathbb{\Gamma}_1}, \mathbb{\Gamma}_2, \xi_2^{\mathbb{\Gamma}_2}.(\xi_1 \preccurlyeq^\star \xi_2 \ \wedge\ [\![\mathbb{\Gamma}_1]\!]_{\xi_1} \vdash M : \alpha) \ \rightarrow\ M\xi_1 \equiv M\xi_2$$

Proof. Trivial given $[\![\mathbb{\Gamma}_1]\!]_{\xi_1} \Vdash M : \alpha$ and $\forall\, x \in \mathsf{dom}([\![\mathbb{\Gamma}_1]\!]_{\xi_1}).\ \xi_1(x) \equiv_{sy} \xi_2(x)$. □

Derivations are shown to be equivalent under model extensions assuming the same typing LTC as follows.

Lemma 102 (Evaluation under model extensions are equivalent)**.**

$$\begin{aligned}&\forall\, \mathbb{\Gamma}_1, \xi_1^{\mathbb{\Gamma}_1}, \mathbb{\Gamma}_2, \xi_2^{\mathbb{\Gamma}_2}, \mathbb{\Gamma}_0, M, V.\\ &\quad (\xi_1 \preccurlyeq^\star \xi_2 \ \wedge\ \mathbb{\Gamma}_1 \Vdash \mathbb{\Gamma}_0 \ \wedge\ \mathring{a}(V) \cap \mathring{a}(\xi_2) \subseteq \mathring{a}(\xi_1))\\ &\quad \rightarrow\ (M \overset{[\mathbb{\Gamma}_0,\ \xi_1]}{\rightsquigarrow} V \ \leftrightarrow\ M \overset{[\mathbb{\Gamma}_0,\ \xi_2]}{\rightsquigarrow} V)\end{aligned}$$

Proof. Lem. 101 ($M\xi_1 \equiv M\xi_2$) and *Sem.* $\rightarrow$ prove this in both directions of the $\leftrightarrow$ assuming $(\xi_1 \preccurlyeq^\star \xi_2 \ \wedge\ \mathbb{\Gamma}_1 \Vdash \mathbb{\Gamma}_0 \ \wedge\ \mathring{a}(V) \cap \mathring{a}(\xi_2) \subseteq \mathring{a}(\xi_1))$.

1	Assume $\mathbb{\Gamma}_1, \mathbb{\Gamma}_2, \xi_1^{\mathbb{\Gamma}_1}, \xi_2^{\mathbb{\Gamma}_2}, \mathbb{\Gamma}_0$ s.t	
2	$\xi_1 \preccurlyeq^\star \xi_2 \ \wedge\ \mathbb{\Gamma}_1 \Vdash \mathbb{\Gamma}_0 \ \wedge\ M \overset{[\mathbb{\Gamma}_0,\ \xi_1]}{\rightsquigarrow} V \ \wedge\ \mathring{a}(V) \cap \mathring{a}(\xi_2) \subseteq \mathring{a}(\xi_1)$	*Assume*
3	$\xi_1 \preccurlyeq^\star \xi_2 \ \wedge\ \mathbb{\Gamma}_1 \Vdash \mathbb{\Gamma}_0$ $\wedge\ \mathring{a}(M) = \emptyset \ \wedge\ [\![\mathbb{\Gamma}_0]\!]_{\xi_1} \vdash M : \alpha$ $\wedge\ (\mathring{a}(\xi_1), M\xi_1) \Downarrow (\mathring{a}(\xi_1), G',\ V) \ \leftrightarrow\ (\mathring{a}(\xi_1), M\xi_2) \Downarrow (\mathring{a}(\xi_1), G',\ V)$	*Lem. 101* *$(M\xi_1 \equiv M\xi_2)$*
4	$\xi_1 \preccurlyeq^\star \xi_2 \ \wedge\ \mathbb{\Gamma}_1 \Vdash \mathbb{\Gamma}_0$ $\wedge\ \mathring{a}(M) = \emptyset \ \wedge\ [\![\mathbb{\Gamma}_0]\!]_{\xi_1} \vdash M : \alpha$ $\wedge\ (\mathring{a}(\xi_1), M\xi_2) \Downarrow (\mathring{a}(\xi_1), G',\ V) \ \leftrightarrow\ (\mathring{a}(\xi_2), M\xi_2) \Downarrow (\mathring{a}(\xi_2), G',\ V)$	*Sem.* $\rightarrow$ *$(\mathring{a}(V) \cap \mathring{a}(\xi_2) \subseteq \mathring{a}(\xi_1))$*
5	hence: $\forall\, \mathbb{\Gamma}_1, \xi_1^{\mathbb{\Gamma}_1}, \mathbb{\Gamma}_2, \xi_2^{\mathbb{\Gamma}_2}, \mathbb{\Gamma}_0, M.$ $(\xi_1 \preccurlyeq^\star \xi_2 \ \wedge\ \mathbb{\Gamma}_1 \Vdash \mathbb{\Gamma}_0)$ $\rightarrow\ (M \overset{[\mathbb{\Gamma}_0,\ \xi_1]}{\rightsquigarrow} V \ \leftrightarrow\ M \overset{[\mathbb{\Gamma}_0,\ \xi_2]}{\rightsquigarrow} V)$	*Lem. 99,* $[\![\mathbb{\Gamma}_0]\!]_{\xi_1} \equiv [\![\mathbb{\Gamma}_0]\!]_{\xi_2}$ $Sem.^{[,]}_{\rightsquigarrow}$

□

This includes the case where $\xi \preccurlyeq^\star \xi \cdot \delta : \mathbb{\Gamma}_0$ *for any* $\mathbb{\Gamma}_0$.

Extensions are often needed to maintain the extensional property when an LTC derived value is added to both models. This is proven by first proving this for the single step extension in the following lemma, then the reflexive, transitive closure extension in the following lemma after that.

Lemma 103 (Values derived from $\mathbb{\Gamma}$ maintain extension (single) when added to models which are extensions)**.**

$$\forall\, \mathbb{\Gamma}, \xi^{\mathbb{\Gamma}}, \mathbb{\Gamma}', \xi'^{\mathbb{\Gamma}'}, M^\alpha, V.\ \xi \preccurlyeq \xi' \ \wedge\ M \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V \ \wedge\ \mathring{a}(V) \cap \mathring{a}(\xi') \subseteq \mathring{a}(\xi) \ \rightarrow\ \xi \cdot x : V \preccurlyeq \xi' \cdot x : V$$

Proof. This is proven by proving the two cases of single step extension individually as follows.

1	Assume $\mathbb{\Gamma}, \xi^{\mathbb{\Gamma}}, \mathbb{\Gamma}', \xi^{\mathbb{\Gamma}'}, M^\alpha, V$ s.t. $\xi \preccurlyeq \xi' \ \wedge\ M \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V$	
2	The case for $\xi' \equiv \xi \cdot \delta : \mathbb{\Gamma}_0 \backslash_{-TCV}$ is proven below:	
3	assume some $\mathbb{\Gamma}_0$ s.t. $\mathbb{\Gamma} \Vdash \mathbb{\Gamma}_0$ with $\xi \preccurlyeq \xi \cdot \delta : \mathbb{\Gamma}_0 \backslash_{-TCV}$ then $\mathbb{\Gamma} + x \Vdash \mathbb{\Gamma}_0$ and hence $\xi \cdot x : V \preccurlyeq \xi \cdot \delta : \mathbb{\Gamma}_0 \backslash_{-TCV} \cdot x : V$ as adding δ doesn't extend the number of values that can be derived.	*Sem.*$\preccurlyeq$
4	The case for $\xi' \equiv \xi \cdot y : V'$ is proven below:	
5	Assume $(\xi \preccurlyeq \xi' \ \wedge\ M \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V)$	*(Sem.* $\preccurlyeq$ *implies* $\xi \preccurlyeq \xi \cdot x : V$*)*
6	$\rightarrow\ \exists M'.\ M' \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V' \ \wedge\ \xi' \equiv \xi \cdot y' : V' \ \wedge\ M \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V$	*Sem.*$\preccurlyeq$
7	$\rightarrow\ \exists M'.\ M' \overset{[\mathbb{\Gamma},\ \xi \cdot x:V]}{\rightsquigarrow} V' \ \wedge\ \xi' \cdot x : V \equiv \xi \cdot x : V \cdot y' : V'$	*Lem. 102*
8	$\rightarrow\ \exists M'.\ M' \overset{[\mathbb{\Gamma}+x:\alpha_x,\ \xi \cdot x:V]}{\rightsquigarrow} V' \ \wedge\ \xi' \cdot x : V \equiv \xi \cdot x : V \cdot y' : V'$	*Lem. 93*
9	$\rightarrow\ \xi \cdot x : V \preccurlyeq \xi' \cdot x : V$	*Sem.*$\preccurlyeq$

□

If a base model and two extensions of the base model are considered, then Lem. 104 ensures the extensions can be combined and still maintain the extension property under certain conditions. This can be thought of as the diamond property for model extensions as seen in Fig. 5.1.

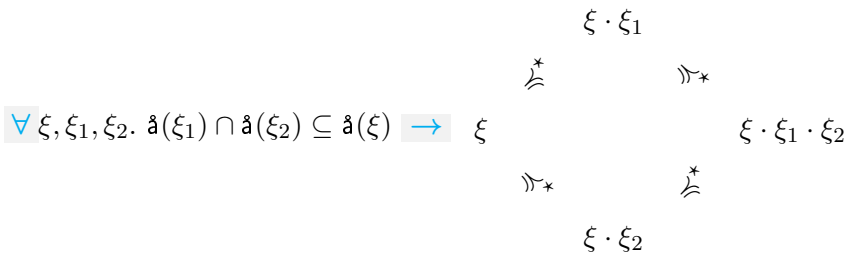

Figure 5.1: Diamond property for model extensions.

Lemma 104 (Two extensions combine to make extensions of each other (diamond property for model extensions))**.**

$$\forall\, \mathbb{\Gamma}, \mathbb{\Gamma}_1, \mathbb{\Gamma}_2, \xi^{\mathbb{\Gamma}}, \xi_1^{\mathbb{\Gamma}_1}, \xi_2^{\mathbb{\Gamma}_2}.$$
$$\xi \preccurlyeq^\star \xi \cdot \xi_1 \ \wedge\ \xi \preccurlyeq^\star \xi \cdot \xi_2 \ \wedge\ \mathring{a}(\xi_1) \cap \mathring{a}(\xi_2) \subseteq \mathring{a}(\xi)$$
$$\rightarrow\ (\xi \cdot \xi_1 \preccurlyeq^\star \xi \cdot \xi_1 \cdot \xi_2 \leftrightarrow \xi \cdot \xi_2 \preccurlyeq^\star \xi \cdot \xi_1 \cdot \xi_2)$$

Proof. By induction on the structure of ξ_1 the following cases hold.

$\xi_1 \equiv \emptyset$ Holds trivially.

$\xi_1 \equiv \delta : \mathbb{\Gamma}_i$ Clearly $\xi \cdot \xi_1 \cdot \xi_2 \preccurlyeq^\star \xi \cdot \xi_1 \cdot \xi_2 \cdot \delta : \mathbb{\Gamma}_i$ holds as $\mathbb{\Gamma}+\mathbb{\Gamma}_1 \Vdash \mathbb{\Gamma}_i$ implies $\mathbb{\Gamma}+\mathbb{\Gamma}_1+\mathbb{\Gamma}_2 \Vdash \mathbb{\Gamma}_i$.

$\xi_1 \equiv x : V$ Holds via Lem. 103.

$\xi_1 \equiv \xi_1' \cdot \xi_1''$ Holds by I.H. on both ξ_1' and ξ_1''.

□

If a TCV does not appear in a formulae then the TCV can be removed or added to the model in any position and still satisfy the formula.

Lemma 105 (Extending the model by δ maintains models)**.** *(A* -ExtInd$_{Syn}$ *not required)*

$$\forall\, \mathbb{\Gamma}, \xi^{\mathbb{\Gamma}}, \mathbb{\Gamma}', \xi'^{\mathbb{\Gamma}'}, \mathbb{\Gamma}_a, A^{-\delta}.\ \mathbb{\Gamma} \Vdash \mathbb{\Gamma}_a \rightarrow (\xi \cdot \xi' \models A^{-\delta} \leftrightarrow \xi \cdot \delta : \mathbb{\Gamma}_a \backslash_{-TCV} \cdot \xi' \models A^{-\delta})$$

where $A^{-\delta}$ means δ does not occur syntactically in A.

Proof. By definition of $\preccurlyeq$, $\xi \preccurlyeq \xi \cdot \delta : \mathbb{\Gamma}_a \backslash_{-TCV}$ holds and implies via Def.$\preccurlyeq^\star$ that $\xi \cdot \xi' \preccurlyeq^\star \xi \cdot \delta : \mathbb{\Gamma}_a \backslash_{-TCV} \cdot \xi'$ which allows the use of Lem. 100 and Lem. 102 ($\mathring{a}(\xi \cdot \xi') \equiv \mathring{a}(\xi \cdot \delta : \mathbb{\Gamma}_a \backslash_{-TCV} \cdot \xi')$).

The proof follows by I.H. on structure of A:

– $e_1 = e_2$ proven by noting that $[\![e_i]\!]_{\xi\cdot\xi'} \equiv [\![e_i]\!]_{\xi\cdot\delta:\mathbb{\Gamma}_a\backslash_{-TCV}\cdot\xi'}$ (Lem. 100).

– $A_1 \wedge A_2$, $A_1 \vee A_2$, $A_1 \to A_2$: I.H. on A_1 and A_2.

– $\neg A_1$, proof by I.H. on A_1, assume $\xi \cdot \delta : \mathbb{\Gamma}_a\backslash_{-TCV} \cdot \xi' \models \neg A_1$ then clearly $\xi \cdot \delta : \mathbb{\Gamma}_a\backslash_{-TCV} \cdot \xi' \not\models A_1$ and by I.H. on A_1 then $\xi \cdot \xi' \not\models A_1$ i.e. $\xi \cdot \xi' \models \neg A_1$.

– $x\#\mathbb{\Gamma}'$ holds by Lem. 100 and Lem. 102 ($\mathring{a}(\xi \cdot \xi') \equiv \mathring{a}(\xi \cdot \delta : \mathbb{\Gamma}_a\backslash_{-TCV} \cdot \xi')$) which ensure $[\![x]\!]_{\xi\cdot\xi'} \equiv [\![x]\!]_{\xi\cdot\delta:\mathbb{\Gamma}_a\backslash_{-TCV}\cdot\xi'}$ and
$\exists\, M_x.\ M_x \overset{[\mathbb{\Gamma}',\ \xi\cdot\xi']}{\rightsquigarrow} V \leftrightarrow \exists\, M_x.\ M_x \overset{[\mathbb{\Gamma}',\ \xi\cdot\delta:\mathbb{\Gamma}_a\backslash_{-TCV}\cdot\xi']}{\rightsquigarrow} V$ respectively, hence clearly
$\neg\, \exists\, M_x.\ M_x \overset{[\mathbb{\Gamma}',\ \xi\cdot\xi']}{\rightsquigarrow} [\![x]\!]_{\xi\cdot\xi'} \leftrightarrow \neg\, \exists\, M_x.\ M_x \overset{[\mathbb{\Gamma}',\ \xi\cdot\delta:\mathbb{\Gamma}_a\backslash_{-TCV}\cdot\xi']}{\rightsquigarrow} [\![x]\!]_{\xi\cdot\delta:\mathbb{\Gamma}_a\backslash_{-TCV}\cdot\xi'}$.

– $e \bullet e' = m\{A_1\}$ by I.H. on A_1 with $[\![e]\!]_{\xi\cdot\xi'} \equiv [\![e]\!]_{\xi\cdot\delta:\mathbb{\Gamma}_a\backslash_{-TCV}\cdot\xi'}$
and $[\![e']\!]_{\xi\cdot\xi'} \equiv [\![e']\!]_{\xi\cdot\delta:\mathbb{\Gamma}_a\backslash_{-TCV}\cdot\xi'}$ (Lem. 100) hence evaluation is equivalent and this holds.

– $\forall x \in (\mathbb{\Gamma}').A_1$ knowing that $\delta \notin \mathbb{\Gamma}'$ then the same possible terms are quantified over given Lem. 102 ($M_x \overset{[\mathbb{\Gamma}',\ \xi\cdot\xi']}{\rightsquigarrow} V \leftrightarrow M_x \overset{[\mathbb{\Gamma}',\ \xi\cdot\delta:\mathbb{\Gamma}_a\backslash_{-TCV}\cdot\xi']}{\rightsquigarrow} V$) and by I.H. on A_1 the case holds.

– $\forall\delta'.A_1$ by I.H. on A_1 this holds for any model containing δ and δ' then by the semantics the lemma holds.

□

5.4.7 Lemmas Regarding Congruent Models

Congruent values should be interchangeable in models, assuming the models are well constructed, hence the introduction of congruent models in Def. 106, ensuring the two models are point-wise contextually congruent. Congruent models are proven to satisfy the exact same formulae in Lem. 108 using Lem. 107 which states that any term used to derive a value from one model produces a contextually congruent value using congruent models. These proofs assume that any fresh name produced by one model can be swapped for the same names as those produced by another model. This is valid under injective-renaming of fresh names (nominal determinacy Def. 25).

Definition 106 (Congruent models)**.** *Two models ξ_1 and ξ_2 are* congruent models *($\xi_1 \cong \xi_2$) iff the values are contextually congruent and the LTC are identical in both models as*

follows.

$$\begin{aligned}
&\forall\, \mathbb{\Gamma}_1, \mathbb{\Gamma}_2, \xi_1^{\mathbb{\Gamma}_1}, \xi_2^{\mathbb{\Gamma}_2}.\\
&\xi_1^{\mathbb{\Gamma}_1} \cong \xi_2^{\mathbb{\Gamma}_2} \;\leftrightarrow\; \mathring{a}(\xi_1) \cup \mathring{a}(\xi_2) \equiv G\\
&\qquad\qquad \wedge\; \mathsf{dom}(\mathbb{\Gamma}_1) \equiv \mathsf{dom}(\mathbb{\Gamma}_2) \;\wedge\; [\![\mathbb{\Gamma}_1]\!]_{\xi_1} \equiv [\![\mathbb{\Gamma}_2]\!]_{\xi_2}\\
&\qquad\qquad \wedge\; \forall\, \delta^{\mathbb{TC}} \in \mathsf{dom}(\mathbb{\Gamma}_1).\ \xi_1(\delta) \equiv \xi_2(\delta)\\
&\qquad\qquad \wedge\; \forall\, x^{\alpha} \in \mathsf{dom}(\mathbb{\Gamma}_1 \backslash_{-TCV}).\ \xi_1(x) \cong_{\alpha}^{G} \xi_2(x)
\end{aligned}$$

The TCVs in $\mathbb{\Gamma}_1$ are equated in the second to last line and the standard variables are equated in the final line, hence the use of $\mathbb{\Gamma}_1 \backslash_{-TCV}$ in the bottom line. This definition requires ξ_1 and ξ_2 to be well constructed models. The LTCs in $\xi_1^{\mathbb{\Gamma}_1} \cong \xi_2^{\mathbb{\Gamma}_2}$ can be different but must contain the same domain, the LTCs are often dropped for brevity.

Lemma 107 (Values derived from equivalent terms extend congruent models)**.**

$$\forall\, M^{\alpha}, \mathbb{\Gamma}_1, \mathbb{\Gamma}_2, \xi_1^{\mathbb{\Gamma}_1}, \xi_2^{\mathbb{\Gamma}_2}.\ \xi_1 \cong \xi_2 \;\wedge\; M \overset{[\mathbb{\Gamma}_i,\ \xi_i]}{\rightsquigarrow} V_i \;\rightarrow\; \xi_1 \cdot x : V_1 \cong \xi_2 \cdot x : V_2$$

Assuming the names produced in V_1 are the same as those in V_2 which can be written more formally as follows.

$$\begin{aligned}
&\forall\, M^{\alpha}, \mathbb{\Gamma}_1, \mathbb{\Gamma}_2, \xi_1^{\mathbb{\Gamma}_1}, \xi_2^{\mathbb{\Gamma}_2}.\ \xi_1 \cong \xi_2\\
&\qquad \wedge\; \forall\, V_1.\ M \overset{[\mathbb{\Gamma}_1,\ \xi_1]}{\rightsquigarrow} V_1\\
&\qquad\qquad \rightarrow\; \exists\, V_2.\ M \overset{[\mathbb{\Gamma}_2,\ \xi_2]}{\rightsquigarrow} V_2 \;\rightarrow\; \xi_1 \cdot x : V_1 \cong \xi_2 \cdot x : V_2
\end{aligned}$$

Proof. The requirement for the names to be the same in both V_1 and V_2 is possible under injective renaming as the fresh names produced in V_2 can be selected to be the fresh names produced in V_1. The proof that this holds is simply a case of proving that because the values that close M in ξ_1 are congruent to those that close M in ξ_2 then $M\xi_1 \cong_{\alpha}^{\mathring{a}(\xi_1,\xi_2)} M\xi_2$ the values produced are by definition congruent up to the renaming of certain fresh names (nominal determinacy Def. 25). □

Lemma 108 (Congruent extensions implies their evaluation added to a model, model equivalently)**.**

$$\begin{aligned}
&\forall\, \mathbb{\Gamma}_1, \mathbb{\Gamma}_2, \xi_1^{\mathbb{\Gamma}_1}, \xi_2^{\mathbb{\Gamma}_2}, A.\ \xi_1 \cong \xi_2 \;\wedge\; \mathbb{\Gamma}_1 \Vdash A \;\wedge\; \mathbb{\Gamma}_2 \Vdash A\\
&\qquad\qquad \rightarrow\\
&\qquad\qquad \xi_1 \models A \;\leftrightarrow\; \xi_2 \models A
\end{aligned}$$

Use I.H. as the additions are all of the same form when proving A:

Proof. Proof by I.H. on the structure of A:

1	Assume $\mathbb{\Gamma}_1$, $\mathbb{\Gamma}_2$, $\xi_1^{\mathbb{\Gamma}_1}$, V_1^α, V_2^α s.t. $\xi \cdot x : V_1 \cong \xi \cdot x : V_2 \wedge \mathbb{\Gamma}_i \Vdash A$
2	Let $G \equiv \mathring{a}(\xi_1) \cup \mathring{a}(\xi_2)$
3	Assume $\xi_1 \models A$ to prove $\xi_2 \models A$
4	Where $\xi_2 \models A \rightarrow \xi_1 \models A$ by symmetry

Prove the final step in line 2 by I.H. on the structure of A as follows.

$A \equiv e = e'$ This holds trivially via Lem. 107 as the models ξ_i have contextually congruent values hence $[\![e]\!]_{\xi_1} \cong_\alpha^G [\![e]\!]_{\xi_2}$ and $[\![e']\!]_{\xi_1} \cong_\alpha^G [\![e']\!]_{\xi_2}$.

$A \equiv \neg A'$ By I.H. on A'.

$A \equiv A_1 \wedge A_2$ By I.H. on A_1 and A_2.

$A \equiv A_1 \vee A_2$ By I.H. on A_1 and A_2.

$A \equiv A_1 \rightarrow A_2$ By I.H. on A_1 and A_2.

$A \equiv e \bullet e' = m^\alpha\{A'\}$ Assuming $\xi_1 \models e \bullet e' = m^\alpha\{A'\}$ iff $\exists V_m.\ ee' \overset{[\mathbb{\Gamma}_1,\ \xi_1]}{\rightsquigarrow} V_m \wedge \xi_1 \cdot m : V_m \models A'$
then it is clear that for all W_m s.t. $ee' \overset{[\mathbb{\Gamma}_2,\ \xi_2]}{\rightsquigarrow} W_m$
assuming the same critical names are produced in both V_m and W_m then Lem. 107 ensures $V_m \cong_\alpha^{G \cup \mathring{a}(V_m, W_m)} W_m$
thus $\xi_1 \cdot m : V_m \cong \xi_2 \cdot m : W_m$
hence by I.H. on A': $\xi_2 \cdot m : W_m \models A'$
hence $\xi_2 \models e \bullet e' = m^\alpha\{A'\}$

$A \equiv \forall u^\alpha \in (\mathbb{\Gamma}_0).A'$ Assume $\xi_1 \models \forall u^\alpha \in (\mathbb{\Gamma}_0).A'$ iff $\forall M_u^\alpha.M_u \overset{[\mathbb{\Gamma}_0,\ \xi_1]}{\rightsquigarrow} V_u \rightarrow \xi_1 \cdot u : V_u \models A'$
Prove $\xi_2 \models \forall u^\alpha \in (\mathbb{\Gamma}_0).A'$ iff $\forall M_u^\alpha.M_u \overset{[\mathbb{\Gamma}_0,\ \xi_2]}{\rightsquigarrow} W_u \rightarrow \xi_2 \cdot u : W_u \models A'$
Assume some M_u such that $M_u \overset{[\mathbb{\Gamma}_0,\ \xi_2]}{\rightsquigarrow} W_u$ then given $[\![\mathbb{\Gamma}_0]\!]_{\xi_2} \equiv [\![\mathbb{\Gamma}_0]\!]_{\xi_1}$
the assumption implies $M_u \overset{[\mathbb{\Gamma}_0,\ \xi_1]}{\rightsquigarrow} V_u$ and $\xi_1 \cdot u : V_u \models A'$
and as W_u and V_u are derived from M_u then Lem. 107 ensures $V_u \cong_\alpha^{G \cup \mathring{a}(V_u, W_u)} W_u$
hence $\xi_1 \cdot u : V_u \cong \xi_2 \cdot u : W_u$
by induction on A' this implies $\xi_2 \cdot u : W_u \models A'$.
Hence $\forall M_u.M_u \overset{[\mathbb{\Gamma}_0,\ \xi_2]}{\rightsquigarrow} W_u \rightarrow \xi_2 \cdot u : W_u \models A'$ hence $\xi_2 \models \forall u \in (\mathbb{\Gamma}_0).A'$

$A \equiv \exists u \in (\mathbb{\Gamma}_0).A'$ Proof holds as $\exists u \in (\mathbb{\Gamma}_0).A' \overset{def}{=} \neg\forall u \in (\mathbb{\Gamma}_0).\neg A'$ which can be proven using the cases above.

$A \equiv \forall\delta.A'$ Assume $\xi_1 \models \forall\delta.A'$ iff $\forall\, \mathbb{\Gamma}'_1, \xi_1^{/\mathbb{\Gamma}'_1}.\xi_1 \preccurlyeq^\star \xi'_1 \rightarrow \xi'_1 \cdot \delta : \mathbb{\Gamma}'_1 \backslash_{-TCV} \models A'$
Prove $\xi_2 \models \forall\delta.A'$ iff $\forall\, \mathbb{\Gamma}'_2, \xi_2^{/\mathbb{\Gamma}'_2}.\xi_2 \preccurlyeq^\star \xi'_2 \rightarrow \xi'_2 \cdot \delta : \mathbb{\Gamma}'_2 \backslash_{-TCV} \models A'$
Assume some $\mathbb{\Gamma}'_2$ and $\xi_2^{/\mathbb{\Gamma}'_2}$ such that $\xi_2 \preccurlyeq^\star \xi'_2$
then derivation (terms and LTCs) required to derive ξ'_2 from ξ_2 derives a model ξ'_1 from ξ_1 such that $\xi'_1 \cong \xi'_2$ (using Lem. 107)
and hence $\xi'_1 \cdot \delta : \mathbb{\Gamma}'_1 \backslash_{-TCV} \cong \xi'_2 \cdot \delta : \mathbb{\Gamma}'_2 \backslash_{-TCV}$ (Def. 106)
and I.H. on A' can now be used to show that given $\xi'_1 \cdot \delta : \mathbb{\Gamma}'_1 \backslash_{-TCV} \models A'$ then $\xi'_2 \cdot \delta : \mathbb{\Gamma}'_2 \backslash_{-TCV} \models A'$ hence the case holds.

$A \equiv e\#\mathbb{\Gamma}_0$ Proof holds by definition of $e\#\mathbb{\Gamma}_0 \stackrel{def}{=} \forall z^{\mathsf{Nm}} \in (\mathbb{\Gamma}_0).z \neq e$ whose cases are covered above.

□

The next 3 lemmas are introduced to prove axiom ($utc1$) which requires that model extensions are maintained when a derivable name is added to both models, proven in Lem. 111. To prove this the single-point extension version in Lem. 110 uses Lem. 109 which states that for any $\mathbb{\Gamma}, \xi$ derived value W, a contextually congruent value can be derived from $\mathbb{\Gamma}{+}x : \mathsf{Nm}, \xi \cdot x : n$ where n is any name derived using the original value W.

Lemma 109 (Derivable names can be added to the model to maintain extensions)**.**

$$\forall\, \mathbb{\Gamma}, \xi^{\mathbb{\Gamma}}, W^\alpha, M, \mathsf{n}.\quad \mathsf{n} \notin \mathring{a}(\xi) \ \wedge\ \mathsf{n} \in \mathring{a}(W)$$
$$\wedge\ M \overset{[\mathbb{\Gamma}+z:\alpha,\ \xi\cdot z:W]}{\rightsquigarrow} \mathsf{n}$$
$$\rightarrow\ \forall\, N^\alpha.N \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} W \rightarrow\ \exists\, N_x^\alpha, W_x^\alpha.\ N_x \overset{[\mathbb{\Gamma}+x:\mathsf{Nm},\ \xi\cdot x:\mathsf{n}]}{\rightsquigarrow} W_x$$
$$\wedge\ W \cong_\alpha^{\mathring{a}(W,W_x)} W_x$$

The essence of the proof is that the value W_x can be derived in the same way W is derived but with a swap of names. The name swapping can occur because both names are accessible (through x and through M) and a name swapping function is provided.

Proof. Using $N_x \equiv N$ would produce W with a fresh name $\mathsf{n}_\#$ where n should be and $\mathsf{n} \notin \mathring{a}(W)$. Define a set of functions that allows for the replacement of one name n' by another name n, in an outputted value V as $\mathsf{Replace}_\alpha(\mathsf{n}\ \mathsf{for}\ \mathsf{n}'\ \mathsf{in}\ V)$. Then $\mathsf{Replace}_\alpha(\mathsf{n}\ \mathsf{for}\ \mathsf{n}'\ \mathsf{in}\ V)$ reduces to a value which is clearly contextually congruent to V, except any occurrence of the name n in V would be swapped for the name n'. This function is defined inductively

on the type of V as follows.

$$\begin{array}{lcl}
\mathsf{Replace}_{\mathsf{Unit}}(x \text{ for } x' \text{ in } V) & = & V \\
\mathsf{Replace}_{\mathsf{Bool}}(x \text{ for } x' \text{ in } V) & = & V \\
\mathsf{Replace}_{\mathsf{Nm}}(x \text{ for } x' \text{ in } V) & = & \mathsf{let}\ v = V\ \mathsf{in}\ \mathsf{if}\ v = x'\ \mathsf{then}\ x\ \mathsf{else}\ v \\
\mathsf{Replace}_{\alpha_1 \times \alpha_2}(x \text{ for } x' \text{ in } V) & = & \mathsf{let}\ v = V\ \mathsf{in} \\
 & & \langle \mathsf{Replace}_{\alpha_1}(x \text{ for } x' \text{ in } \pi_1(v)), \mathsf{Replace}_{\alpha_2}(x \text{ for } x' \text{ in } \pi_2(v)) \rangle \\
\mathsf{Replace}_{\alpha_1 \to \alpha_2}(x \text{ for } x' \text{ in } V) & = & \mathsf{let}\ v = V\ \mathsf{in}\ \lambda z^{\alpha_1}.\mathsf{Replace}_{\alpha_2}(x \text{ for } x' \text{ in } vz)
\end{array}$$

Clearly $\forall\, V(\mathsf{n})^{\alpha}, \mathsf{n}'.\ \mathsf{n}' \notin \mathring{\mathsf{a}}(V(\mathsf{n})) \to V(\mathsf{n}) \cong_{\alpha}^{\mathring{\mathsf{a}}(V(\mathsf{n}),\mathsf{n}')} \mathsf{Replace}_{\alpha}(\mathsf{n} \text{ for } \mathsf{n}' \text{ in } V(\mathsf{n}'))$ holds by definition. This function is used to prove that the value W can be obtained from N_x by defining the N_x given that $N \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} W$ and $M \overset{[\mathbb{\Gamma}+z:\alpha,\ \xi\cdot z:W]}{\rightsquigarrow} \mathsf{n}$.

$$\begin{array}{lll}
N_x \equiv & \mathsf{let}\ z = N\ \mathsf{in} & \text{produces } W \text{ with different } \mathsf{n} \\
 & \mathsf{let}\ x_{new} = M\ \mathsf{in} & \text{extracts new } \mathsf{n} \text{ from new } W \\
 & \quad \mathsf{Replace}_{\alpha}(x_{new} \text{ for } x \text{ in } z) & \text{swap } x_{new} \text{ for } x \text{ in the } W \text{ produced}
\end{array}$$

This produces the value W_x which, assuming all other fresh names are equivalent (nominal determinacy Def. 25), satisfies $W \cong_{\alpha}^{\mathring{\mathsf{a}}(W,W_x)} W_x$ by definition, as required.

□

Lemma 110 (Derivable names can be added to the model to maintain extensions)**.**

$$\forall\, \mathbb{\Gamma}, \xi^{\mathbb{\Gamma}}, W^{\alpha}, M, \mathsf{n}. \left(\begin{array}{l} \mathsf{n} \notin \mathring{\mathsf{a}}(\xi) \ \wedge\ \mathsf{n} \in \mathring{\mathsf{a}}(W) \\ \wedge\ M \overset{[\mathbb{\Gamma}+z:\alpha,\ \xi\cdot z:W]}{\rightsquigarrow} \mathsf{n} \\ \wedge\ \xi \preccurlyeq \xi \cdot z : W \end{array} \right) \to \exists\, W_x^{\alpha}.\ W \cong_{\alpha}^{\mathring{\mathsf{a}}(W)\cup\mathring{\mathsf{a}}(W_x)} W_x \to \xi \cdot x : \mathsf{n} \preccurlyeq \xi \cdot z : W_x \cdot x : \mathsf{n}$$

Proof. This is as a direct consequence of Lem. 109 and the semantics of extensions $\preccurlyeq$. □

Lemma 111 (Names produced from an extension can be added to an extension and still hold)**.**

$$\forall\, \mathbb{\Gamma}, \xi^{\mathbb{\Gamma}}, \mathbb{\Gamma}', \xi'^{\mathbb{\Gamma}'}, M, \mathsf{n}.\ M \overset{[\mathbb{\Gamma}',\ \xi']}{\rightsquigarrow} \mathsf{n} \ \wedge\ \xi \preccurlyeq^{\star} \xi' \to \exists\, \xi''.\ \xi' \cdot x : \mathsf{n} \cong \xi'' \cdot x : \mathsf{n} \ \wedge\ \xi \cdot x : \mathsf{n} \preccurlyeq^{\star} \xi'' \cdot x : \mathsf{n}$$

Proof. Proof by induction on the structure of $\xi \preccurlyeq^{\star} \xi'$ as follows.

$\xi \equiv \xi'$ The proof is trivial given $\xi'' \equiv \xi' \equiv \xi$.

$\xi \preccurlyeq \xi'$ Then $\xi' \equiv \xi \cdot \delta : \mathbb{\Gamma}_0$ holds trivially with some $\mathbb{\Gamma}_0'$ such that $[\![\mathbb{\Gamma}_0]\!]_{\xi} \equiv [\![\mathbb{\Gamma}_0']\!]_{\xi'}$ via $\xi'' \equiv \xi \cdot \delta : \mathbb{\Gamma}_0$.

Then $\xi' \equiv \xi \cdot z : W$ holds by Lem. 110.

$\xi \preccurlyeq^{\star} \xi'' \ \wedge\ \xi' \preccurlyeq^{\star} \xi'$ Then this holds by induction.

□

5.5 Summary

A model for the ν-logic has been introduced, based on the LTC from the logic. Derivations of values from a model and LTC are introduced to ensure hidden names in the model can be used, but are not revealed. The derivations are used to define models being extensions of one another, which, again, maintains the hidden property of names within the model. These two concepts provide the key novelties in the semantics introduced for the expressions, formulae and triples of the ν-logic.

Semantic definitions of properties of formulae are introduced for the two syntactic properties of extension independence and thinness. These semantic definitions model what the syntactic definitions were initially intending.

As a precursor to proving soundness, numerous lemmas regarding the model are introduced and proven. These lemmas facilitate the soundness proofs in the next chapter.

Chapter 6

Soundness

Soundness of a Hoare logic ensures that for any triple derivable from the rules and axioms ($\vdash \{A\}\ M :_u \{B\}$) then the triple is logically valid with respect to the semantics ($\models \{A\}\ M :_u \{B\}$). The converse, *completeness* ensures that any logically valid triple with respect to the model can be proven via the rules. In this section the ν-logic is proven sound with respect to the semantics provided in Chapt. 5. This is achieved through proving all axioms sound in Sec. 6.2 and then proving all the rules sound in Sec. 6.3. The lemmas in Sec. 5.4 provide a basis from which to prove soundness of the axioms and rules introduced in Chapt. 4. The definitions of syntactic extension independence and syntactic thinness introduced in Sec. 4.4 are proven to classify formulae which are indeed semantically extension independent and thin in the model in Sec. 6.1.

In Sec. 6.5 a definition of what it means for a program logic to be a conservative extension is defined alongside a sketch of the ν_{GS}-calculus program logic being a conservative extension of the STLC program logic seen in Sec. 2.2.2.

6.1 Soundness of Properties of Formulae

Syntactic definitions of extension independence and thinness introduced in Def. 54 and Def. 55, respectively, can now be proven to be so by the model under Def. 77 and Def. 78 respectively. The proof of these properties is not essential to the understanding of the logic, but is key to the soundness of the logic. These are represented in the following two lemmas.

The $\textsc{ExtInd}_{Sem}$ property of formulae is used throughout the soundness proofs in Chapt. 6 to ensure specific formulae modelled by one model are also modelled by any extensions and contractions of that model, assuming the typing condition holds.

The syntactic definition of ExtInd_{Syn} in Def. 54 provides a syntactic check on formulae which is proven to imply satisfaction of the ExtInd_{Sem} property from Def. 77 in the Lem. 112.

Lemma 112 (ExtInd_{Syn} implies ExtInd_{Sem}).

$$A\text{-ExtInd}_{Syn} \rightarrow A\text{-ExtInd}_{Sem}$$

Proof. This is proven in Ap. A.1, Lem. 122. □

The syntactic definition is designed to capture all formulae in the reasoning examples yet still satisfy this property, hence this result is not unexpected.

As with the ExtInd_{Sem} property of formulae, the semantic thinness property of formulae is formally defined in Def. 78. The property is used throughout the soundness proofs to ensure certain variables can be removed from a model whilst maintaining satisfaction of the formulae.

In Def. 55 a syntactic check is introduced which guarantees a formulae is thin with respect to a certain variable. A proof that each syntactically thin formulae is indeed semantically thin is as follows.

Lemma 113 (Syntactically thin formulae implies semantically thin).

$$\forall \mathbb{\Gamma}, A, x.\ \mathbb{\Gamma}\backslash x \Vdash A \rightarrow (A\text{-Thin}_{Syn}(x) \rightarrow A\text{-Thin}_{Sem}(x))$$

Proof. This is proven in Ap. A.1 in Lem. 123. □

The syntactic definition is designed to capture all formulae in the reasoning examples yet still satisfy this property.

6.2 Soundness of Axioms

The axioms introduced in Sec. 4.5 are proven sound in the following sections, categorised into primary logical constructors used as follows. The axioms for equality are proven in Sec. 6.2.1. The axioms for universal restricted quantification (and existential) are proven sound in Sec. 6.2.2. The axioms for the freshness constructor are proven sound in Sec. 6.2.3. The axioms for universal quantification over TCV are proven sound in Sec. 6.2.4. Finally, the axioms for evaluation formulae are proven sound in Sec. 6.2.5.

6.2.1 Soundness of Axioms for Equality

The following axioms are proven trivially:
$(eq1) \equiv e = e$, $(eq2) \equiv e = e' \leftrightarrow e' = e$, $(eq3) \equiv e = e' \wedge e' = e'' \rightarrow e = e''$

Soundness Proof of Axiom (*eq4*)

This axiom is critical to allow us to deconstruct the logical constructor of equality. The primary idea is that because x and e are congruent they can be substituted in the model for one another using Lem. 108.

$$\mathbb{\Gamma} \Vdash A \wedge x = e \to A[e/x]_{\mathbb{\Gamma}}$$

Proof.

1	Assume $\mathbb{\Gamma}$, $\xi^{\mathbb{\Gamma}_d}$ s.t. $\mathbb{\Gamma} \triangleright \xi \wedge \mathbb{\Gamma} \Vdash A \wedge x = e \to A[e/x]_{\mathbb{\Gamma}}$ then:	
2	$\xi^{\mathbb{\Gamma}_d} \models A \wedge x = e$ and e free for x in A	*Assume*
3	$\xi \models A \wedge \xi \models x = e$	*Sem.* $\wedge$
4	$\xi \models A \wedge [\![x]\!]_\xi \cong_\alpha^{\mathring{a}(\xi)} [\![e]\!]_\xi$	*Sem.*$=$
5	$\exists V_e.\ \xi \models A \wedge [\![x]\!]_\xi \cong_\alpha^{\mathring{a}(\xi)} [\![e]\!]_\xi \wedge e \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_e$	*Lem. 88*
6	$\exists V_e.\ \xi \models A \wedge [\![x]\!]_\xi \cong_\alpha^{\mathring{a}(\xi)} [\![e]\!]_\xi \wedge [\![e]\!]_\xi \cong_\alpha^{\mathring{a}(\xi)} V_e \wedge e \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_e$	*Lem. 91*
7	$\exists V_e.\ \xi \models A \wedge [\![x]\!]_\xi \cong_\alpha^{\mathring{a}(\xi)} V_e \wedge e \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_e$	*Transitivity of* $\cong$
8	$\to \exists V_e.\ \xi \models A \wedge e \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_e$ $\wedge \forall x', A_0.\ x' \notin \mathsf{dom}(\mathbb{\Gamma}) \wedge \mathbb{\Gamma} \cdot x' : \mathbb{\Gamma}(x) \Vdash A_0$ $\to (\xi \cdot x' : \xi(x) \models A_0[x'/x] \leftrightarrow \xi \cdot x' : V_e \models A_0[x'/x])$	*Lem. 108*
9	$\leftrightarrow \exists V_e.\ \xi \models A \wedge e \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_e$ $\wedge \forall x'.\ x' \notin \mathsf{dom}(\mathbb{\Gamma})$ $\to (\xi \models A \leftrightarrow \xi \cdot x' : V_e \models A[x'/x])$	$\xi \cdot x' : \xi(x) \models A[x'/x] \leftrightarrow \xi \models A$ *Let* $A_0 \equiv A$
10	$\to \exists V_e.\ e \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_e \wedge \forall x'.x' \notin \mathsf{dom}(\mathbb{\Gamma}) \to \xi \cdot x' : V_e \models A[x'/x]$	*M.P.*
11	$\to \forall x'.x' \notin \mathsf{dom}(\mathbb{\Gamma}) \to \exists V_e.\ e \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_e \wedge \xi \cdot x' : V_e \models A[x'/x])$	*F.O.L.*
12	$\leftrightarrow \forall x'.x' \notin \mathsf{dom}(\mathbb{\Gamma}) \to \xi \models (A[x'/x])[e/x']$	*Sem.*$A[e/x']$ $(x' \notin \mathsf{fv}(\xi))$
13	$\leftrightarrow \xi \models A[e/x]$	*Sem.*$A[e/x]$ $(x \in \mathsf{fv}(\xi))$
14	Hence: $\forall \xi^{\mathbb{\Gamma}}.\mathbb{\Gamma} \Vdash A \wedge x = e \to A[e/x] \to \xi \models A \wedge x = e \to A[e/x]$	*Lines.1-13*

□

6.2.2 Soundness of Axioms for Restricted Quantification

Soundness Proof of Axiom $(u1)$

$$\mathbb{\Gamma}_0 \Vdash e : \alpha \quad \rightarrow \quad \mathbb{\Gamma} \Vdash \forall x^{\alpha} \in (\mathbb{\Gamma}_0).A \quad \rightarrow \quad A[e/x]_{\mathbb{\Gamma}}$$

Proof.

1	Let: $(u1) \equiv \forall x^{\alpha} \in (\mathbb{\Gamma}_0).A \ \rightarrow \ A[e/x]$	
2	Assume $\mathbb{\Gamma}$, $\xi^{\mathbb{\Gamma}_d}$ s.t. $\mathbb{\Gamma} \triangleright \xi \ \wedge \ \mathbb{\Gamma} \Vdash (u1)$ and $\mathbb{\Gamma}_0 \Vdash e : \alpha$	
3	Assume $\xi \models \forall x^{\alpha} \in (\mathbb{\Gamma}_0).A$ and $\mathbb{\Gamma}_0 \vdash e : \alpha$	
4	$\forall\, M_x, V_x . M_x \overset{[\mathbb{\Gamma}_0,\ \xi]}{\rightsquigarrow} V_x \ \rightarrow \ \xi \cdot x : V_x \models A$	*Sem.*$\forall \in ()$.
5	$e -$ expression $\rightarrow$ $e -$ term $\rightarrow$ $\mathring{a}(e) = \emptyset$	*Lem. 89*
6	$\mathbb{\Gamma} \Vdash \mathbb{\Gamma}_0 \ \rightarrow \ (\llbracket \mathbb{\Gamma}_0 \rrbracket_{\xi} \Vdash e : \alpha \ \rightarrow \ \llbracket \mathbb{\Gamma} \rrbracket_{\xi} \Vdash e : \alpha)$	*Lem. 92*
7	$\mathbb{\Gamma}_0 \Vdash e : \alpha \ \rightarrow \ \exists\, V_e .\ e \overset{[\mathbb{\Gamma}_0,\ \xi]}{\rightsquigarrow} V_e$	*Lem. 88*
8	$\exists\, V_e .\ e \overset{[\mathbb{\Gamma}_0,\ \xi]}{\rightsquigarrow} V_e \ \wedge \ \xi \cdot x : V_e \models A$	*Instant. Line.3* M_x, V_x *to* e, V_e, *M.P.*
9	$\xi \models A[e/x]$	*Sem.*$[e/x]$, $x \notin \mathsf{dom}(\xi)$
10	Hence $\xi \models \forall x^{\alpha} \in (\mathbb{\Gamma}_0).A$ and $\mathbb{\Gamma}_0 \vdash e : \alpha$ implies $\xi \models A[e/x]$	*Lines.3-9*
11	Hence $\forall\, \mathbb{\Gamma}, \mathbb{\Gamma}_0, A, e.\ \mathbb{\Gamma} \Vdash (u1) \ \wedge \ \mathbb{\Gamma}_0 \Vdash e : \alpha \ \rightarrow \ \forall\, \xi . \mathbb{\Gamma} \triangleright \xi \ \rightarrow \ \xi \models (u1)$	*Lines.1-10*

□

Soundness Proof of Axiom $(u2)$

$$A\text{-}\textsc{ExtInd}_{Syn} \quad \rightarrow \quad A^{-x} \leftrightarrow \forall x^{\alpha} \in (\mathbb{\Gamma}_0).A$$

Proof. Given Lem. 112 ensures A-$\textsc{ExtInd}_{Syn}$ implies A-$\textsc{ExtInd}_{Sem}$, then clearly it is only required to show that for any V derived from $\mathbb{\Gamma}_0$ and ξ then $\xi \preccurlyeq^{\star} \xi \cdot x : V$ which is easily proven by Lem. 93 and Def. 70 and hence this holds. □

Soundness Proof of Axiom $(u3)$

$$(\forall x^{\alpha} \in (\mathbb{\Gamma}_0).(A \wedge B)) \leftrightarrow (\forall x^{\alpha} \in (\mathbb{\Gamma}_0).A) \wedge (\forall x^{\alpha} \in (\mathbb{\Gamma}_0).B)$$

Proof. This clearly holds given the F.O.L. axiom in the meta-logic: $(\forall\, x.A \wedge B) \leftrightarrow (\forall\, x.A \wedge \forall\, x.B)$ and the use of $\forall$ in the semantics of $\forall x \in (\mathbb{\Gamma}_0)$.. □

Soundness Proof of Axiom $(u4)$

$$\mathbb{\Gamma}_0 \Vdash \mathbb{\Gamma}_1 \quad \to \quad \forall x^{\alpha} \in (\mathbb{\Gamma}_0).A \quad \to \quad \forall x^{\alpha} \in (\mathbb{\Gamma}_1).A$$

Proof. Clearly holds by Lem. 93 and the semantics of $\forall x \in (\mathbb{\Gamma}_i)$.. □

Soundness Proof of Axiom $(u5)$

$$\alpha \in \alpha_{\text{-Nm}} \quad \to \quad \forall x^{\alpha} \in (\mathbb{\Gamma}_0).A \;\leftrightarrow\; \forall x^{\alpha} \in (\emptyset).A$$

Proof. Both $\forall x \in (\mathbb{\Gamma}_0).$ and $\forall x \in (\emptyset).$ quantify over the same set of values as the values are all of type $\alpha_{\text{-Nm}}$ hence Lem. 87 applies. The essence is that every value of type $\alpha_{\text{-Nm}}$ is contextually equivalent to a value without a name (i.e. an STLC term) which can be derived from $\emptyset$. □

Soundness Proof of Axiom $(u6)$

$$\mathbb{\Gamma}{+}x{+}y \Vdash x\#\mathbb{\Gamma} \wedge y\#\mathbb{\Gamma}{+}x \to \forall f^{\mathsf{Nm}\to\mathsf{Nm}} \in (\mathbb{\Gamma}).f \bullet x = m\{m \neq y\}$$

Proof. Trivial given some model $\mathbb{\Gamma}{+}x{+}y \triangleright \xi$ such that $\xi \models x\#\mathbb{\Gamma} \wedge y\#\mathbb{\Gamma}{+}x$ then for any M_f s.t. $M_f \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_f$ then $fx \overset{[\mathbb{\Gamma}+x+y+f,\ \xi\cdot f:V_f]}{\rightsquigarrow} V$. Given $\xi(y) \notin \mathring{a}(M_f, V_f, x)$ then $\xi(y) \notin \mathring{a}((fx)\xi)$ and $\xi(y) \notin \mathring{a}(V) = V$ hence $V \neq \xi(y)$ and the axiom holds. □

Soundness Proof of Axiom $(u7)$

$$\mathbb{\Gamma}{+}x{+}y \Vdash x\#\mathbb{\Gamma} \wedge y\#\mathbb{\Gamma}{+}x \;\to\; \forall f^{\mathsf{Nm}\to\mathsf{Bool}} \in (\mathbb{\Gamma}).f \bullet x = m\{f \bullet y = n\{m = n\}\}$$

Proof. Clearly f is derived from $\mathbb{\Gamma}$ which does not contain the names x or y given well constructed models and $x\#\mathbb{\Gamma}$ and $y\#\mathbb{\Gamma}+x$. Hence the function f cannot distinguish names x and y or any other fresh name meaning the output of fx is a Boolean and the same Boolean is outputted by fy (and also $f(\mathsf{gensym}())$). This is holds given the nominal determinacy property Def. 25. □

This is used in Ex. 36 where ideally the axiom would have the form

$$\mathbb{\Gamma}{+}x{+}y{+}a : (\mathsf{Nm} \to \mathsf{Bool}) \to \mathsf{Bool} \Vdash \; x\#\mathbb{\Gamma} \wedge y\#\mathbb{\Gamma}{+}x \to \; \forall f^{\mathsf{Nm}\to\mathsf{Bool}} \in (\mathbb{\Gamma}{+}a).f \bullet x = m\{f \bullet y = n\{m = n\}\}$$

However this fails for some cases of a for instance if $a : \lambda z^{\mathsf{Nm}\to\mathsf{Bool}}.zx$
and $f : \lambda m^{\mathsf{Nm}}.(a(\lambda n^{\mathsf{Nm}}.n = m))$ hence this version of the axiom does not hold.

Soundness Proof of Axiom $(u8)$

$$\mathbb{\Gamma} \Vdash (\forall x^{\mathsf{Nm}} \in (\emptyset).\ x\#\mathbb{\Gamma} \wedge A) \quad \leftrightarrow \quad \forall x^{\mathsf{Nm}} \in (\emptyset).\ A$$

Proof. Clearly the only name derivable from the LTC $\emptyset$ is a fresh name produced by a term equivalent to $\mathsf{gensym}()$ hence the name stored at x must be fresh from all other names derived from the nameset from which it is fresh which is represented by $\mathbb{\Gamma}$ hence $x\#\mathbb{\Gamma}$ is guaranteed.

□

Soundness Proof of Axiom $(u9)$

$$\mathbb{\Gamma}_1 \Vdash e : \mathsf{Nm} \quad \rightarrow \quad \mathbb{\Gamma} \Vdash \forall x^{\mathsf{Nm}} \in (\mathbb{\Gamma}_1).a \bullet x = e \ \rightarrow\ \forall x^{\mathsf{Nm}} \in (\mathbb{\Gamma}_1{+}a).a \bullet x = e$$

Proof. Clearly any application of a to a name derivable from $\mathbb{\Gamma}_1$ is another name derivable from $\mathbb{\Gamma}_1$, however there is a possibility that a can be applied to a name derivable from $\mathbb{\Gamma}_1{+}a$ yet not derivable from $\mathbb{\Gamma}_1$ hence producing a name not equivalent to e. Assume some model $\mathbb{\Gamma} \triangleright \xi$ such that $\mathbb{\Gamma} \Vdash \mathbb{\Gamma}_1{+}a$, and $\mathbb{\Gamma}_1{+}a \Vdash e : \mathsf{Nm}$ and $\xi \models \forall x^{\mathsf{Nm}} \in (\mathbb{\Gamma}_1).a \bullet x = e$ it is required to prove $\xi \models \forall x^{\mathsf{Nm}} \in (\mathbb{\Gamma}_1{+}a).a \bullet x = e$.

For example consider the case where $\xi(b) \equiv \lambda f^{\mathsf{Nm}\to\mathsf{Nm}}\mathsf{if}\ f\mathsf{n}_1 = \mathsf{n}_2\ \mathsf{then}\ \mathsf{n}_3\ \mathsf{else}\ \mathsf{n}_0$ where n_1, n_2, n_3 are not derivable from $\xi\backslash a$, with $\xi(a) \equiv \lambda x^{\mathsf{Nm}}.\mathsf{if}\ x = \mathsf{n}_1\ \mathsf{then}\ \mathsf{n}_2\ \mathsf{else}\ \mathsf{n}_0$. The evaluation of ba returns n_3 which was not previously derivable from $\xi\backslash a$, however this is impossible by the fact that ξ is a well constructed model hence cannot reveal previously hidden names and thus the names n_1 and n_2 cannot appear in a unless they are already derivable from $\xi\backslash a$ or are fresh which is known not to be the case.

The general case is satisfied by the fact that ξ is a well constructed model ensuring that any name n' derivable from ξ ensures $a\mathsf{n}'$ does not return e. By definition n' must be in $\mathring{\mathsf{a}}(\xi(a))$ (otherwise $a\mathsf{n}' \cong^{\ldots}_{\mathsf{Nm}} a(\mathsf{gensym}())(\cong^{\ldots}_{\mathsf{Nm}} e)$), hence n' cannot be derived from ξ. This means n' isn't quantified over by $\forall x \in (\mathbb{\Gamma}_1{+}a).$ and hence this axiom holds. □

Soundness Proof of Axiom $(u10)$

$$\mathbb{\Gamma}{+}a : \alpha \to \mathsf{Bool} \Vdash \forall f^{\alpha} \in (\mathbb{\Gamma}).a \bullet f = c \ \rightarrow\ \forall f^{\alpha} \in (\mathbb{\Gamma}{+}a).a \bullet f = c$$

Consider the simple case where $\alpha \equiv \mathsf{Nm}$ then $M(a)$ uses $a : \mathsf{Nm} \to \mathsf{Bool}$. However by definition of a well constructed model, the name at f cannot be the fresh name hidden in a as this is not revealed by a. Hence f must be fresh (in which case a cannot distinguish it), or derivable from ξ and hence no new terms are derivable from $\mathbb{\Gamma}{+}a$ which were not already derivable from $\mathbb{\Gamma}$.

Proof.

1	Assume $\xi \cdot a : V_a$ s.t. $\mathbb{\Gamma}{+}a : \alpha \to \mathsf{Bool} \triangleright \xi \cdot a : V_a$
2	Assume $\xi \cdot a : V_a \models \forall f^{\alpha} \in (\mathbb{\Gamma}).a \bullet f = c$
3	$\leftrightarrow$ $\forall\, M_f^{\alpha}.M_f \overset{[\mathbb{\Gamma},\ \xi\cdot a:V_a]}{\rightsquigarrow} V_f$ $\to$ $\xi \cdot a : V_a \cdot f : V_f \models a \bullet f = c$
4	Prove $\xi \cdot a : V_a \models \forall f^{\alpha} \in (\mathbb{\Gamma}{+}a).a \bullet f = c$
5	$\leftrightarrow$ $\forall\, M_f'^{\alpha}.M_f' \overset{[\mathbb{\Gamma}+a,\ \xi\cdot a:V_a]}{\rightsquigarrow} V_f'$ $\to$ $\xi \cdot a : V_a \cdot f : V_f' \models a \bullet f = c$
6	Proof by contradiction:
7	Assume $\xi \cdot a : V_a \models \forall f^{\alpha} \in (\mathbb{\Gamma}).a \bullet f = c \ \wedge\ \exists f^{\alpha} \in (\mathbb{\Gamma}{+}a).a \bullet f = \neg c$
8	$\to$ $\exists\, M_f'^{\alpha}.M_f' \overset{[\mathbb{\Gamma},\ \xi\cdot a:V_a]}{\rightsquigarrow} V_f'$ $\wedge$ $\xi \cdot a : V_a \cdot f : V_f' \models a \bullet f = c$
9	Select smallest M_f' for which this is the case, and show a contradiction exists
10	$[\![\mathbb{\Gamma}{+}a]\!]_{\xi\cdot a:V_a} \vdash M_f' : \alpha$ $\wedge$ $\emptyset \vdash V_a : \alpha \to \mathsf{Bool}$
11	Hence M_f' must apply (or discard) every use of a
12	Each application of a in M_f' must be to a term smaller than M_f' hence will return c
13	Hence each application of V_a in $(M_f'[V_a/a])\xi$ is equivalent to $\lambda z.c$
14	Hence $(M_f'[V_a/a])\xi$ is equivalent to $(M_f'[\lambda z.c/a])\xi$
15	Given $\mathbb{\Gamma} \vdash M_f'[\lambda z.c/a] : \alpha$ then by line 3 implies $(M_f'[\lambda z.c/a]) \overset{[\mathbb{\Gamma},\ \xi\cdot a:V_a]}{\rightsquigarrow} c$
16	Hence a contradiction with line 9 meaning no such term exists
17	Hence contradiction with the original assumption in line 7 and the axiom holds

□

Soundness Proof of Axiom $(ex1)$

$$(\mathbb{\Gamma} \Vdash \mathbb{\Gamma}_0 \wedge \mathbb{\Gamma}_0 \Vdash e : \alpha) \quad \to \quad \mathbb{\Gamma} \Vdash A[e/x]_{\mathbb{\Gamma}} \to \exists x \in (\mathbb{\Gamma}_0).x = e \wedge A$$

Proof. Assume $\mathbb{\Gamma}^{-x}, \xi^{\mathbb{\Gamma}_d}, \mathbb{\Gamma}_0^{-x}, e, A$

such that $\mathbb{\Gamma} \Vdash A[e/x] \to \exists x \in (\mathbb{\Gamma}).A$, $\mathbb{\Gamma} \Vdash \mathbb{\Gamma}_0$ and $\mathbb{\Gamma}_0 \Vdash e : \alpha$

1	Assume $\mathbb{\Gamma}$ s.t. $\mathbb{\Gamma} \Vdash A[e/x] \to \exists x \in (\mathbb{\Gamma}_0).x = e \wedge A$	
2	Assume $\mathbb{\Gamma}_0$, e s.t. $\mathbb{\Gamma} \Vdash \mathbb{\Gamma}_0$ and $\mathbb{\Gamma}_0 \Vdash e : \alpha$	
3	Assume $\xi^{\mathbb{\Gamma}_d}$ s.t. $\mathbb{\Gamma} \triangleright \xi$ and $\xi \models A[e/x]$	
4	$\leftrightarrow$ $\exists V.\ e \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V \wedge \xi \cdot x : V \models A$	*Sem.*$A[e/x]$, $x \notin \mathsf{dom}(\xi)$
5	$\leftrightarrow$ $\exists V.\ e \overset{[\mathbb{\Gamma}_0,\ \xi]}{\rightsquigarrow} V \wedge \xi \cdot x : V \models A \wedge e(\xi \cdot x : V) \cong_\alpha^{\mathring{a}(\xi \cdot x:V)} x(\xi \cdot x : V)$	*Lem. 91*
6	$\leftrightarrow$ $\exists V.\ e \overset{[\mathbb{\Gamma}_0,\ \xi]}{\rightsquigarrow} V \wedge \xi \cdot x : V \models x = e \wedge A$	*Sem.*$=, \wedge$
7	$\to$ $\exists M_e, V.M_e \overset{[\mathbb{\Gamma}_0,\ \xi]}{\rightsquigarrow} V \wedge \xi \cdot x : V \models x = e \wedge A$	*Clearly holds for* $M_e = e$
8	$\leftrightarrow$ $\xi \models \exists x \in (\mathbb{\Gamma}_0).x = e \wedge A$	*Sem.*$\exists x \in (\mathbb{\Gamma}_0).$

□

Soundness Proof of Axiom $(ex2)$

$$A\text{-}\textsc{ExtInd}_{Syn} \wedge x \notin \mathsf{fv}(A) \quad \to \quad A \wedge \exists x \in (\mathbb{\Gamma}_0).B \ \leftrightarrow \ \exists x \in (\mathbb{\Gamma}_0).(A \wedge B)$$

This is inspired by a similar axiom in F.O.L. and is proven using this axiom in the meta-F.O.L..

Proof. Clearly A-$\textsc{ExtInd}_{Syn}$ implies A-$\textsc{ExtInd}_{Sem}$ via Lem. 112.

1	Let: $(ex2) \equiv A \wedge \exists x \in (\mathbb{\Gamma}_0).B \ \leftrightarrow \ \exists x \in (\mathbb{\Gamma}_0).(A \wedge B)$	
2	Assume $\mathbb{\Gamma}$ s.t. $\mathbb{\Gamma} \Vdash (ex2)$ and $\xi^{\mathbb{\Gamma}_d}$ s.t. $\mathbb{\Gamma} \triangleright \xi$	
3	Assume $\xi \models A \wedge \exists x \in (\mathbb{\Gamma}_0).B$	
4	$\leftrightarrow$ $\xi \models A \wedge \exists M, V.\ M \overset{[\mathbb{\Gamma}_0,\ \xi]}{\rightsquigarrow} V \wedge \xi \cdot x : V \models B$	*Sem.*$\wedge, \exists \in ().$
5	$\leftrightarrow$ $\exists M, V.\ M \overset{[\mathbb{\Gamma}_0,\ \xi]}{\rightsquigarrow} V \wedge \xi \models A \wedge \xi \cdot x : V \models B$	*F.O.L.*
6	$\leftrightarrow$ $\exists M, V.\ M \overset{[\mathbb{\Gamma}_0,\ \xi]}{\rightsquigarrow} V \wedge \xi \cdot x : V \models A \wedge \xi \cdot x : V \models B$	A-$\textsc{ExtInd}_{Sem}$
7	$\leftrightarrow$ $\xi \models \exists x \in (\mathbb{\Gamma}_0).(A \wedge B)$	*Sem.*$\wedge, \exists \in ().$
8	Hence: $\forall \mathbb{\Gamma}.\ \mathbb{\Gamma} \Vdash (ex2) \to \forall \xi^{\mathbb{\Gamma}_d}.\ \mathbb{\Gamma} \triangleright \xi \to \xi \models (ex2)$	*Lines.1-7*

□

Soundness Proof of Axiom $(ex3)$

$$\{a,b\} \subseteq \mathsf{fv}(\mathbb{\Gamma}_0) \quad \to \quad \mathbb{\Gamma}{+}x{+}\mathbb{\Gamma}_0 \Vdash a \bullet b = c\{c = x\} \ \to \ \exists x' \in (\mathbb{\Gamma}_0).x = x'$$

Proof.

1	Let: $(ex3) \equiv a \bullet b = c\{c = x\} \ \to \ \exists x' \in (\mathbb{\Gamma}_0).x = x'$	
2	Assume $\xi^{\mathbb{\Gamma}_d}$ s.t. $\mathbb{\Gamma}{+}x{+}\mathbb{\Gamma}_0 \triangleright \xi$ and $\xi \models a \bullet b = c\{c = x\}$	
3	$\to \exists\, V_c.\ ab \overset{[\mathbb{\Gamma}_0,\ \xi]}{\rightsquigarrow} V_c \ \wedge\ \xi \cdot c : V_c \models x = c$	$Sem.\bullet = \{\}$
4	$\to \exists\, M'_x, V_c.\ M'_x \overset{[\mathbb{\Gamma}_0,\ \xi]}{\rightsquigarrow} V_c \wedge \xi \cdot c : V_c \models x = c$	$M'_x \equiv ab,\ \{a,b\} \subseteq \mathsf{fv}(\mathbb{\Gamma}_0)$
5	$\to \xi \models \exists x' \in (\mathbb{\Gamma}_0).x = x'$	$Sem.\exists \in ().$
6	Hence: $\forall\, \mathbb{\Gamma}{+}x{+}\mathbb{\Gamma}_0.\ \mathbb{\Gamma}{+}x{+}\mathbb{\Gamma}_0 \Vdash (ex3) \ \to\ \forall\, \xi^{\mathbb{\Gamma}_d}.\ \mathbb{\Gamma}{+}x{+}\mathbb{\Gamma}_0 \triangleright \xi \ \to\ \xi \models (ex3)$	

□

Soundness Proof of Axiom $(ex4)$

$$\mathbb{\Gamma}{+}x \Vdash \forall y^{\mathsf{Nm}} \in (\emptyset).\exists z^{\mathsf{Nm}} \in (\mathbb{\Gamma}_0{+}y).x = z \ \to \ \exists z \in (\mathbb{\Gamma}_0).x = z$$

Proof.

1	Assume some model $\xi^{\mathbb{\Gamma}_d}$ s.t. $\mathbb{\Gamma}{+}x \triangleright \xi$ and $\xi \models \forall y^{\mathsf{Nm}} \in (\emptyset).\exists z^{\mathsf{Nm}} \in (\mathbb{\Gamma}_0{+}y).x = z$	
2	Let: $\xi_y \equiv \xi \cdot y : V_y,\ \xi_{yz} \equiv \xi_y \cdot z : V_z,\ \xi_z \equiv \xi \cdot z : V_z$	
3	$\to \forall\, M_y^{\mathsf{Nm}}, V_y.\ M_y \overset{[\emptyset,\ \xi]}{\rightsquigarrow} V_y \ \to \exists\, M_z^{\mathsf{Nm}}, V_z.\ M_z \overset{[\mathbb{\Gamma}_0{+}y,\ \xi_y]}{\rightsquigarrow} V_z \wedge \xi_{yz} \models x = z$	
4	$\to \forall\, M_y^{\mathsf{Nm}}, V_y.\ M_y \overset{[\emptyset,\ \xi]}{\rightsquigarrow} V_y \ \to \exists\, M_z^{\mathsf{Nm}}, V_z.\ M_z \overset{[\mathbb{\Gamma}_0,\ \xi_y]}{\rightsquigarrow} V_z \wedge \xi_{yz} \models x = z$	*Lem. 32*
5	$\to \forall\, M_y^{\mathsf{Nm}}, V_y.\ M_y \overset{[\emptyset,\ \xi]}{\rightsquigarrow} V_y \ \to \exists\, M_z^{\mathsf{Nm}}, V_z.\ M_z \overset{[\mathbb{\Gamma}_0,\ \xi]}{\rightsquigarrow} V_z \wedge [\![x]\!]_{\xi_{yz}} \cong_{\mathsf{Nm}}^{\mathring{\mathsf{a}}(\xi_{yz})} [\![z]\!]_{\xi_{yz}}$	$Sem.=$
6	$\to \forall\, M_y^{\mathsf{Nm}}, V_y.\ M_y \overset{[\emptyset,\ \xi]}{\rightsquigarrow} V_y$ $\to \exists\, M_z^{\mathsf{Nm}}, V_z.\ M_z \overset{[\mathbb{\Gamma}_0,\ \xi]}{\rightsquigarrow} V_z \wedge [\![x]\!]_{\xi_z} \cong_{\mathsf{Nm}}^{\mathring{\mathsf{a}}(\xi_{yz})} [\![z]\!]_{\xi_z}$	$[\![x]\!]_{\xi_{yz}} = [\![x]\!]_{\xi_z}$
7	$\to \forall\, M_y^{\mathsf{Nm}}, V_y.\ M_y \overset{[\emptyset,\ \xi]}{\rightsquigarrow} V_y \ \to \exists\, M_z^{\mathsf{Nm}}, V_z.\ M_z \overset{[\mathbb{\Gamma}_0,\ \xi]}{\rightsquigarrow} V_z \wedge [\![x]\!]_{\xi_z} \cong_{\mathsf{Nm}}^{\mathring{\mathsf{a}}(\xi_z)} [\![z]\!]_{\xi_z}$	*Lem. 32*
8	$\to \exists\, M_z^{\mathsf{Nm}}, V_z.\ M_z \overset{[\mathbb{\Gamma}_0,\ \xi]}{\rightsquigarrow} V_z \wedge [\![x]\!]_{\xi_z} \cong_{\mathsf{Nm}}^{\mathring{\mathsf{a}}(\xi_z)} [\![z]\!]_{\xi_z}$	*F.O.L.*
9	$\to \xi \models \exists z \in (\mathbb{\Gamma}_0).x = z$	$Sem.\exists \in ().$
10	Hence: $\forall\, \xi^{\mathbb{\Gamma}_d}.\ \mathbb{\Gamma}{+}x \triangleright \xi$ $\to \xi \models \forall y^{\mathsf{Nm}} \in (\emptyset).\exists z^{\mathsf{Nm}} \in (\mathbb{\Gamma}_0{+}y).x = z \ \to \ \exists z \in (\mathbb{\Gamma}_0).x = z$	*Lines.1-9*

□

6.2.3 Soundness of Axioms for Freshness

Soundness Proof of Axiom $(f1)$

$$\mathbb{\Gamma}_0 \Vdash e' : \mathsf{Nm} \quad \rightarrow \quad e\#\mathbb{\Gamma}_0 \ \rightarrow \ e \neq e'$$

Proof. Proven directly from syntactic definition of $e\#\mathbb{\Gamma}_0$ and $(u1)$. □

Soundness Proof of Axiom $(f2)$

$$\mathbb{\Gamma}_0 \Vdash \mathbb{\Gamma}_1 \quad \rightarrow \quad e\#\mathbb{\Gamma}_0 \ \rightarrow \ e\#\mathbb{\Gamma}_1$$

Proof. Proven directly from syntactic definition of $e\#\mathbb{\Gamma}_0$, and $(u4)$. □

Soundness Proof of Axiom $(f3)$

$$\mathbb{\Gamma}{+}x : \mathsf{Nm}{+}f : \alpha \to \alpha_{\text{-}(\mathsf{Nm},\to)} \Vdash x\#\mathbb{\Gamma} \to x\#\mathbb{\Gamma}{+}f : \alpha \to \alpha_{\text{-}(\mathsf{Nm},\to)}$$

Proof. Use Lem. 84 to so show that f cannot help produce x if $f : \alpha \to \alpha_{\text{-}(\mathsf{Nm},\to)}$.

1	Let: $\mathbb{\Gamma}_{xf} \equiv \mathbb{\Gamma}{+}x : \mathsf{Nm}{+}f : \alpha \to \alpha_{\text{-}(\mathsf{Nm},\to)}$ and $\mathbb{\Gamma}_{0xf} \equiv \mathbb{\Gamma}_0{+}x{+}f$	
2	Assume ξ_{xf} s.t. $\mathbb{\Gamma}_{xf} \triangleright (\xi^{\mathbb{\Gamma}_0} \cdot x : \mathsf{n}_x \cdot f : V_f)^{\mathbb{\Gamma}_{0xf}} \equiv \xi_{xf}$	
3	$\mathbb{\Gamma}_{xf} \triangleright \xi_{xf} \to \exists M_x.M_x \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} \mathsf{n}_x \wedge \exists M_f.M_f \overset{[\mathbb{\Gamma}+x,\ \xi\cdot x:\mathsf{n}_x]}{\rightsquigarrow} V_f$	
4	Assume $\xi_{xf} \models x\#\mathbb{\Gamma}$	*Assumption*
5	Hence: $\exists M_x.M_x \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} \mathsf{n}_x \wedge \neg \exists N_x.N_x \overset{[\mathbb{\Gamma},\ \xi_{xf}]}{\rightsquigarrow} \mathsf{n}_x$	*Sem.*$x\#\mathbb{\Gamma}$*, Lines.3, 4*
6	Hence: $\exists M_x.M_x \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} \mathsf{n}_x \wedge \neg \exists N_x.N_x \overset{[\mathbb{\Gamma},\ \xi\cdot x:\mathsf{n}_x]}{\rightsquigarrow} \mathsf{n}_x$	*Lem. 102*
7	$\to \mathsf{n}_x \notin \mathring{\mathsf{a}}(\xi)$	*Lem. 94*
8	Assume $\mathsf{n}_x \notin \mathring{\mathsf{a}}(V_f) \to \neg \exists P_x.P_x \overset{[\mathbb{\Gamma}+f,\ \xi_{xf}]}{\rightsquigarrow} \mathsf{n}_x$	*Trivial Lem. 97*
9	Assume $\mathsf{n}_x \in \mathring{\mathsf{a}}(V_f)$	*(proof by contradiction, Lines.10-15)*
10	Assume $\exists P_x.P_x \overset{[\mathbb{\Gamma}+f,\ \xi_{xf}]}{\rightsquigarrow} \mathsf{n}_x$	
11	$\leftrightarrow \exists P_x.\ \mathring{\mathsf{a}}(P_x) = \emptyset \wedge [\![\mathbb{\Gamma}]\!]_\xi, f \vdash P_x : \mathsf{Nm}$ $\wedge\ (\mathring{\mathsf{a}}(\xi_{xf}),\ P_x\xi_{xf}) \Downarrow (\mathring{\mathsf{a}}(\xi_{xf}), G',\ \mathsf{n}_x)$	*Sem.*$\overset{[,]}{\rightsquigarrow}$
12	$\leftrightarrow \exists P_x.\ \mathring{\mathsf{a}}(P_x) = \emptyset \wedge [\![\mathbb{\Gamma}]\!]_\xi, f \vdash P_x : \mathsf{Nm}$ $\wedge\ (\mathring{\mathsf{a}}(\xi_{xf}),\ P_x[V_f/f]\xi) \Downarrow (\mathring{\mathsf{a}}(\xi_{xf}), G',\ \mathsf{n}_x)$	*Lem. 63,* P_x^{-x}
13	$\leftrightarrow \exists P_x.\ \mathring{\mathsf{a}}(P_x) = \emptyset \wedge [\![\mathbb{\Gamma}]\!]_\xi, f \vdash P_x : \mathsf{Nm}$ $\wedge\ (\mathring{\mathsf{a}}(\xi_{xf}),\ (P_x\xi)[V_f/f]) \Downarrow (\mathring{\mathsf{a}}(\xi_{xf}), G',\ \mathsf{n}_x)$	*Def.closure,* V_f*-value*
14	$\leftrightarrow \exists P_x.\ \mathring{\mathsf{a}}(P_x) = \emptyset \wedge [\![\mathbb{\Gamma}]\!]_\xi, f \vdash P_x : \mathsf{Nm}$ $\wedge\ (\mathring{\mathsf{a}}(\xi_{xf}),\ (P_x\xi)[V_f/f]) \Downarrow (\mathring{\mathsf{a}}(\xi_{xf}), G',\ \mathsf{n}_x)$ $\wedge\ \neg(\mathring{\mathsf{a}}(\xi_{xf}),\ (P_x\xi)[V_f/f]) \Downarrow (\mathring{\mathsf{a}}(\xi_{xf}), G',\ \mathsf{n}_x)$	$\mathsf{n}_x \notin \xi \to \mathsf{n}_x \notin \mathring{\mathsf{a}}(P_x\xi)$, $\mathsf{n}_x \in \mathring{\mathsf{a}}(V_f)$ *Lem. 84* $\to$ $\neg(\ldots \Downarrow \mathsf{n}_x)$
15	Contradiction, hence: $\neg \exists P_x.P_x \overset{[\mathbb{\Gamma}+f,\ \xi_{xf}]}{\rightsquigarrow} \mathsf{n}_x$	
16	$\to \xi_{xf} \models x\#\mathbb{\Gamma}{+}f$	*for both* n_x *cases, Sem.*$\#$
17	Hence: $\forall \xi_{xf}.\mathbb{\Gamma}{+}x{+}f \triangleright \xi_{xf} \to \xi_{xf} \models x\#\mathbb{\Gamma} \to x\#\mathbb{\Gamma}{+}f$	*Lines.1-16*

□

Soundness Proof of Axiom $(f4)$

$$y \notin \mathsf{fv}(e) \;\to\; (e\#\mathbb{\Gamma}_0 \wedge \forall y^{\alpha_y} \in (\mathbb{\Gamma}_0).A) \;\leftrightarrow\; \forall y^{\alpha_y} \in (\mathbb{\Gamma}_0).(e\#(\mathbb{\Gamma}_0{+}y:\alpha_y) \wedge A)$$

Proof. The $\leftarrow$ direction is elementary given $(f2)$, $(u2)$ and $(u3)$.

The $\rightarrow$ direction is proven as follows.

1	Let: $(f4) \equiv (e\#\mathbb{\Gamma}_0 \wedge \forall y^{\alpha_y} \in (\mathbb{\Gamma}_0).A) \;\to\; \forall y^{\alpha_y} \in (\mathbb{\Gamma}_0).(e\#(\mathbb{\Gamma}_0{+}y:\alpha_y) \wedge A)$	
2	Assume some $\mathbb{\Gamma}_1, \xi^{\mathbb{\Gamma}_1}$ s.t. $\mathbb{\Gamma}_1 \Vdash (f4)$	
3	Assume $\xi \models e\#\mathbb{\Gamma}_0 \wedge (\forall y \in (\mathbb{\Gamma}_0).A)$	
4	$\neg\, \exists\, M_e.M_e \overset{[\mathbb{\Gamma}_0,\ \xi]}{\rightsquigarrow} [\![e]\!]_\xi$ $\wedge\ \forall\, M_y, V_y.\ M_y \overset{[\mathbb{\Gamma}_0,\ \xi]}{\rightsquigarrow} V_y \;\to\; \xi \cdot y : V_y \models A$	*Sem.*$e\#\mathbb{\Gamma}_0$ *Sem.*$\forall y \in (\mathbb{\Gamma}_0).$
5	$\forall\, M_y, V_y.\ M_y \overset{[\mathbb{\Gamma}_0,\ \xi]}{\rightsquigarrow} V_y$ $\to\ \neg\, \exists\, M_e.M_e \overset{[\mathbb{\Gamma}_0,\ \xi]}{\rightsquigarrow} [\![e]\!]_\xi$ $\wedge\ \xi \cdot y : V_y \models A$	*F.O.L.* $A^{-x} \wedge \forall\, x.\ B \to C$ $\to\ \forall\, x.B \to (A \wedge C)$
6	$\forall\, M_y, V_y.\ M_y \overset{[\mathbb{\Gamma}_0,\ \xi]}{\rightsquigarrow} V_y$ $\to\ \neg\, \exists\, M_e.M_e \overset{[\mathbb{\Gamma}_0{+}y,\ \xi\cdot y:V_y]}{\rightsquigarrow} [\![e]\!]_{\xi\cdot y:V_y}$ $\wedge\ \xi \cdot y : V_y \models A$	*Lem.* 95 $s \overset{[\mathbb{\Gamma}_0,\ \xi]}{\rightsquigarrow} \leftrightarrow \overset{[\mathbb{\Gamma}_0{+}y,\ \xi_y]}{\rightsquigarrow} s$, *Lem.* 100, $[\![e]\!]_\xi \equiv [\![e]\!]_{\xi\cdot y:V_y}$
7	$\forall\, M_y, V_y.\ M_y \overset{[\mathbb{\Gamma}_0,\ \xi]}{\rightsquigarrow} V_y \;\to\; \xi \cdot y : V_y \models e\#\mathbb{\Gamma}{+}y \wedge \models A$	*Sem.*$\#, \wedge$
8	$\xi \models \forall y \in (\mathbb{\Gamma}_0).e\#(\mathbb{\Gamma}_0{+}y) \wedge A$	*Sem.*$\forall \in ().$
9	Hence: $\forall\, \mathbb{\Gamma}_1.\mathbb{\Gamma}_1 \Vdash (f4) \to \forall\, \xi.\mathbb{\Gamma}_1 \triangleright \xi \to \xi \models (f4)$	*Lines.1-8*

□

6.2.4 Soundness of Axioms for Universal Type Context Quantification

Soundness Proof of Axiom $(utc1)$

$$\mathbb{\Gamma} \Vdash (\forall \delta.A) \quad\to\quad A[\mathbb{\Gamma}/\delta]_{\mathbb{\Gamma}}$$

Proof.

1	Assume $\mathbb{\Gamma}$ s.t. $\mathbb{\Gamma} \Vdash (\forall \delta.A) \quad \rightarrow \quad A[\mathbb{\Gamma}/\delta]$	
2	Assume $\xi^{\mathbb{\Gamma}_d}$ s.t. $\mathbb{\Gamma} \triangleright \xi$ and $\xi \models \forall \delta.A$	
3	$\leftrightarrow \ \forall \xi_0^{\mathbb{\Gamma}_0}.\ \xi \preccurlyeq^\star \xi_0 \ \rightarrow \ \xi_0 \cdot \delta : \mathbb{\Gamma}_0 \models A$	*Sem.*$\forall\delta$.
4	$\rightarrow \ \xi \preccurlyeq^\star \xi \ \rightarrow \ \xi \cdot \delta : \mathbb{\Gamma}_d \models A$	*Instantiate* $\forall \xi_0^{\mathbb{\Gamma}_0}$ *with* ξ
5	$\rightarrow \ \xi \cdot \delta : \mathbb{\Gamma}_d \models A$	*F.O.L.* $(\xi \preccurlyeq^\star \xi) \quad (\equiv \mathsf{T})$
6	$\rightarrow \ \xi \cdot \delta : \mathbb{\Gamma} \models A$	$\mathbb{\Gamma} \triangleright \xi^{\mathbb{\Gamma}_d} \ \rightarrow \ [\![\mathbb{\Gamma}]\!]_\xi \equiv [\![\mathbb{\Gamma}_d]\!]_\xi$
7	$\rightarrow \ \xi \models A[\mathbb{\Gamma}/\delta]$	*Sem.*$[\mathbb{\Gamma}/\delta]$
8	Hence: $\forall \mathbb{\Gamma}.\ \mathbb{\Gamma} \Vdash (\forall \delta.A) \ \rightarrow \ A[\mathbb{\Gamma}/\delta]$ $\rightarrow \ \forall \xi^{\mathbb{\Gamma}_d}.\ \mathbb{\Gamma} \triangleright \xi \ \rightarrow \ \xi \models (\forall \delta.A) \ \rightarrow \ A[\mathbb{\Gamma}/\delta]$	*Lines.1-7*

□

Soundness Proof of Axiom (*utc*2)

$$A - \textsc{ExtInd}_{Syn} \ \rightarrow \ A^{-\delta} \ \leftrightarrow \ \forall \delta.A$$

This holds trivially given A-$\textsc{ExtInd}_{Syn}$ implies A-$\textsc{ExtInd}_{Sem}$ via Lem. 112. The full proof is included here.

Proof. Assume: $\mathbb{\Gamma}, \xi^{\mathbb{\Gamma}_d}$ s.t. $\mathbb{\Gamma} \triangleright \xi$ then assume $\mathbb{\Gamma} \Vdash A^{-\delta} \ \leftrightarrow \ \forall \delta.A$ for some $A^{-\delta}$-$\textsc{ExtInd}_{Sem}$, then:

$\leftarrow$: Use (*utc*1) knowing that $A^{-\delta}[\mathbb{\Gamma}/\delta] \equiv A$.

$\rightarrow$:

1	Assume $\xi \models A^{-\delta}$	
2	Prove: $\xi \models \forall \delta.A$	
3	$\xi \models A^{-\delta} \ \rightarrow \ \forall \xi'^{\mathbb{\Gamma}'}.\ \xi \preccurlyeq^\star \xi' \ \rightarrow \ \xi \models A^{-\delta}$	*Tautology*
4	$\xi \models A^{-\delta} \ \rightarrow \ \forall \xi'^{\mathbb{\Gamma}'}.\ \xi \preccurlyeq^\star \xi' \ \rightarrow \ \xi' \models A^{-\delta}$	A -$\textsc{ExtInd}_{Sem}$
5	$\xi \models A^{-\delta} \ \rightarrow \ \forall \xi'^{\mathbb{\Gamma}'}.\xi \preccurlyeq^\star \xi' \ \wedge \ \xi' \cdot \delta : \mathbb{\Gamma}' \models A^{-\delta}$	*Lem. 105*
6	$\xi \models A^{-\delta} \ \rightarrow \ \xi \models \forall \delta.A$	*Sem.*$\forall\delta$.

Hence $\forall \xi^{\mathbb{\Gamma}_d}, A.\ A$-$\textsc{ExtInd}_{Syn} \ \rightarrow \ \xi \models A^{-\delta} \leftrightarrow \forall \delta.A$. □

Soundness Proof of Axiom (*utc*3)

$$\forall\delta.(A \wedge B) \;\leftrightarrow\; (\forall\delta.A) \wedge (\forall\delta.B)$$

This holds trivially given F.O.L. but the full proof is included here.

Proof. Assume $\xi^{\mathbb{\Gamma}_d}$ then:

$\rightarrow$:

1	Assume $\xi \models \forall\delta.(A \wedge B)$	
2	$\forall\, \xi'^{\mathbb{\Gamma}'}.\ \xi \preccurlyeq^\star \xi' \rightarrow \xi' \cdot \delta : \mathbb{\Gamma}' \models A \wedge B$	*Sem.*$\forall\delta$.
3	$\leftrightarrow\ \forall\, \xi'^{\mathbb{\Gamma}'}.\ \xi \preccurlyeq^\star \xi' \rightarrow \xi' \cdot \delta : \mathbb{\Gamma}' \models A \wedge B$ $\wedge\ \forall\, \xi'^{\mathbb{\Gamma}'}.\ \xi \preccurlyeq^\star \xi' \rightarrow \xi' \cdot \delta : \mathbb{\Gamma}' \models A \wedge B$	*F.O.L.* $A \leftrightarrow (A \wedge A)$
4	$\rightarrow\ \forall\, \xi'^{\mathbb{\Gamma}'}.\ \xi \preccurlyeq^\star \xi' \rightarrow \xi' \cdot \delta : \mathbb{\Gamma}' \models A$ $\wedge\ \forall\, \xi'^{\mathbb{\Gamma}'}.\ \xi \preccurlyeq^\star \xi' \rightarrow \xi' \cdot \delta : \mathbb{\Gamma}' \models B$	$\wedge$-*elim*
5	$\leftrightarrow\ \xi \models (\forall\delta.A) \wedge (\forall\delta.B)$	*Sem.*$\forall\delta.$, $\wedge$

$\leftarrow$:

1	Assume $\xi \models (\forall\delta.A) \wedge (\forall\delta.B)$	
2	$\leftrightarrow\ \forall\, \xi'^{\mathbb{\Gamma}'}.\ \xi \preccurlyeq^\star \xi' \rightarrow \xi' \cdot \delta : \mathbb{\Gamma}' \models A$ $\wedge\ \forall\, \xi'^{\mathbb{\Gamma}'}.\ \xi \preccurlyeq^\star \xi' \rightarrow \xi' \cdot \delta : \mathbb{\Gamma}' \models B$	*Sem.* $\wedge$,$\forall$.
3	$\rightarrow\ \forall\, \xi'^{\mathbb{\Gamma}'}.\ \xi \preccurlyeq^\star \xi' \rightarrow \left(\begin{array}{l}\xi' \cdot \delta : \mathbb{\Gamma}' \models A \\ \wedge\ \xi' \cdot \delta : \mathbb{\Gamma}' \models B\end{array}\right)$	$\forall\, \xi'$ *unifing*
4	$\leftrightarrow\ \xi \models \forall\delta.(A \wedge B)$	*Sem.*$\wedge$, $\forall\delta$.

Hence $\forall\, \xi^{\mathbb{\Gamma}_d}.\ \xi \models \forall\delta.(A \wedge B) \leftrightarrow (\forall\delta.A) \wedge (\forall\delta.B)$. □

Soundness Proof of Axiom (*utc*4)

$$A\text{-}\textsc{ExtInd}_{Syn} \quad \rightarrow \quad \mathbb{\Gamma} \Vdash \forall x^{\mathsf{Nm}} \in (\mathbb{\Gamma}).A^{-\delta} \quad \leftrightarrow \quad \forall\delta.\forall x^{\mathsf{Nm}} \in (\mathbb{\Gamma}+\delta).A^{-\delta}$$

Proof. Clearly A-$\textsc{ExtInd}_{Syn}$ implies A-$\textsc{ExtInd}_{Sem}$ via Lem. 112.

$\leftarrow$: Holds through (*utc*1).

$\rightarrow$:

1	Let: $(utc4)_{\rightarrow} \equiv \forall x^{\mathsf{Nm}} \in (\mathbb{\Gamma}).A^{-\delta} \rightarrow \forall \delta.\forall x^{\mathsf{Nm}} \in (\mathbb{\Gamma}{+}\delta).A^{-\delta}$
2	Assume $\mathbb{\Gamma}$ s.t. $\mathbb{\Gamma} \Vdash (utc4)_{\leftarrow}$
3	Assume $\xi^{\mathbb{\Gamma}_d}$ s.t. $\mathbb{\Gamma} \triangleright \xi$ and $\xi \models \forall x^{\mathsf{Nm}} \in (\mathbb{\Gamma}).A$
4	$\leftrightarrow$ $\forall M, \mathsf{n}_0.\ M \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} \mathsf{n}_0 \rightarrow \xi \cdot x : \mathsf{n}_0 \models A$
5	Assume $\xi'^{\mathbb{\Gamma}'}$, M', n_1 s.t. $\xi \preccurlyeq^{\star} \xi'$ and $M' \overset{[\mathbb{\Gamma}+\delta,\ \xi'\cdot\delta:\mathbb{\Gamma}'\setminus_{-TCV}]}{\rightsquigarrow} \mathsf{n}_1$ — *" $\forall\delta.\forall x \in (\delta)."$*
6	$\leftrightarrow$ $M' \overset{[\mathbb{\Gamma}',\ \xi']}{\rightsquigarrow} \mathsf{n}_1$ — $[\![\mathbb{\Gamma}{+}\delta]\!]_{\xi'\cdot\delta:\mathbb{\Gamma}'\setminus_{-TCV}} \equiv \mathbb{\Gamma}'$, *Lem. 102*
7	$\mathsf{n}_1 \in \mathring{\mathsf{a}}(\xi) \rightarrow \mathsf{n}_0 \equiv \mathsf{n}_1 \rightarrow \xi \cdot x : \mathsf{n}_1 \models A$ — *Lem. 96* $\rightarrow$ $\mathsf{n}_0 \equiv \mathsf{n}_1$ *obtainable*
8	$\mathsf{n}_1 \notin \mathring{\mathsf{a}}(\xi) \rightarrow$ fresh-$\mathsf{n}_0 \equiv \mathsf{n}_1 \rightarrow \xi \cdot x : \mathsf{n}_1 \models A$ — *Let* $\mathsf{n}_0 \equiv \mathsf{n}_1$ *as* $\mathsf{n}_1 \notin \mathring{\mathsf{a}}(\xi)$ *(fresh names can be swapped)*
9	$\rightarrow$ $\xi' \cdot x : \mathsf{n}_1 \models A$ — *Lem. 111, Lem. 108,* A-ExtInd$_{Sem}$
10	$\rightarrow$ $\xi' \cdot \delta : \mathbb{\Gamma}'\setminus_{-TCV} \cdot x : \mathsf{n}_1 \models A$ — *Lem. 105,* $A^{-\delta}$
11	$\rightarrow$ $\xi \models \forall\delta.\forall x^{\mathsf{Nm}} \in (\mathbb{\Gamma}{+}\delta).A^{-\delta}$ — *Lines.4-9*
12	Hence: $\forall\, \mathbb{\Gamma}.\ \mathbb{\Gamma} \Vdash (utc4)_{\rightarrow} \rightarrow \forall\, \xi^{\mathbb{\Gamma}_d}.\mathbb{\Gamma} \triangleright \xi \rightarrow \xi \models (utc4)_{\rightarrow}$ (Assuming A-ExtInd$_{Sem}$) — *Lines.1-10*

Essentially in line 7: if n_1 is in $\mathring{\mathsf{a}}(\xi)$ then it can be derived from ξ, and in line 8 if n_1 is not in $\mathring{\mathsf{a}}(\xi)$ then instantiating $M \equiv \mathsf{gensym}()$ produces a fresh name which can easily be set as (or swapped for) n_1 using Lem. 111 and Lem. 108.

□

Soundness Proof of Axiom $(utc5)$

$$\mathbb{\Gamma} \Vdash \forall f^{\alpha} \in (\mathbb{\Gamma}).a \bullet f = c \;\rightarrow\; \forall \delta.\forall f^{\alpha} \in (\delta).a \bullet f = c \qquad c \in \{\mathsf{true}, \mathsf{false}\}$$

Proof. This is the proof for $c \equiv \mathsf{false}$, however the proof holds for $c \equiv \mathsf{true}$ via symmetry.

1	Assume $\xi^{\mathbb{\Gamma}_d}$. $\mathbb{\Gamma} \triangleright \xi \;\wedge\; \xi \models \forall f^{\alpha} \in (\mathbb{\Gamma}).a \bullet f = \mathsf{false}$	
2	Assume $\xi'^{\mathbb{\Gamma}'}$ s.t. $\xi \preccurlyeq^{\star} \xi'$ then prove $\xi' \models \forall f^{\alpha} \in (\mathbb{\Gamma}').a \bullet f = \mathsf{false}$	
3	Assume false then proof by contradiction $\xi' \models \exists f^{\alpha} \in (\mathbb{\Gamma}').a \bullet f = \mathsf{true}$	
4	i.e. $\exists M_f.\ M_f \overset{[\mathbb{\Gamma}',\ \xi']}{\rightsquigarrow} V_f \;\wedge\; af \overset{[\mathbb{\Gamma}'+f,\ \xi'\cdot f:V_f]}{\rightsquigarrow} \mathsf{true}$	
5	$\rightarrow \exists M_f.\ M_f[M_1/x_1] \overset{[\mathbb{\Gamma}'',\ \xi'']}{\rightsquigarrow} V_f \;\wedge\; af \overset{[\mathbb{\Gamma}''+f,\ \xi''\cdot f:V_f]}{\rightsquigarrow} \mathsf{true}$	$\xi' \equiv \xi''^{\mathbb{\Gamma}''} \cdot x_1 : V_1$ $M_1 \overset{[\mathbb{\Gamma}'',\ \xi'']}{\rightsquigarrow} V_1$
6	$\rightarrow$ ...	*The TCV mappings have no effect on the evaluation of* af
7	$\rightarrow \exists M_f.\ M_f[M_1/x_1]...[M_k/x_k] \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_f \;\wedge\; af \overset{[\mathbb{\Gamma}+f,\ \xi'\cdot f:V_f]}{\rightsquigarrow} \mathsf{true}$	$\xi' \equiv \xi \cdot \tilde{x} : \tilde{V}$ $M_k \overset{[\mathbb{\Gamma}+\ldots,\ \xi,\ \ldots]}{\rightsquigarrow} V_k$
8	$\rightarrow \exists N_f.\ N_f \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_f \;\wedge\; af \overset{[\mathbb{\Gamma}+f,\ \xi'\cdot f:V_f]}{\rightsquigarrow} \mathsf{true}$	$N_f \equiv M_f[M_1/x_1][M_k/x_k]$
9	Hence contradiction with Line.2	
10	Hence axiom holds	

□

This works due to $\xi \preccurlyeq^{\star} \xi'$ requiring the values in $\xi' \setminus \xi$ being derived from ξ in an orderly manner. Each value being derived from a term ensures that term can be used to reconstruct V_f from ξ.

The axiom fails to hold in a more general form i.e. $\mathbb{\Gamma} \Vdash \forall f^{\alpha} \in (\mathbb{\Gamma}).A \;\rightarrow\; \forall \delta.\forall f^{\alpha} \in (\delta).A$ even if A-ExtInd$_{Syn}$ but the restricted form of A in this case provides enough constraints to ensure this holds. There may be more general forms of this axiom that hold but are not currently needed in the reasoning examples.

6.2.5 Soundness of Axioms for Evaluation Formulae

Many of these axioms are adapted from the program logic for the STLC as seen in Sec. 2.2.2, and the soundness proofs are included to show their validity under the new model construction and semantics.

Soundness Proof of Axiom (ext)

For all $e_1, e_2 : \alpha_1 \to \alpha_2$ s.t. α_1 and α_2 in $\alpha_{\text{-Nm}}$:

$$(\forall x^{\alpha_1} \in (\emptyset).e_1 \bullet x = m_1^{\alpha_2}\{e_2 \bullet x = m_2^{\alpha_2}\{m_1 = m_2\}\}) \;\leftrightarrow\; e_1 =^{\alpha_1 \to \alpha_2} e_2$$

Proof. Use Lem. 87 to see that any ν_{GS}-calculus term of type $\alpha_{\text{-Nm}}$ has an equivalent STLC (name-free) term hence the proof for (ext) in the STLC holds here too.

□

Soundness Proof of Axiom (e_α)

$$e \bullet e' = m\{e \bullet e' = a\{a = m \wedge A\}\} \leftrightarrow e \bullet e' = m^{\alpha}\{A\} \qquad \alpha \in \alpha_{\text{-Nm}},\; a \notin \mathsf{fv}(e, e'), a \notin \mathsf{fv}(A)$$

Proof. Given $\alpha \in \alpha_{\text{-Nm}}$ and Lem. 87 clearly the value at m is equivalent to a name-free value equivalent to the value at a hence this holds.

□

This fails to hold for all types given ee' equivalent to $\mathsf{gensym}()$ producing two fresh names at m_1 and m_2 which are not congruent.

Soundness Proof of Axiom $(e1)$

$$e \bullet e' = m\{A \wedge B\} \;\leftrightarrow\; e \bullet e' = m\{A\} \wedge e \bullet e' = m\{B\}$$

Proof.

1	Let: $(e1) \equiv e \bullet e' = m\{A \wedge B\} \;\leftrightarrow\; e \bullet e' = m\{A\} \wedge e \bullet e' = m\{B\}$	
2	Assume $\xi^{\mathbb{\Gamma}_d}$	
3	$\xi \models e \bullet e' = m\{A \wedge B\}$	
4	$\leftrightarrow\; \exists V.\, ee' \overset{[\mathbb{\Gamma},\, \xi]}{\rightsquigarrow} V \;\wedge\; \xi \cdot m : V \models A \;\wedge\; \xi \cdot m : V \models B$	*Sem.*, $\bullet = \{\}$, $\wedge$
5	$\leftrightarrow\; \exists V.\, ee' \overset{[\mathbb{\Gamma},\, \xi]}{\rightsquigarrow} V \;\wedge\; \xi \cdot m : V \models A$ $\wedge\; \exists V.\, ee' \overset{[\mathbb{\Gamma},\, \xi]}{\rightsquigarrow} V \;\wedge\; \xi \cdot m : V \models B$	*F.O.L., nominal determinacy Def. 25*
6	$\leftrightarrow\; \xi \models e \bullet e' = m\{A\} \;\wedge\; \xi \models e \bullet e' = m\{A\}$	*Sem.*$\bullet = \{\}$, $\wedge$
7	Hence: $\forall \xi^{\mathbb{\Gamma}_d}.\, \xi \models (e1)$	

The line 5 is guaranteed as termination is guaranteed and the fresh names can be chosen to be identical. □

Soundness Proof of Axiom $(e2)$

$$e \bullet e' = m\{\neg A\} \;\leftrightarrow\; \neg e \bullet e' = m\{A\}$$

Proof.

1	Assume $\xi^{\mathbb{\Gamma}_d}$ s.t.	
2	$\xi \models e \bullet e' = m\{\neg A\}$	
3	$\leftrightarrow\ \exists V.\, ee' \overset{[\mathbb{\Gamma},\,\xi]}{\rightsquigarrow} V \ \wedge\ \xi \cdot m : V \models \neg A$	*Sem* $\bullet = \{\}$
4	$\leftrightarrow\ \exists V.\, ee' \overset{[\mathbb{\Gamma},\,\xi]}{\rightsquigarrow} V \ \wedge\ \neg\, \xi \cdot m : V \models A$	*Sem* $\neg$
5	$\leftrightarrow\ \neg\ \exists V.\, ee' \overset{[\mathbb{\Gamma},\,\xi]}{\rightsquigarrow} V \ \wedge\ \xi \cdot m : V \models A$	*Termination Guaranteed*
6	$\leftrightarrow\ \xi \models \neg e \bullet e' = m\{A\}$	*Sem.*$\bullet = \{\}, \neg$
7	Hence: $\forall\, \xi^{\mathbb{\Gamma}_d}.\ \xi \models (e2)$	

□

Soundness Proof of Axiom $(e3)$

$$m \notin \mathsf{fv}(A) \wedge A\text{-ExtInd}_{Syn} \quad \rightarrow \quad e_1 \bullet e_2 = m\{A^{-m} \wedge B\} \;\leftrightarrow\; (A \wedge e_1 \bullet e_2 = m\{B\})$$

Proof. Clearly A-ExtInd$_{Syn}$ implies A-ExtInd$_{Sem}$ via Lem. 112.

1	Let: $(e3) \equiv e_1 \bullet e_2 = m\{A^{-m} \wedge B\} \;\leftrightarrow\; (A \wedge e_1 \bullet e_2 = m\{B\})$	
2	Assume $\xi^{\mathbb{\Gamma}_d}$ s.t.	
3	$\xi \models e_1 \bullet e_2 = m\{A^{-m} \wedge B\}$	$m \notin \mathsf{dom}(\mathbb{\Gamma})$
4	$\leftrightarrow\ \exists V.\, e_1 e_2 \overset{[\mathbb{\Gamma},\,\xi]}{\rightsquigarrow} V \ \wedge\ \xi \cdot m : V \models A \ \wedge\ \xi \cdot m : V \models B$	*Sem.*$\bullet = \{\}, \wedge$
5	$\leftrightarrow\ \exists V.\, e_1 e_2 \overset{[\mathbb{\Gamma},\,\xi]}{\rightsquigarrow} V \ \wedge\ \xi \models A \ \wedge\ \xi \cdot m : V \models B$	*Sem.*$\preccurlyeq^\star$, *Lem. 93*, A-ExtInd$_{Sem}$
6	$\leftrightarrow\ \xi \models (A \wedge e_1 \bullet e_2 = m\{B\})$	*Sem.*$\bullet = \{\}, \wedge$
7	Hence: $\forall\, \xi^{\mathbb{\Gamma}_d}.\ A\text{-ExtInd}_{Syn} \wedge m \notin \mathsf{fv}(A) \rightarrow \xi \models (e3)$	

□

Soundness Proof of Axiom (*e*4)

$m \notin \mathsf{fv}(\mathbb{\Gamma}) \ \wedge \ x \notin \mathsf{fv}(e_1, e_2, m) \ \rightarrow \ e_1 \bullet e_2 = m\{\forall x \in (\mathbb{\Gamma}).A\} \ \leftrightarrow \ \forall x \in (\mathbb{\Gamma}).e_1 \bullet e_2 = m\{A\}$

The typing of this axiom implies $x \notin \mathsf{fv}(e_1, e_2, m)$ and $m \notin \mathsf{fv}(\mathbb{\Gamma})$.

Proof.

1	Assume $\xi^{\mathbb{\Gamma}_d}$	
2	$\xi \models \forall x \in (\mathbb{\Gamma}).e_1 \bullet e_2 = m\{A\}$	
3	$\leftrightarrow \ \forall M_x, V_x.\ M_x \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_x$ $\rightarrow \ \exists V_m.\ (e_1 e_2 \overset{[\mathbb{\Gamma},\ \xi.x:V_x]}{\rightsquigarrow} V_m \ \wedge \ \xi \cdot m : V_m \cdot x : V_x \models A)$	*Sem.*$\forall \in ().,\ \bullet = \{\}$
4	$\leftrightarrow \ \forall M_x, V_x.\ M_x \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_x$ $\rightarrow \ \exists V_m.\ (e_1 e_2 \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_m \ \wedge \ \xi \cdot m : V_m \cdot x : V_x \models A)$	$x \notin \mathsf{fv}(e_1, e_2, m)$, *Lem.* 102 $(\mathring{a}(V_x) \cap \mathring{a}(V_m) \subseteq \mathring{a}(\xi \backslash x))$
5	$\leftrightarrow \ \exists V'_m.\ e_1 e_2 \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V'_m$ $\wedge \ \forall M_x, V_x.\ M_x \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_x$ $\rightarrow \ \exists V_m.\ e_1 e_2 \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_m$ $\wedge \ V_m \cong_\alpha^{\mathring{a}(\xi, V_m, V'_m)} V'_m$ $\wedge \ \xi \cdot m : V_m \cdot x : V_x \models A$	$V_m \cong_\alpha^{\cdots} V'_m$
6	$\leftrightarrow \ \exists V'_m.\ e_1 e_2 \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V'_m$ $\wedge \ \forall M_x, V_x.\ M_x \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_x$ $\rightarrow \ \exists V_m.\ e_1 e_2 \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_m$ $\wedge \ V_m \cong_\alpha^{\mathring{a}(\xi, V_m, V'_m)} V'_m$ $\wedge \ \xi \cdot m : V'_m \cdot x : V_x \models A$	*Lem.* 108
7	$\leftrightarrow \ \exists V'_m.\ e_1 e_2 \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V'_m$ $\wedge \ \forall M_x, V_x.\ M_x \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_x \ \rightarrow \ \xi \cdot m : V'_m \cdot x : V_x \models A$	$V_m \cong_\alpha^{\cdots} V'_m$
8	$\xi \models e_1 \bullet e_2 = m\{\forall x \in (\mathbb{\Gamma}).A\}$	*Sem.*$\forall \in ().,\ \bullet = \{\}$

□

Soundness Proof of Axiom ($e5$)

$$A\text{-}\textsc{ExtInd}_{Syn} \;\to\; e_1 \bullet e_2 = m^{\alpha\text{-}(\mathsf{Nm},\to)}\{\forall\delta.A\} \;\leftrightarrow\; \forall\delta.e_1 \bullet e_2 = m^{\alpha\text{-}(\mathsf{Nm},\to)}\{A\}$$

Proof. Clearly A-$\textsc{ExtInd}_{Syn}$ implies A-$\textsc{ExtInd}_{Sem}$ via Lem. 112.

1	Assume $\xi^{\mathbb{\Gamma}_d}$	
2	Assume $\xi \models e_1 \bullet e_2 = m^{\alpha\text{-}(\mathsf{Nm},\to)}\{\forall\delta.A\}$	
3	$\leftrightarrow\ \exists V_m.\ e_1e_2 \overset{[\mathbb{\Gamma},\,\xi]}{\rightsquigarrow} V_m$ $\wedge\ \forall \xi_m'^{\mathbb{\Gamma}'_m}.\ \xi\cdot m : V_m \preccurlyeq^\star \xi_m' \ \to\ \xi_m'\cdot\delta : \mathbb{\Gamma}'_m\backslash_{-TCV} \models A$	$Sem.\bullet = \{\}$, $\forall\delta.$
4	$\leftrightarrow\ \exists V_m.\ e_1e_2 \overset{[\mathbb{\Gamma},\,\xi]}{\rightsquigarrow} V_m$ $\wedge\ \forall \xi'^{\mathbb{\Gamma}'}.\ \xi\cdot m : V_m \preccurlyeq^\star \xi'\cdot m : V_m$ $\to\ \xi'\cdot m : V_m\cdot\delta : \mathbb{\Gamma}'\backslash_{-TCV} \models A$	*Rewrite* ξ'_m *Lem. 82*
5	$\leftrightarrow\ \exists V_m.\ e_1e_2 \overset{[\mathbb{\Gamma},\,\xi]}{\rightsquigarrow} V_m$ $\wedge\ \forall \xi'^{\mathbb{\Gamma}'}.\ \xi \preccurlyeq^\star \xi' \ \to\ \xi'\cdot m : V_m\cdot\delta : \mathbb{\Gamma}'\backslash_{-TCV} \models A$	*Lem. 83*
6	$\leftrightarrow\ \exists V_m.\ e_1e_2 \overset{[\mathbb{\Gamma},\,\xi]}{\rightsquigarrow} V_m$ $\wedge\ \forall \xi'^{\mathbb{\Gamma}'}.\ \xi \preccurlyeq^\star \xi' \ \to\ \exists V_m'.\ e_1e_2 \overset{[\mathbb{\Gamma},\,\xi]}{\rightsquigarrow} V_m' \ \wedge\ V_m \cong^{\mathring{\mathsf{a}}(\xi)}_{\alpha\text{-}(\mathsf{Nm},\to)} V_m'$ $\wedge\ \xi'\cdot m : V_m\cdot\delta : \mathbb{\Gamma}'\backslash_{-TCV} \models A$	$V_m \cong^{\dots}_{\alpha\text{-}(\mathsf{Nm},\to)} V_m'$
7	$\leftrightarrow\ \exists V_m.\ e_1e_2 \overset{[\mathbb{\Gamma},\,\xi]}{\rightsquigarrow} V_m$ $\wedge\ \forall \xi'^{\mathbb{\Gamma}'}.\ \xi \preccurlyeq^\star \xi' \ \to\ \exists V_m'.\ e_1e_2 \overset{[\mathbb{\Gamma},\,\xi]}{\rightsquigarrow} V_m' \ \wedge\ V_m \cong^{\mathring{\mathsf{a}}(\xi)}_{\alpha\text{-}(\mathsf{Nm},\to)} V_m'$ $\wedge\ \xi'\cdot m : V_m'\cdot\delta : \mathbb{\Gamma}'\backslash_{-TCV} \models A$	*Lem. 108*
8	$\leftrightarrow\ \forall \xi'^{\mathbb{\Gamma}'}.\ \xi \preccurlyeq^\star \xi'$ $\to\ \exists V_m'.\ e_1e_2 \overset{[\mathbb{\Gamma},\,\xi]}{\rightsquigarrow} V_m' \ \wedge\ \xi'\cdot m : V_m'\cdot\delta : \mathbb{\Gamma}'\backslash_{-TCV} \models A$	$V_m \cong^{\dots}_{\alpha\text{-}(\mathsf{Nm},\to)} V_m'$
9	$\leftrightarrow\ \forall \xi'^{\mathbb{\Gamma}'}.\ \xi \preccurlyeq^\star \xi'$ $\to\ \exists V_m'.\ (e_1e_2 \overset{[\mathbb{\Gamma},\ \xi'\cdot\delta:\mathbb{\Gamma}'\backslash_{-TCV}]}{\rightsquigarrow} V_m' \ \wedge\ \xi'\cdot\delta : \mathbb{\Gamma}'\backslash_{-TCV}\cdot m : V_m' \models A)$	*Lem. 102*
10	$\leftrightarrow\ \forall \xi'^{\mathbb{\Gamma}'}.\ \xi \preccurlyeq^\star \xi'$ $\to\ \exists V_m'.\ (e_1e_2 \overset{[\mathbb{\Gamma}'+\delta,\ \xi'\cdot\delta:\mathbb{\Gamma}'\backslash_{-TCV}]}{\rightsquigarrow} V_m' \ \wedge\ \xi'\cdot\delta : \mathbb{\Gamma}'\backslash_{-TCV}\cdot m : V_m' \models A)$	$\mathbb{\Gamma} \Vdash e_1e_2 : \alpha_{\text{-}(\mathsf{Nm},\to)}$
11	$\leftrightarrow\ \xi \models \forall\delta.e_1 \bullet e_2 = m^{\alpha\text{-}(\mathsf{Nm},\to)}\{A\}$	$Sem.\bullet = \{\}$, $\forall\delta.$

□

6.3 Soundness of Rules

In this section, the soundness of the rules is proven.

Given the logic is limited to static syntax, the condition then for any term s.t. $\mathbb{\Gamma} \vdash M : \alpha$ where $\mathring{a}(M) = \emptyset$ is implied then clearly $(\mathring{a}(\xi), M\xi) \Downarrow (\mathring{a}(\xi), G',\ V) \leftrightarrow M \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V$ hence this will be used equivalently for brevity. Throughout the proofs it is assumed that there exists a value V_i that a term M_i reduces to given the guaranteed termination of well typed terms in the language hence this $\exists\, V_i$. ... is often dropped for brevity.

Both Lem. 113 and Lem. 112 ensure that for any A then A-$\textsc{Thin}_{Syn}(x)$ implies A-$\textsc{Thin}_{Sem}(x)$ and A-$\textsc{ExtInd}_{Syn}$ implies A-$\textsc{ExtInd}_{Sem}$ respectively and are both used at each time the syntactic definition is used.

The soundness proofs are split into the three separate sections in which they are introduced. The core rules are proven sound in Sec. 6.3.1, with the structural rules proven in Sec. 6.3.2, and finally a derivation from these rules proves the Derived rules sound in Sec. 6.3.3. Let "Op. Sem." refer to operational semantics of the ν_{GS}-calculus in Sec. 3.1.1.

6.3.1 Soundness of Core Rules

Soundness of $[\textsc{Var}]_\nu$

$$\frac{-}{\{A[x/m]\}\ x :_m \{A\}}\ [\textsc{Var}]_\nu$$

Proof.

1	Assume $\mathbb{\Gamma}$ s.t. $\mathbb{\Gamma} \Vdash \{A[x/m]\}\ x :_m \{A\}$ s.t.	
2	Assume ξ s.t. $\mathbb{\Gamma} \triangleright \xi$ and $\xi \models A[x/m]$	
3	$\xi \models A[x/m] \leftrightarrow \xi \cdot m : \xi(x) \models A$	*Sem.*$[x/m]$, *m-fresh*
4	$\xi \models A[x/m] \rightarrow x \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} \xi(x) \wedge \xi \cdot m : \xi(x) \models A$	$\xi(x)$ *a value*
5	$\forall\, \xi.\mathbb{\Gamma} \triangleright \xi \rightarrow \xi \models A[x/m] \rightarrow x \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} \xi(x) \wedge \xi \cdot m : \xi(x) \models A$	*Lines.2, 4*
6	$\forall\, \mathbb{\Gamma}.\ \mathbb{\Gamma} \Vdash \{A[x/m]\}\ x :_m \{A\} \rightarrow \forall\, \xi.\mathbb{\Gamma} \triangleright \xi \rightarrow \xi \models \{A[x/m]\}\ x :_m \{A\}$	*Lines.1,5*
7	Hence: $\dfrac{-}{\models \{A[x/m]\}\ x :_m \{A\}}$	

□

Soundness of $[\textsc{Const}]_\nu$

$$\frac{-}{\{A[c/m]\}\ c :_m \{A\}}\ [\textsc{Const}]_\nu$$

Proof. See proof for $[\textsc{Var}]_\nu$ and replace x with c. □

Soundness of $[\textsc{Eq}]_\nu$

$$\frac{\{A\}\ M :_m \{B\} \quad \{B\}\ N :_n \{C[m = n/u]\} \quad C\ \textsc{Thin}_{Syn}(m,n)}{\{A\}\ M = N :_u \{C\}}\ [\textsc{Eq}]_\nu$$

Proof. Clearly Lem. 113 ensures $A\text{-}\textsc{Thin}_{Syn}(x)$ implies $A\text{-}\textsc{Thin}_{Sem}(x)$.

1	Assume $I.H.(1):\ \forall\, \mathbb{\Gamma}_0, \xi_1^{\mathbb{\Gamma}_1}.\ \mathbb{\Gamma}_0 \Vdash \{A\}\ M :_m \{B\} \ \wedge\ \mathbb{\Gamma}_0 \triangleright \xi_1 \rightarrow \xi_1 \models \{A\}\ M :_m \{B\}$	
2	Assume $I.H.(2):\ \forall\, \mathbb{\Gamma}_0, \xi_1^{\mathbb{\Gamma}_1}.\ \mathbb{\Gamma}_0 \Vdash \{B\}\ N :_n \{C[m = n/u]\} \ \wedge\ \mathbb{\Gamma}_0 \triangleright \xi_1 \rightarrow \xi_1 \models \{B\}\ N :_n \{C[m = n/u]\}$	
3	Assume $\mathbb{\Gamma}$ s.t. $\mathbb{\Gamma} \Vdash \{A\}\ M = N :_u \{C\}$	
4	Assume ξ s,t, $\mathbb{\Gamma} \triangleright \xi \ \wedge\ \xi \models A$	
5	$\rightarrow\ (\mathfrak{a}(\xi),\ M\xi) \Downarrow (\mathfrak{a}(\xi), G_m,\ V_m) \ \wedge\ \xi \cdot m : V_m \models B$	*I.H.(1)*
6	$\rightarrow\ M \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_m \ \wedge\ N \overset{[\mathbb{\Gamma},\ \xi \cdot m:V_m]}{\rightsquigarrow} V_n \ \wedge\ \xi \cdot m : V_m \cdot n : V_n \models C[m = n/u]$	*I.H.(2)*
7	$\rightarrow\ M \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_m \ \wedge\ N \overset{[\mathbb{\Gamma},\ \xi \cdot m:V_m]}{\rightsquigarrow} V_n \ \wedge\ (m = n) \overset{[\mathbb{\Gamma}+m+n,\ \xi \cdot m:V_m \cdot n:V_n]}{\rightsquigarrow} V_u \ \wedge\ \xi_{mn} \cdot u : V_u \models C$	*Sem.*$C[m = n/u]$
8	$\rightarrow\ M \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_m \ \wedge\ N \overset{[\mathbb{\Gamma},\ \xi \cdot m:V_m]}{\rightsquigarrow} V_n \ \wedge\ (m = n) \overset{[\mathbb{\Gamma}+m+n,\ \xi \cdot m:V_m \cdot n:V_n]}{\rightsquigarrow} V_u \ \wedge\ \xi \cdot u : V_u \models C$	$C\ \textsc{Thin}_{Sem}(m,n)$
9	$\rightarrow\ (M = N) \overset{[\mathbb{\Gamma}+m+n,\ \xi \cdot m:V_m \cdot n:V_n]}{\rightsquigarrow} V_u \ \wedge\ \xi \cdot u : V_u \models C$	*Op. Sem.*$(=)$
10	$\rightarrow\ \forall\, \xi.\mathbb{\Gamma} \triangleright \xi \rightarrow \xi \models \{A\}\ M = N :_u \{C\}$	*Lines.4-9*
11	Hence: $\dfrac{\models \{A\}\ M :_m \{B\} \qquad \models \{B\}\ N :_n \{C^{-m,n}[m = n/u]\}}{\models \{A\}\ M = N :_u \{C\}}$ ($C\ \textsc{Thin}_{Syn}(m,n)$)	*Lines.1-10*

A common shorthand abbreviation is used here in the form of $m = n$ as an expression which is substituted for the variable u in $C[m = n/u]$. Although this isn't strictly legal

it is obvious this means where $m = n$ appears in a formula it is substituted for $u = \mathsf{true}$. A formalisation of this has not been written out in full but is often used in the literature without explicit formalisation [28, 72, 5]. □

Soundness of $[\textsc{GenSym}]_\nu$

$$\frac{-}{\{\mathsf{T}\}\ \mathsf{gensym} :_u \{\forall \delta.u \bullet () = m\{m\#\delta\}\}}\ [\textsc{GenSym}]_\nu$$

Proof.

1	Assume $\mathbb{\Gamma}$ s.t. $\mathbb{\Gamma} \Vdash \{\mathsf{T}\}\ \mathsf{gensym} :_u \{\forall \delta.u \bullet () = m\{m\#\delta\}\}$
2	Assume ξ s.t. $\mathbb{\Gamma} \triangleright \xi \ \wedge\ \xi \models \mathsf{T}$
3	Let: $\xi_u^{\mathbb{\Gamma}_u} \equiv \xi \cdot u : \mathsf{gensym} \ \wedge\ \xi_{1d} \equiv \xi_1 \cdot \delta : \mathbb{\Gamma}_1 \backslash_{-TCV}$ $\quad$ gensym-*value and* $u \notin \mathsf{dom}(\xi)$
4	$\forall\, \xi_x^{\mathbb{\Gamma}_x}, \mathsf{n}.\ (\mathring{\mathsf{a}}(\xi_x),\ \mathsf{gensym}()) \Downarrow ((\mathring{\mathsf{a}}(\xi_x), \mathsf{n}),\ \mathsf{n}) \ \wedge\ \mathsf{n} \notin \mathring{\mathsf{a}}(\xi_x)$ $\quad$ *Op. Sem.*gensym()
5	$\rightarrow\ \forall\, \xi_1^{\mathbb{\Gamma}_1}.\ \xi_u \preccurlyeq^\star \xi_1 \rightarrow\ \exists\, \mathsf{n}.\ u() \overset{[\mathbb{\Gamma}+u,\ \xi_1]}{\rightsquigarrow} \mathsf{n} \wedge\ \mathsf{n} \notin \mathring{\mathsf{a}}(\xi_1)$ $\quad$ *restrict* $\forall\, \xi_x$ *to* $\xi_u \preccurlyeq^\star \xi_x$
6	$\forall\, \xi_1^{\mathbb{\Gamma}_1}.\ \xi_u \preccurlyeq^\star \xi_1 \rightarrow\ \exists\, \mathsf{n}.u() \overset{[\mathbb{\Gamma}+u,\ \xi_1]}{\rightsquigarrow} \mathsf{n} \wedge \neg\ \exists\, M_m^{\mathsf{Nm}}.M_m \overset{[\mathbb{\Gamma}_1,\ \xi_1 \cdot m:\mathsf{n}]}{\rightsquigarrow} \mathsf{n}$ $\quad$ *Lem. 98*
7	$\forall\, \xi_1^{\mathbb{\Gamma}_1}.\ \xi_u \preccurlyeq^\star \xi_1$ $\quad$ $\mathsf{n} \equiv [\![m]\!]_{\xi_{1d} \cdot m:\mathsf{n}}$ $\rightarrow\ \exists\, \mathsf{n}.\ u() \overset{[\mathbb{\Gamma}+u,\ \xi_1]}{\rightsquigarrow} \mathsf{n} \wedge \neg\ \exists\, M_m^{\mathsf{Nm}}.M_m \overset{[\mathbb{\Gamma}_1,\ \xi_1 \cdot m:\mathsf{n}]}{\rightsquigarrow} [\![m]\!]_{\xi_{1d} \cdot m:\mathsf{n}}$
8	$\forall\, \xi_1^{\mathbb{\Gamma}_1}.\ \xi_u \preccurlyeq^\star \xi_1$ $\quad$ *Lem. 102* $\rightarrow\ \exists\, \mathsf{n}.\ u() \overset{[\mathbb{\Gamma}+u,\ \xi_1]}{\rightsquigarrow} \mathsf{n} \wedge \neg\ \exists\, M_m^{\mathsf{Nm}}.M_m \overset{[\mathbb{\Gamma}_1,\ \xi_{1d} \cdot m:\mathsf{n}]}{\rightsquigarrow} [\![m]\!]_{\xi_{1d} \cdot m:\mathsf{n}}$
9	$\forall\, \xi_1^{\mathbb{\Gamma}_1}.\ \xi_u \preccurlyeq^\star \xi_1$ $\quad$ $[\![\mathbb{\Gamma}_1]\!]_{\xi_{1d}} \equiv [\![\mathbb{\Gamma}_1 + \delta]\!]_{\xi_{1d}}$ $\rightarrow\ \exists\, \mathsf{n}.\ u() \overset{[\mathbb{\Gamma}+u,\ \xi_1]}{\rightsquigarrow} \mathsf{n} \wedge \neg\ \exists\, M_m^{\mathsf{Nm}}.M_m \overset{[\mathbb{\Gamma}_1+\delta,\ \xi_{1d} \cdot m:\mathsf{n}]}{\rightsquigarrow} [\![m]\!]_{\xi_{1d} \cdot m:\mathsf{n}}$
10	$\forall\, \xi_1^{\mathbb{\Gamma}_1}.\ \xi_u \preccurlyeq^\star \xi_1 \rightarrow\ \exists\, \mathsf{n}.\ u() \overset{[\mathbb{\Gamma}+u,\ \xi_1]}{\rightsquigarrow} \mathsf{n} \wedge\ \xi_{1d} \cdot m : \mathsf{n} \models m\#\delta$ $\quad$ *Sem.*$\#\delta$
11	$\forall\, \xi_1^{\mathbb{\Gamma}_1}.\ \xi_u \preccurlyeq^\star \xi_1 \rightarrow\ \exists\, \mathsf{n}.\ u() \overset{[\mathbb{\Gamma}+u,\ \xi_{1d}]}{\rightsquigarrow} \mathsf{n} \rightarrow\ \xi_{1d} \cdot m : \mathsf{n} \models m\#\delta$ $\quad$ *Lem. 102*
12	$\xi \cdot u : \mathsf{gensym} \models \forall \delta.u \bullet () = m\{m\#\delta\}$ $\quad$ *Sem.*$\forall \delta.u \bullet () = m\{m\#\delta\}$
13	$\forall\, \xi^{\mathbb{\Gamma}}.\mathbb{\Gamma} \triangleright \xi \rightarrow\ \xi \models \mathsf{T} \rightarrow\ \mathsf{gensym} \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} \mathsf{gensym}$ $\quad$ *Lines.2-12* $\wedge\ \xi \cdot u : \mathsf{gensym} \models \forall \delta.u \bullet () = m\{m\#\mathbb{\Gamma}+u+\delta\}$
14	Hence: $\dfrac{-}{\models \{\mathsf{T}\}\ \mathsf{gensym} :_u \{\forall \delta.u \bullet () = m\{m\#\delta\}\}}$ $\quad$ *Lines.1-13*

□

Soundness of $[\textsc{Lam}]_\nu$

$$\frac{(\mathbb{\Gamma}{+}\delta{+}x:\alpha) \Vdash \{A \wedge B\}\ M :_m \{C\} \quad A\text{-}\textsc{ExtInd}_{Syn}}{\mathbb{\Gamma} \Vdash \{A\}\ \lambda x^\alpha.M :_u \{\forall\delta.\forall x^\alpha \in (\delta).(B \to u \bullet x = m\{C\})\}}\ [\textsc{Lam}]_\nu$$

Proof. Clearly Lem. 112 ensures A-$\textsc{ExtInd}_{Syn}$ implies A-$\textsc{ExtInd}_{Sem}$.

1	Assume $\mathbb{\Gamma}$ s.t. $\mathbb{\Gamma} \Vdash \{A\}\ \lambda x^\alpha.M :_u \{\forall\delta.\forall x^\alpha \in (\delta).(B \to u \bullet x = m\{C\})\}$
2	and $\mathbb{\Gamma}{+}\delta{+}x \Vdash \{A \wedge B\}\ M :_m \{C\}$
3	Assume $I.H.(1)$: $\forall\, \xi_0^{\mathbb{\Gamma}_1+\delta+x}.\ \mathbb{\Gamma}{+}\delta{+}x \triangleright \xi_0$ $\wedge\ \xi_0 \models A \wedge B \ \to\ \exists\, V_m.\ (\mathring{a}(\xi_0),\ M\xi_0) \Downarrow (G',\ V_m)\ \wedge\ \xi_0 \cdot m : V_m \models C$
4	Assume $\xi^{\mathbb{\Gamma}_1}$ s.t. $\mathbb{\Gamma} \triangleright \xi\ \wedge\ \xi \models A$
5	$\to\ (\mathring{a}(\xi),\ (\lambda x.M)\xi) \Downarrow (\mathring{a}(\xi),\ \lambda x.(M\xi\backslash x))$ — *Op. Sem.$\lambda x.M$*
6	Let: $V_u \equiv \lambda x.M(\xi\backslash x)$ & $\xi_u \equiv \xi \cdot u : V_u$ $\xi_1^{\mathbb{\Gamma}_1} \equiv \xi_{ud} \cdot \delta : \mathbb{\Gamma}_{ud}\backslash_{-TCV}$ & $\xi_2^{\mathbb{\Gamma}_2} \equiv \xi_{ud} \cdot \delta : \mathbb{\Gamma}_{ud}\backslash_{-TCV} \cdot x : V_x$
7	$\to\ \forall\, \xi_{ud}^{\mathbb{\Gamma}_{ud}}.\ \xi_u \preccurlyeq^\star \xi_{ud}$ — *Tautology $(B \to B)$* $\wedge\ \forall\, P_x, V_x.\ P_x\ {}^{[\mathbb{\Gamma}_1,\ \xi_1]}\!\rightsquigarrow V_x\ \wedge\ \xi_2 \models B\ \to\ \xi_2 \models B$
8	$\to\ \forall\, \xi_{ud}^{\mathbb{\Gamma}_{ud}}.\xi_u \preccurlyeq^\star \xi_{ud}$ — *Line.4, $A-$ExtInd$_{Sem}$, Sem.$\wedge$* $\wedge\ \forall\, P_x, V_x.\ P_x\ {}^{[\mathbb{\Gamma}_1,\ \xi_1]}\!\rightsquigarrow V_x\ \wedge\ \xi_2 \models B\ \to\ \xi_2 \models A \wedge B$
9	$\to\ \forall\, \xi_{ud}^{\mathbb{\Gamma}_{ud}}.\xi_u \preccurlyeq^\star \xi_{ud}$ — *I.H.(1)* $\wedge\ \forall\, P_x, V_x.\ P_x\ {}^{[\mathbb{\Gamma}_1,\ \xi_1]}\!\rightsquigarrow V_x$ — $\mathbb{\Gamma}{+}u{+}\delta{+}x \triangleright \xi_2$ $\wedge\ \xi_2 \models B\ \to\ \exists\, V_m.\ M\ {}^{[\mathbb{\Gamma}_2,\ \xi_2]}\!\rightsquigarrow V_m\ \wedge\ \xi_2 \cdot m : V_m \models C$
10	$\to\ \forall\, \xi_{ud}^{\mathbb{\Gamma}_{ud}}.\xi_u \preccurlyeq^\star \xi_{ud}$ — $M\xi \equiv ((\lambda x.M)x)\xi \equiv [\![u]\!]_{\xi_2}[\![x]\!]_{\xi_2}$ $\wedge\ \forall\, P_x, V_x.\ P_x\ {}^{[\mathbb{\Gamma}_1,\ \xi_1]}\!\rightsquigarrow V_x$ — $[\![\mathbb{\Gamma}_1]\!]_{\xi_1} \equiv [\![\mathbb{\Gamma}_{ud}{+}\delta]\!]_{\xi_1}$ $\wedge\ \xi_2 \models B\ \to\ \exists\, V_m.\ \ ux\ {}^{[\mathbb{\Gamma}_2,\ \xi_2]}\!\rightsquigarrow V_m\ \wedge\ \xi_2 \cdot m : V_m \models C$
11	$\to\ \forall\, \xi_{ud}^{\mathbb{\Gamma}_{ud}}.\ \xi_u \preccurlyeq^\star \xi_{ud}$ — *Sem.$B \to u \bullet x = m\{C\}$* $\wedge\ \forall\, P_x, V_x.\ P_x\ {}^{[\mathbb{\Gamma}_1,\ \xi_1]}\!\rightsquigarrow V_x\ \to\ \xi_2 \models B \to u \bullet x = m\{C\}$
12	$\to\ (\mathring{a}(\xi),\ (\lambda x.M)\xi) \Downarrow (\mathring{a}(\xi),\ V_u)$ — *Line.5 & Sem.$\forall\delta.\forall x \in (\delta).$* $\wedge\ \xi \cdot u : V_u \models \forall\delta.\forall x \in (\delta).B \to u \bullet x = m\{C\}$
13	$\to\ \models \{A\}\ \lambda x.M :_u \{\forall\delta.\forall x \in (\mathbb{\Gamma}{+}u{+}\delta).(B \to u \bullet x = m\{C\})\}$ — *Lines.1, 4-12*
14	Hence: $\dfrac{A\text{-}\textsc{ExtInd}_{Syn} \quad \models \{A \wedge B\}\ M :_m \{C\}}{\models \{A\}\ \lambda x.M :_u \{\forall\delta.\forall x \in (\mathbb{\Gamma}{+}u{+}\delta).(B \to u \bullet x = m\{C\})\}}$ — *Lines.1-13*

□

Soundness of $[\textsc{App}]_\nu$

$$\frac{\{A\}\ M :_m \{B\} \quad \{B\}\ N :_n \{m \bullet n = u\{C\}\} \quad C\text{-}\textsc{Thin}_{Syn}(m,n)}{\{A\}\ MN :_u \{C\}}\ [\textsc{App}]_\nu$$

Proof. Clearly Lem. 113 ensures A-$\textsc{Thin}_{Syn}(x)$ implies A-$\textsc{Thin}_{Sem}(x)$.

1	Let: $\xi_m \equiv \xi \cdot m : V_m$ and $\xi_{mn} \equiv \xi_m \cdot n : V_n$
2	Assume $\mathbb{\Gamma}$ s.t. $\mathbb{\Gamma} \Vdash \{A\}\ MN :_u \{C\}$
3	$\rightarrow\ \mathbb{\Gamma} \Vdash \{A\}\ M :_m \{B\}\ \wedge\ \mathbb{\Gamma}{+}m \Vdash \{B\}\ N :_n \{m \bullet n = u\{C\}\}$
4	$\mathbb{\Gamma} \downarrow_{-TC} \vdash MN : \alpha_2\ \rightarrow\ \mathbb{\Gamma} \downarrow_{-TC} \vdash M : \alpha_1 \to \alpha_2\ \wedge\ \mathbb{\Gamma} \downarrow_{-TC} \vdash N : \alpha_1$
5	$I.H.(1):\ \forall\, \xi_1^{\mathbb{\Gamma}_1}.\mathbb{\Gamma} \triangleright \xi_1\ \rightarrow\ \xi_1 \models A \rightarrow M \overset{[\mathbb{\Gamma}_1,\ \xi_1]}{\rightsquigarrow} V_m\ \wedge\ \xi_m \models B$
6	$I.H.(2):\ \forall\, \xi_2^{\mathbb{\Gamma}_2}.\mathbb{\Gamma}{+}m \triangleright \xi_2\ \rightarrow\ \xi_2 \models B \rightarrow\ N \overset{[\mathbb{\Gamma}_2,\ \xi_2]}{\rightsquigarrow} V_n$ $\wedge\ \xi_2 \cdot n : V_n \models m \bullet n = u\{C\}$
7	Assume $\xi^{\mathbb{\Gamma}_0}$ s.t. $\mathbb{\Gamma} \triangleright \xi\ \wedge\ \xi \models A$
8	$\rightarrow\ M \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_m\ \wedge\ \xi \cdot m : V_m \models B$ — *I.H.(1)*
9	$\rightarrow\ M \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_m\ \wedge\ N \overset{[\mathbb{\Gamma}_m,\ \xi_m]}{\rightsquigarrow} V_n\ \wedge\ \xi_{mn} \models m \bullet n = u\{C\}$ — *I.H.(2)*
10	$\rightarrow\ M \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_m\ \wedge\ N \overset{[\mathbb{\Gamma}_m,\ \xi_m]}{\rightsquigarrow} V_n$ — *Sem.* $\bullet = \{\}$ $\wedge\ \exists\, V_u.\ mn \overset{[\mathbb{\Gamma}+m+n,\ \xi_{mn}]}{\rightsquigarrow} V_u\ \wedge\ \xi_{mn} \cdot u : V_u \models C$
11	$\rightarrow\ M \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_m\ \wedge\ N \overset{[\mathbb{\Gamma}_m,\ \xi_m]}{\rightsquigarrow} V_n$ — *Sem.* $[\![x]\!]_\xi$ $\wedge\ \exists\, V_u.\ (\mathring{a}(\xi_{mn}),\ V_m V_n) \Downarrow (G''',\ V_u)\ \wedge\ \xi_{mn} \cdot u : V_u \models C$
12	$\rightarrow\ M \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_m\ \wedge\ N \overset{[\mathbb{\Gamma}_m,\ \xi_m]}{\rightsquigarrow} V_n$ — C-$\textsc{Thin}_{Sem}(m,n)$ $\wedge\ \exists\, V_u.\ (\mathring{a}(\xi_{mn}),\ V_m V_n) \Downarrow (G''',\ V_u)\ \wedge\ \xi \cdot u : V_u \models C$
13	$\rightarrow\ \exists\, V_u.\ MN \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_u\ \wedge\ \xi \cdot u : V_u \models C$ — *Op. Sem.(App)*
14	$\rightarrow\ \forall\, \xi^{\mathbb{\Gamma}_0}.\ \mathbb{\Gamma} \triangleright \xi \rightarrow\ \xi \models A \rightarrow\ \exists\, V_u.\ MN \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_u$ — *Lines.7-13* $\wedge\ \xi \cdot u : V_u \models C$
15	Hence: $\dfrac{\models \{A\}\ M :_m \{B\} \qquad \models \{B\}\ N :_n \{m \bullet n = u\{C\}\}}{\models \{A\}\ MN :_u \{C\}}$ — *Lines.1-13*

□

Soundness of Other Key Rules

The soundness proof of the rules $[\textsc{Pair}]_\nu$, $[\textsc{Proj}_i]_\nu$, $[\textsc{Neg}]_\nu$ and $[\textsc{If}]_\nu$ follow closely those of the STLC. They are not included here but are proven in Ap. A.2.

6.3.2 Soundness of Structural Rules

The structural rules from Fig. 4.9 are similar to those that are seen in the literature [28, 5, 72] with adaptations to the alternative logical constructors. They are proven sound in the following subsections.

Soundness of Structural Rules $[\wedge \to]_\nu$, $[\to \wedge]_\nu$, $[\vee\text{-}\textsc{Pre}]_\nu$, $[\wedge\text{-}\textsc{Post}]_\nu$

$$\frac{\{A \wedge B\}\ M :_m \{C\} \quad B\text{-}\textsc{ExtInd}_{Syn}}{\{A\}\ M :_m \{B \to C\}}\ [\wedge \to]_\nu \qquad \frac{\{A\}\ M :_m \{B \to C\} \quad B\text{-}\textsc{ExtInd}_{Syn}}{\{A \wedge B\}\ M :_m \{C\}}\ [\to \wedge]_\nu$$

$$\frac{\{A\}\ M :_m \{B\} \quad \{A'\}\ M :_m \{B\}}{\{A \vee A'\}\ M :_m \{B\}}\ [\vee\text{-}\textsc{Pre}]_\nu \qquad \frac{\{A\}\ M :_m \{B\} \quad \{A\}\ M :_m \{B'\}}{\{A\}\ M :_m \{B \wedge B'\}}\ [\wedge\text{-}\textsc{Post}]_\nu$$

Clearly Lem. 112 ensures A-$\textsc{ExtInd}_{Syn}$ implies A-$\textsc{ExtInd}_{Sem}$, hence the proofs for these rules follow the equivalent proof of the λ-logic rule with the addition of B-$\textsc{ExtInd}_{Sem}$, ensuring these hold trivially.

Soundness of Structural Rule $[\textsc{Conseq}]_\nu$

The $[\textsc{Conseq}]_\nu$ rule allows for the use of the logic of axioms in Sec. 4.5 in the logic of rules. The proof is trivial and follows the proof of $[\textsc{Conseq}]_\lambda$ as follows.

$$\frac{A \to A' \quad \{A'\}\ M :_m \{B'\} \quad B' \to B}{\{A\}\ M :_m \{B\}}\ [\textsc{Conseq}]_\nu$$

Proof. By application of the assumptions.

1	Assume $\mathbb{\Gamma}$, s.t. $\mathbb{\Gamma} \Vdash \{A\}\ M :_m \{B\}$	
2	Assume typing holds for assumptions i.e. $\mathbb{\Gamma} \Vdash A \to A' \ \wedge\ \mathbb{\Gamma} \Vdash \{A'\}\ M^\alpha :_m \{B'\} \ \wedge\ \mathbb{\Gamma}{+}m : \alpha \Vdash B' \to B$	
3	Assume I.H.(1): $\forall\, \xi_0^{\mathbb{\Gamma}_0}.\ \mathbb{\Gamma} \triangleright \xi_0 \ \to\ \xi_0 \models A \to A'$	
4	Assume I.H.(2): $\forall\, \xi_0^{\mathbb{\Gamma}_0}.\ \mathbb{\Gamma} \triangleright \xi_0 \ \to\ \xi_0 \models \{A'\}\ M :_m \{B'\}$	
5	Assume I.H.(3): $\forall\, \xi_1^{\mathbb{\Gamma}_1}.\ \mathbb{\Gamma}{+}m \triangleright \xi_1 \ \to\ \xi_1 \models B' \to B$	
6	Assume $\xi^{\mathbb{\Gamma}'}$ s.t. $\mathbb{\Gamma} \triangleright \xi \ \wedge\ \xi \models A$	
7	$\to\ \xi \models A'$	*M.P., I.H.(1)*
8	$\to\ M \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V \ \wedge\ \xi \cdot m : V \models B'$	*M.P., I.H.(2)*
9	$\to\ M \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V \ \wedge\ \xi \cdot m : V \models B$	*M.P., I.H.(3)*
10	$\to\ \xi \models A \ \to\ M \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V \ \wedge\ \xi \cdot m : V \models B$	*Lines.6-9*
11	$\to\ \forall\, \xi^{\mathbb{\Gamma}'}.\mathbb{\Gamma} \triangleright \xi \ \to\ \xi \models \{A\}\ M :_m \{B\}$	*Lines.3-10*
12	Hence: $\dfrac{A \to A' \quad \models \{A'\}\ M :_m \{B'\} \quad B' \to B}{\models \{A\}\ M :_m \{B\}}$	*Lines.1-11*

□

Soundness of Structural Rule $[\textsc{Invar}]_\nu$

The $[\textsc{Invar}]_\nu$ rule is similar to the STLC program logic $[\textsc{Invar}]_\lambda$ rule with the additional requirement that C-$\textsc{ExtInd}_{Syn}$ to ensure C is satisfied when the resulting value derived from M extend the model in the proof.

$$\frac{C\text{-}\textsc{ExtInd}_{Syn} \quad \{A\}\ M :_m \{B\}}{\{A \wedge C\}\ M :_m \{B \wedge C\}}\ [\textsc{Invar}]_\nu$$

Proof. Clearly Lem. 112 ensures A-$\textsc{ExtInd}_{Syn}$ implies A-$\textsc{ExtInd}_{Sem}$.

1	Assume $\mathbb{\Gamma}$ s.t. $\mathbb{\Gamma} \Vdash \{A \wedge C\}\ M :_m \{B \wedge C\}$	
2	$\rightarrow$ $\mathbb{\Gamma} \Vdash \{A\}\ M :_m \{B\}$	
3	Assume $I.H.(1)$: $\forall\, \xi^{\mathbb{\Gamma}_0}.\mathbb{\Gamma} \triangleright \xi$ $\rightarrow$ $\xi \models A \rightarrow$ $M \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V$ $\wedge$ $\xi \cdot m : V \models B$	
4	Assume for some model $\xi^{\mathbb{\Gamma}_0}$ s.t. $\mathbb{\Gamma} \triangleright \xi$ $\wedge$ $\xi \models A \wedge C$	
5	$\xi \models A$ $\wedge$ $\xi \models C$	*Sem.*$\wedge$
6	$M \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V$ $\wedge$ $\xi \cdot m : V \models B$ $\wedge$ $\xi \models C$	*I.H.(1)*
7	$M \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V$ $\wedge$ $\xi \cdot m : V \models B$ $\wedge$ $\xi \cdot m : V \models C$	C-$\textsc{ExtInd}_{Sem}$ $\xi \preccurlyeq^\star \xi \cdot m : V$
8	$M \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V$ $\wedge$ $\xi \cdot m : V \models B \wedge C$	*Sem.*$\wedge$
9	$\forall\, \xi^{\mathbb{\Gamma}_0}.\mathbb{\Gamma} \triangleright \xi$ $\rightarrow$ $\xi \models \{A \wedge C\}\ M :_m \{B \wedge C\}$	*Lines.5-8*
10	Hence: $\dfrac{C\text{-}\textsc{ExtInd}_{Syn} \quad \models \{A\}\ M :_m \{B\}}{\models \{A \wedge C\}\ M :_m \{B \wedge C\}}$	*Lines.1-9*

□

Soundness of structural rule $[\textsc{Weak}(x)]_\nu$

$$\frac{\mathbb{\Gamma} \Vdash \{A\}\ M :_m \{B\} \quad A, B\text{-}\textsc{ExtInd}_{Syn}}{\mathbb{\Gamma}{+}x \Vdash \{A^{-x}\}\ M^{-x} :_m \{B^{-x}\}}\ [\textsc{Weak}(x)]_\nu$$

Proof. Clearly Lem. 112 ensures A-$\textsc{ExtInd}_{Syn}$ implies A-$\textsc{ExtInd}_{Sem}$.

1	Assume $\mathbb{\Gamma}$ s.t. $\mathbb{\Gamma}{+}x \Vdash \{A^{-x}\}\ M :_m \{B^{-x}\}$	
2	$\rightarrow$ $\mathbb{\Gamma} \downarrow_{-TC} \vdash M : \alpha$ $\leftrightarrow$ $(\mathbb{\Gamma}{+}x) \downarrow_{-TC} \vdash M^{-x} : \alpha$	*Def. 43*, M^{-x}
3	Assume $I.H.(1)$: $\forall\, \xi^{\mathbb{\Gamma}+d}.\ \mathbb{\Gamma} \triangleright \xi \rightarrow \xi \models A \rightarrow M \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V \wedge \xi \cdot u : V \models B$	
4	Assume $\mathbb{\Gamma}{+}x \triangleright \xi_x^{\mathbb{\Gamma}+x} \equiv (\xi \cdot x : V_x)$ i.e. $\mathbb{\Gamma} \triangleright \xi$ and $\xi \preccurlyeq^\star \xi \cdot x : V_x$	
5	Assume $\xi_x \models A$	
6	$\leftrightarrow$ $\xi \models A$	$\xi \preccurlyeq^\star \xi_x$, A -$\textsc{ExtInd}_{Sem}$
7	$\rightarrow$ $M \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V \wedge \xi \cdot u : V \models B$	*I.H.(1)*
8	$\leftrightarrow$ $M \overset{[\mathbb{\Gamma},\ \xi_x]}{\rightsquigarrow} V \wedge \xi \cdot u : V \models B$	*Lem. 102* $(\mathring{a}(V) \cap \mathring{a}(V_x) \subseteq \mathring{a}(\xi))$
9	$\rightarrow$ $M \overset{[\mathbb{\Gamma}+x,\ \xi_x]}{\rightsquigarrow} V \wedge \xi \cdot u : V \models B$	*Lem. 93*
10	(Lem. 104 $\wedge$ $(\mathring{a}(V) \cap \mathring{a}(V_x) \subseteq \mathring{a}(\xi))$ $\rightarrow$ $\xi \cdot u : V \preccurlyeq^\star \xi_x \cdot u : V$) $\rightarrow$ $M \overset{[\mathbb{\Gamma}+x,\ \xi_x]}{\rightsquigarrow} V \wedge \xi_x \cdot u : V \models B$	B-$\textsc{ExtInd}_{Sem}$
11	$\xi \cdot x : V_x \models A \rightarrow M \overset{[\mathbb{\Gamma}+x,\ \xi_x]}{\rightsquigarrow} V \wedge \xi_x \cdot u : V \models B$	*Lines.7-12*
12	$\forall\, \xi_x.\ \mathbb{\Gamma}{+}x \triangleright \xi_x\ \ \xi_x \models A \rightarrow M \overset{[\mathbb{\Gamma}+x,\ \xi_x]}{\rightsquigarrow} V \wedge \xi_x \cdot u : V \models B$	*Lines.6-13*
13	$\rightarrow$ $\models \{A^{-x}\}\ M :_m \{B^{-x}\}$	
14	$\rightarrow$ $\dfrac{A, B\text{-}\textsc{ExtInd}_{Syn} \quad \mathbb{\Gamma} \Vdash \{A\}\ M :_m \{B\}}{\mathbb{\Gamma}{+}x \Vdash \{A^{-x}\}\ M :_m \{B^{-x}\}}$	*Lines.3, 7, 10, 13*

□

Soundness of structural rule $[\textsc{Weak}(\delta)]_\nu$

$$\frac{\mathbb{\Gamma}+\mathbb{\Gamma}' \Vdash \{A\}\ M :_m \{B\} \quad A, B\text{-}\textsc{ExtInd}_{Syn}}{\mathbb{\Gamma}+\delta+\mathbb{\Gamma}' \Vdash \{A^{-\delta}\}\ M :_m \{B^{-\delta}\}}\ [\textsc{Weak}(\delta)]_\nu$$

Proof. Clearly Lem. 112 ensures A-$\textsc{ExtInd}_{Syn}$ implies A-$\textsc{ExtInd}_{Sem}$.

1	Assume $\mathbb{\Gamma}+\mathbb{\Gamma}'$ s.t. $\mathbb{\Gamma}+\delta+\mathbb{\Gamma}' \Vdash \{A^{-\delta}\}\ M :_m \{B^{-\delta}\}$	
2	$\rightarrow$ $(\mathbb{\Gamma}+\mathbb{\Gamma}')\backslash_{-TCV} \vdash M : \alpha$ $\leftrightarrow$ $(\mathbb{\Gamma}+\delta+\mathbb{\Gamma}')\backslash_{-TCV} \vdash M : \alpha$	*Fig. 2.16*
3	Assume $I.H.(1)$: $\forall\, \xi.\ \mathbb{\Gamma}+\mathbb{\Gamma}' \triangleright \xi \rightarrow \xi \models A \rightarrow M \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V \wedge \xi \cdot u : V \models B$	
4	Assume $\xi_2 \equiv (\xi_0 \cdot \xi_d \cdot \delta : \mathbb{\Gamma}_x \cdot \xi_1)$ s.t. $\mathbb{\Gamma}+\delta+\mathbb{\Gamma}' \triangleright \xi_2$ i.e. $\mathbb{\Gamma} \triangleright \xi_0$, $\mathbb{\Gamma}+\delta \triangleright (\xi_0 \cdot \xi_d)^{\mathbb{\Gamma}_x} \cdot \delta : \mathbb{\Gamma}_x\backslash_{-TCV}$	
5	Assume $\mathbb{\Gamma}+\mathbb{\Gamma}' \triangleright \xi_0 \cdot \xi_1$ hence $\xi_0 \preccurlyeq^\star \xi_0 \cdot \xi_1$	
6	Assume $\xi_0 \cdot \xi_d \cdot \delta : \mathbb{\Gamma}_x \cdot \xi_1 \models A$	*Assumption*
7	$\leftrightarrow$ $\xi_0 \cdot \xi_d \cdot \xi_1 \models A$ $\leftrightarrow$ $\xi_0 \cdot \xi_1 \cdot \xi_d \models A$	*Lem. 105, Lem. 108*
8	$\leftrightarrow$ $\xi_0 \cdot \xi_1 \models A$	*(Lem. 104* $\rightarrow \xi_0 \cdot \xi_1 \preccurlyeq^\star \xi_0 \cdot \xi_1 \cdot \xi_d)$ A-$\textsc{ExtInd}_{Sem}$
9	$\rightarrow$ $M \overset{[\mathbb{\Gamma}+\mathbb{\Gamma}',\ \xi_0\cdot\xi_1]}{\rightsquigarrow} V \wedge \xi_0 \cdot \xi_1 \cdot u : V \models B$	*I.H.(1)*
10	$\rightarrow$ $M \overset{[\mathbb{\Gamma}+\mathbb{\Gamma}',\ \xi_0\cdot\xi_1\cdot\xi_d]}{\rightsquigarrow} V \wedge \xi_0 \cdot \xi_1 \cdot u : V \cdot \xi_d \models B$	*Lem. 102* $\wedge$ *Lem. 104* B-$\textsc{ExtInd}_{Sem}$
11	$\rightarrow$ $M \overset{[\mathbb{\Gamma}+\mathbb{\Gamma}',\ \xi_0\cdot\xi_d\cdot\xi_1]}{\rightsquigarrow} V \wedge \xi_0 \cdot \xi_d \cdot \xi_1 \cdot u : V \models B$	*Lem. 107* $\wedge$ *Lem. 108*
12	$\rightarrow$ $M \overset{[\mathbb{\Gamma}+\mathbb{\Gamma}',\ \xi_2]}{\rightsquigarrow} V \wedge \xi_2 \cdot u : V \models B$	*Lem. 102* $\wedge$ *Lem. 105*
13	$\rightarrow$ $M \overset{[\mathbb{\Gamma}+\delta+\mathbb{\Gamma}',\ \xi_2]}{\rightsquigarrow} V \wedge \xi_2 \cdot u : V \models B$	*Lem. 93*
14	$\xi_2 \models A \rightarrow M \overset{[\mathbb{\Gamma}+\delta+\mathbb{\Gamma}',\ \xi_2]}{\rightsquigarrow} V \wedge \xi_2 \cdot u : V \models B$	*Lines.6-13*
15	$\rightarrow$ $\models (\mathbb{\Gamma}+\delta+\mathbb{\Gamma}' \Vdash)\{A^{-\delta}\}\ M :_m \{B^{-\delta}\}$	*Lines.4-14*
16	$\rightarrow$ $\dfrac{\models (\mathbb{\Gamma}+\mathbb{\Gamma}' \Vdash)\{A\}\ M :_m \{B\}}{\models (\mathbb{\Gamma}+\delta+\mathbb{\Gamma}' \Vdash)\{A^{-\delta}\}\ M :_m \{B^{-\delta}\}}$	*Lines.3, 8, 10, 15*

□

6.3.3 Soundness of Derived Rules

The proof that the derived rules $[\textsc{Let}]_\nu$ and $[\textsc{LetFresh}]_\nu$ are introduced in the following sections.

Soundness of $[\textsc{Let}]_\nu$

$$\frac{\{A\}\ M :_x \{B\} \quad \{B\}\ N :_u \{C\} \quad C\ \textsc{Thin}_{Syn}(x) \quad A,B,C - \textsc{ExtInd}_{Syn}}{\{A\}\ \mathsf{let}\ x = M\ \mathsf{in}\ N :_u \{C\}}\ [\textsc{Let}]_\nu$$

Proof. The abbreviation $\mathsf{let}\ x = M\ \mathsf{in}\ N \stackrel{def}{=} (\lambda x.N)(M)$ is used as follows.

1	Let: $D \stackrel{def}{=} \forall\delta.\forall y \in (\delta).(B \to p \bullet x = u\{C\})[y/x]$ which is $\textsc{ExtInd}_{Syn}$	*Def. 54*
2	Assume $\mathbb{\Gamma}$ s.t. $\mathbb{\Gamma} \Vdash \{A\}\ \mathsf{let}\ x = M\ \mathsf{in}\ N :_u \{C\}$	
3	Assume $I.H.(2):\ \mathbb{\Gamma}{+}x \Vdash \{B\}\ N :_u \{C\}$	
4	$\mathbb{\Gamma}{+}\delta{+}x \Vdash \{B^{-\delta}\}\ N :_u \{C^{-\delta}\}$	B,C-$\textsc{ExtInd}_{Syn}$ $[\textsc{Weak}(\delta)]_\nu$
5	$\mathbb{\Gamma} \Vdash \{\mathsf{T}\}\ \lambda x.N :_p \{\forall\delta.\forall x \in (\delta).B \to p \bullet x = u\{C\}\}$	$[\textsc{Lam}]_\nu$
6	$\mathbb{\Gamma} \Vdash \{A\}\ \lambda x.N :_p \{A \wedge D\}$	$[\textsc{Invar}]_\nu$, A -$\textsc{ExtInd}_{Syn}$
7	Assume $I.H.(1):\ \mathbb{\Gamma} \Vdash \{A\}\ M :_x \{B^{-\delta}\}$	
8	$\mathbb{\Gamma}{+}p \Vdash \{A\}\ M :_x \{B\}$	A,B -$\textsc{ExtInd}_{Syn}$, $[\textsc{Weak}(x)]_\nu$
9	$\mathbb{\Gamma}{+}p \Vdash \{A \wedge D\}\ M :_x \{B \wedge D\}$	D-$\textsc{ExtInd}_{Syn}$ $[\textsc{Invar}]_\nu$
10	$\mathbb{\Gamma}{+}p{+}x \Vdash B \wedge D$ $\to\ B \wedge \forall y \in (\mathbb{\Gamma}{+}p{+}x).(B \to p \bullet x = u\{C\})[y/x]$ $\to\ B \wedge (B \to p \bullet x = u\{C\})$ $\to\ p \bullet x = u\{C\}$	$(utc1)$, $B^{-\delta}$ and $C^{-\delta}$ $(A[y/x][x/y] \equiv A)$ $(u1)$ M.P.
11	$\mathbb{\Gamma}{+}p \Vdash \{A \wedge D\}\ M :_x \{p \bullet x = u\{C\}\}$	$[\textsc{Conseq}]_\nu$, *Lines.9, 10*
12	$C\ \textsc{Thin}_{Syn}(x) \to C\ \textsc{Thin}_{Syn}(p)$	*Lem. 56*
13	$\mathbb{\Gamma} \Vdash \{A\}\ (\lambda x.N)(M) :_u \{C\}$	$[\textsc{App}]_\nu$, *Lines.6, 11, 12*
14	$\dfrac{C\ \textsc{Thin}_{Syn}(x) \quad A,B,C - \textsc{ExtInd}_{Syn} \quad \{A\}\ M :_x \{B\} \quad \{B\}\ N :_u \{C\}}{\{A\}\ (\lambda x.N)M :_u \{C\}}$	*Lines.3, 4, 6, 11, 12* $\to$ *Line.13*
15	$\dfrac{C\ \textsc{Thin}_{Syn}(x) \quad A,B,C - \textsc{ExtInd}_{Syn} \quad \{A\}\ M :_x \{B\} \quad \{B\}\ N :_u \{C\}}{\{A\}\ \mathsf{let}\ x = M\ \mathsf{in}\ N :_u \{C\}}$	$\mathsf{let}\ x = M\ \mathsf{in}\ N \stackrel{\text{def}}{=} (\lambda x.N)M$

□

Soundness of the Derived Rule $[\textsc{LetFresh}]_\nu$

The $[\textsc{LetFresh}]_\nu$ rule can be derived from the $[\textsc{Let}]_\nu$ rule and the derivation for $\mathsf{gensym}()$ which can be seen in Ex. 24 in Chapt. 7. This rule is introduced as the $\mathsf{let}\ x = \mathsf{gensym}()\ \mathsf{in}\ \cdot$ construction is commonly used in the reasoning.

$$\frac{\mathbb{\Gamma}{+}x \Vdash \{A \wedge x\#\mathbb{\Gamma}\}\ M :_m \{C\} \quad C\ \textsc{Thin}_{Syn}(x) \quad A, C\text{-}\textsc{ExtInd}_{Syn}}{\mathbb{\Gamma} \Vdash \{A\}\ \mathsf{let}\ x = \mathsf{gensym}\ ()\ \mathsf{in}\ M :_m \{C\}}\ [\textsc{LetFresh}]_\nu$$

Proof.

1	Assume $\mathbb{\Gamma}$ s.t. $\mathbb{\Gamma} \Vdash \{A\}\ \mathsf{let}\ x = \mathsf{gensym}()\ \mathsf{in}\ N :_u \{C\}$	
2	$\mathbb{\Gamma} \Vdash \{\mathsf{T}\}\ \mathsf{gensym}() :_x \{x\#\mathbb{\Gamma}\}$	*See Ex. 24*
3	$\mathbb{\Gamma} \Vdash \{A\}\ \mathsf{gensym}() :_x \{A \wedge x\#\mathbb{\Gamma}\}$	$[\textsc{Invar}]_\nu$, *7*, A-$\textsc{ExtInd}_{Syn}$
4	$\mathbb{\Gamma}{+}x \Vdash \{A \wedge x\#\mathbb{\Gamma}\}\ M :_m \{C\}$	*Assumption*
5	C-$\textsc{Thin}_{Syn}(x)$	*Assumption*
6	A, C-$\textsc{ExtInd}_{Syn} \rightarrow (A \wedge x\#\mathbb{\Gamma})$-$\textsc{ExtInd}_{Syn}$	*Assumption, Def. 54*
7	$\mathbb{\Gamma} \Vdash \{A\}\ \mathsf{let}\ x = \mathsf{gensym}()\ \mathsf{in}\ M :_m \{C\}$	$[\textsc{Let}]_\nu$, *Lines.3, 4, 5, 6*
8	Hence: $\dfrac{\{A \wedge x\#\mathbb{\Gamma}\}\ N :_u \{C\} \quad C\ \textsc{Thin}_{Syn}(x) \quad A, C\text{-}\textsc{ExtInd}_{Syn}}{\{A\}\ \mathsf{let}\ x = M\ \mathsf{in}\ N :_u \{C\}}$	*Lines.1-7*

□

6.4 Soundness Theorem

The individual soundness proofs of each axiom and rule in Sec. 6.2 and Sec. 6.3 can now be used to prove the soundness of the logic in Thm. 114, below.

Theorem 114 (The ν-logic is Sound)**.** *The soundness of the logic is defined as follows.*

$$\forall\, \mathbb{\Gamma}, \{A\}\ M :_u \{B\}.\ \mathbb{\Gamma} \Vdash \{A\}\ M :_u \{B\} \rightarrow \vdash \{A\}\ M :_u \{B\} \rightarrow \vDash \{A\}\ M :_u \{B\}$$

Proof. Trivial given all axioms and rules are proven sound in Sec. 6.2 and Sec. 6.3 respectively. □

6.5 Conservativity

The ν_{GS}-calculus is a direct extension of the STLC with a simple addition of names (in types and syntax constructor/destructor). Thus the ν-logic can reason about STLC terms. If the ν-logic cannot prove anything about an STLC term which the λ-logic cannot prove then the ν-logic is defined as a *conservative extension* of the λ-logic (or *conservativity* for brevity).

First a proof that every triple derivable in the λ-logic can indeed be derived in the ν-logic is introduced in Sec. 6.5.1. A sketch that the ν-logic is a conservative extension of the λ-logic is provided in Sec. 6.5.2. For convenience the λ-logic axioms from Sec. 2.2.2, are written $(\cdot)_\lambda$, and the ν-logic axioms from Sec. 4.5 are written $(\cdot)_\nu$. Similarly Hoare triples from the λ-logic and ν-logic are written $\{A\}\ M :_u \{B\}_\lambda$ and $\{A\}\ M :_u \{B\}_\nu$ respectively with their respective derivation symbols $\vdash_\lambda$ and $\vdash_\nu$.

6.5.1 The ν-Logic Extends the λ-Logic

A translation from triples in the λ-logic to triples in the ν-logic written $\langle\!\langle \cdot \rangle\!\rangle_{\lambda\to\nu}$, primarily translates $\forall x^\alpha.$ to $\forall x^\alpha \in (\emptyset).$. as this is the only discrepancy between the two syntaxes and it is guaranteed that $\alpha \in \alpha_{\text{-Nm}}$.

Definition 115 (Translation from λ-logic to ν-logic)**.** *Define the translation of λ-logic formulae and triples into ν-logic formulae and triples as follows.*

$$
\begin{aligned}
\langle\!\langle e = e' \rangle\!\rangle_{\lambda\to\nu} &\stackrel{\text{def}}{=} e = e' \\
\langle\!\langle \neg A \rangle\!\rangle_{\lambda\to\nu} &\stackrel{\text{def}}{=} \neg\langle\!\langle A \rangle\!\rangle_{\lambda\to\nu} \\
\langle\!\langle A \wedge B \rangle\!\rangle_{\lambda\to\nu} &\stackrel{\text{def}}{=} \langle\!\langle A \rangle\!\rangle_{\lambda\to\nu} \wedge \langle\!\langle B \rangle\!\rangle_{\lambda\to\nu} \\
\langle\!\langle e \bullet e' = m\{A\} \rangle\!\rangle_{\lambda\to\nu} &\stackrel{\text{def}}{=} e \bullet e' = m\{\langle\!\langle A \rangle\!\rangle_{\lambda\to\nu}\} \\
\langle\!\langle \forall x^\alpha . A \rangle\!\rangle_{\lambda\to\nu} &\stackrel{\text{def}}{=} \forall x^\alpha \in (\emptyset). \langle\!\langle A \rangle\!\rangle_{\lambda\to\nu}
\end{aligned}
$$

$$
\langle\!\langle \{A\}\ M :_m \{B\}_\lambda \rangle\!\rangle_{\lambda\to\nu} \stackrel{\text{def}}{=} \{\langle\!\langle A \rangle\!\rangle_{\lambda\to\nu}\}\ M :_m \{\langle\!\langle B \rangle\!\rangle_{\lambda\to\nu}\}_\nu
$$

This leads to the first lemma that if $\{A\}\ M :_u \{B\}_\lambda$ is derivable in the λ-logic then its translation is derivable in the ν-logic.

Lemma 116.

$$
\forall \{A\}\ M :_u \{B\}_\lambda.\ \vdash_\lambda \{A\}\ M :_u \{B\}_\lambda \ \to\ \vdash_\nu \langle\!\langle \{A\}\ M :_u \{B\}_\lambda \rangle\!\rangle_{\lambda\to\nu}
$$

Meaning, if the rules and axioms of the λ-logic allows for the derivation of $\{A\}\ M :_u \{B\}_\lambda$ then the translation can of $\{A\}\ M :_u \{B\}_\lambda$ into the ν-logic can be derived in the ν-logic.

Proof. This holds through noting that each axiom in the λ-logic translates to an axiom in the ν-logic, and each rule in the λ-logic also translates to a rule in the ν-logic. The key axiom used for this proof is $(u5)$ which allows for the removal and addition of LTCs to any restricted quantification of $\alpha_{\text{-Nm}}$ (i.e. all STLC types).

The key axioms are discussed here.

- Axioms of F.O.L. from Fig. 2.18 are identical given the $\alpha_{\text{-Nm}}$ types.
- Equality axioms are identical in both Fig. 2.17 and Fig. 4.3 where substitution of $\alpha_{\text{-Nm}}$ types is identical in both.
- Translated axioms of the F.O.L. from STLC Fig. 2.18 exist in the exact forms in the axioms for the ν-logic Fig. 4.4 given the restriction of name-free types and axiom $(u5)_\nu$ which allows the LTC to be extended in any universal restricted quantification over $\alpha_{\text{-Nm}}$ types.
- The axioms for evaluation formulae are identical in both λ-logic Fig. 2.19 and the ν-logic Fig. 4.7 on $\alpha_{\text{-Nm}}$ types excluding $(e5)$ in the ν-logic which does not affect $\alpha_{\text{-Nm}}$ quantifiers.

The λ-logic rules introduced in Fig. 2.20 and Fig. 2.21 are all discussed here.

- $[\textsc{Const}]_\lambda$, $[\textsc{Var}]_\lambda$ These clearly are identical to the ν-logic equivalent given substitution is equivalent in both logics for constants and variables.
- $[\textsc{Lam}]_\lambda$ This clearly translates to the $[\textsc{Lam}]_\nu$ with the same post-condition of the conclusion given $(u3)$ and $(utc3)$ which can introduce δ and to the restricted quantifier $\forall x \in (\delta).$ via $(u5)_\nu$.
- $[\textsc{App}]_\lambda$ This rule is identical given all constituents are identical aside from the $\textsc{ExtInd}_{Syn}$ condition which can be guaranteed given all axioms and rules introduce δ-free formulae which implies all formulae are $\textsc{ExtInd}_{Syn}$.
- $[\textsc{If}]_\lambda$ This is identical to the the ν-logic version hence holds.
- $[\textsc{Pair}]_\lambda$ This is identical to the the ν-logic version except the substitution which is identical for pairs.
- $[\textsc{Proj}_i]_\lambda$ This is identical to the the ν-logic version except the substitution. Given all LTCs used in universal restricted quantification introduced are of types which are $\alpha_{\text{-Nm}}$ types then the substitution is also identical.

– The structural rules in Fig. 2.21 are all identical aside from the $\textsc{ExtInd}_{Syn}$ which can be guaranteed given all axioms and rules introduce δ-free formulae which implies all formulae are $\textsc{ExtInd}_{Syn}$.

Hence all rules and axioms in the λ-logic have a direct translation to the ν-logic, hence the translation of derived λ-logic triples can be derived in the ν-logic by using the corresponding translated rule or axiom. □

6.5.2 The ν-Logic is a Conservative Extension of the λ-Logic

Conservativity requires the opposite direction of Lem. 116, but for this the opposite direction of translation is required. Clearly $\forall x^{\mathsf{Nm}} \in (\mathbb{\Gamma}).A$ has no direct translation into the λ-logic hence a check on which expressions, formulae and triples are translatable from the ν-logic to λ-logic is introduced as $\cdot$-Nm-free which requires $\alpha_{\text{-Nm}} \stackrel{def}{=} \{\mathsf{Unit}, \mathsf{Bool}, \alpha_{\text{-Nm}} \times \alpha_{\text{-Nm}}, \alpha_{\text{-Nm}} \to \alpha_{\text{-Nm}}\}$ as follows.

Definition 117 (Nameless expressions, formulae and triples)**.**

$$
\begin{array}{rcl}
c\text{-Nm-}\mathit{free} & \stackrel{\text{def}}{=} & \mathsf{true} \\
x\text{-Nm-}\mathit{free} & \stackrel{\text{def}}{=} & \mathbb{\Gamma} \Vdash x : \alpha_{\text{-Nm}} \\
\pi_i(e')\text{-Nm-}\mathit{free} & \stackrel{\text{def}}{=} & e'\text{-Nm-}\mathit{free} \\
\langle e', e'' \rangle\text{-Nm-}\mathit{free} & \stackrel{\text{def}}{=} & e'\text{-Nm-}\mathit{free} \wedge e''\text{-Nm-}\mathit{free} \\
 & & \\
e = e'\text{-Nm-}\mathit{free} & \stackrel{\text{def}}{=} & e\text{-Nm-}\mathit{free} \wedge e'\text{-Nm-}\mathit{free} \\
\neg A_1\text{-Nm-}\mathit{free} & \stackrel{\text{def}}{=} & A_1\text{-Nm-}\mathit{free} \\
A_1 \wedge A_2\text{-Nm-}\mathit{free} & \stackrel{\text{def}}{=} & A_1\text{-Nm-}\mathit{free} \wedge A_2\text{-Nm-}\mathit{free} \\
e \bullet e' = m\{A_1\}\text{-Nm-}\mathit{free} & \stackrel{\text{def}}{=} & e\text{-Nm-}\mathit{free} \wedge e'\text{-Nm-}\mathit{free} \wedge A\text{-Nm-}\mathit{free} \\
\forall x^{\alpha_1} \in (\mathbb{\Gamma}).A_1\text{-Nm-}\mathit{free} & \stackrel{\text{def}}{=} & A_1\text{-Nm-}\mathit{free} \wedge \alpha_1 \in \alpha_{\text{-Nm}} \\
e^{\mathsf{Nm}} \# \mathbb{\Gamma}\text{-Nm-}\mathit{free} & \stackrel{\text{def}}{=} & \mathsf{false} \\
\forall \delta.A_1\text{-Nm-}\mathit{free} & \stackrel{\text{def}}{=} & A_1\text{-Nm-}\mathit{free} \\
 & & \\
\{A\}\ M :_u \{B\}_\nu\text{-Nm-}\mathit{free} & \stackrel{\text{def}}{=} & A\text{-Nm-}\mathit{free} \wedge M \in \mathit{STLC\text{-}syntax} \wedge B\text{-Nm-}\mathit{free}
\end{array}
$$

The translation from ν-logic formulae and triples to λ-logic formulae and triples, written $\langle\!\langle \cdot \rangle\!\rangle_{\nu \to \lambda}$ is defined as follows assuming $\cdot$-Nm-free holds. A translation of expressions is not required as the $\cdot$-Nm-free check ensures all expressions are name free and hence do not need translating.

Definition 118 (Translation from ν-logic to λ-logic)**.** *Given* $\mathbb{\Gamma} \Vdash A$ *and* A-Nm-*free then* $\langle\!\langle A \rangle\!\rangle_{\nu\to\lambda}$ *is defined inductively on* A *as below. Given* $\mathbb{\Gamma} \Vdash \{A\}\ M :_u \{B\}_\nu$ *and* $\{A\}\ M :_u \{B\}_\nu$-Nm-*free then* $\langle\!\langle \{A\}\ M :_u \{B\}_\nu \rangle\!\rangle_{\nu\to\lambda}$ *is defined inductively on* $\{A\}\ M :_u \{B\}_\nu$ *as follows.*

$$\begin{aligned}
\langle\!\langle e = e' \rangle\!\rangle_{\nu\to\lambda} &\stackrel{\text{def}}{=} e = e' \\
\langle\!\langle \neg A_1 \rangle\!\rangle_{\nu\to\lambda} &\stackrel{\text{def}}{=} \neg\langle\!\langle A_1 \rangle\!\rangle_{\nu\to\lambda} \\
\langle\!\langle A_1 \wedge A_2 \rangle\!\rangle_{\nu\to\lambda} &\stackrel{\text{def}}{=} \langle\!\langle A_1 \rangle\!\rangle_{\nu\to\lambda} \wedge \langle\!\langle A_2 \rangle\!\rangle_{\nu\to\lambda} \\
\langle\!\langle e \bullet e' = m\{A_1\} \rangle\!\rangle_{\nu\to\lambda} &\stackrel{\text{def}}{=} e \bullet e' = m\{\langle\!\langle A_1 \rangle\!\rangle_{\nu\to\lambda}\} \\
\langle\!\langle \forall x^{\alpha_1} \in (\mathbb{\Gamma}).A_1 \rangle\!\rangle_{\nu\to\lambda} &\stackrel{\text{def}}{=} \forall x^{\alpha_1}.\langle\!\langle A_1 \rangle\!\rangle_{\nu\to\lambda} \\
\langle\!\langle \forall \delta.A_1 \rangle\!\rangle_{\nu\to\lambda} &\stackrel{\text{def}}{=} \langle\!\langle A_1 \rangle\!\rangle_{\nu\to\lambda}
\end{aligned}$$

$$\langle\!\langle \{A\}\ M :_u \{B\}_\nu \rangle\!\rangle_{\nu\to\lambda} \stackrel{\text{def}}{=} \{\langle\!\langle A \rangle\!\rangle_{\nu\to\lambda}\}\ M :_u \{\langle\!\langle B \rangle\!\rangle_{\nu\to\lambda}\}_\lambda$$

As described in the chapter introduction, the ν_{GS}-calculus is an extension of the STLC hence the ν-logic should conserve λ-logic ensuring that the logics agree on validity on STLC terms. This is formalised as follows.

Theorem 119 (The ν-logic is a conservative extension of the λ-logic)**.** *This is formally defined as follows.*

$$\begin{aligned}
&\forall\ \{A\}\ M :_u \{B\}_\nu.\ (\{A\}\ M :_u \{B\}_\nu\text{-Nm-}\textit{free} \\
&\qquad \to\ (\vdash_\nu \{A\}\ M :_u \{B\}_\nu \ \to\ \vdash_\lambda \langle\!\langle \{A\}\ M :_u \{B\}_\lambda \rangle\!\rangle_{\nu\to\lambda})
\end{aligned}$$

Proof. A general sketch of the why this holds is introduced here.

The logic of axioms. All ν-logic axioms either have a direct translation to the λ-logic or cannot derive any new formulae about Nm-free variables that are not derivable in the λ-logic.

The logic of rules. The $\{A\}\ M :_u \{B\}_\nu$-Nm-free condition ensure that M is gensym-free (and clearly $\mathfrak{a}(M) = \emptyset$ ensures M contains no names either) hence $[\textsc{Gensym}]_\nu$ can never be used hence this rule is ignored. The other rules translate directly to the λ-logic rules with $[\textsc{Lam}]_\nu$ using axiom $(utc1)_\nu$ and $(u5)_\nu$ to be a direct translation. The $[\textsc{Conseq}]_\nu$ rule uses the statement above that no new formulae about Nm-free variables can be derived in the ν-logic.

□

A more detailed proof is left for future work.

6.6 Summary

The soundness of the ν-logic is proven in this chapter. This consists of a proof that syntactic extension independence and thinness imply their respective semantic definition. Each axiom and rule is then proven sound using the numerous lemmas in Sec. 5.4 and the semantic properties of formulae, thus proving the ν-logic sound.

A proof is provided that the ν-logic extends the λ-logic. A sketch is given of the proof that the extension mentioned is a conservative extension.

Chapter 7

Reasoning Examples

The purpose of introducing a program logic is to allow programs to be reasoned about, hence in this chapter a variety of programs in the ν-calculus are reasoned about to show the application of the program logic. Simple example programs from the STLC are introduced in examples Ex. 21-23. Example programs from Sec. 3.1.2 are introduced in Ex. 24-34. Finally Ex. 36 reasons about the "hard" example and the hard example in the critical context discussed in Ex. 19 [2, 52, 68]. A table of the examples reasoned about and where (if) they are reduced in Sec. 3.1.2 alongside the program itself is included for convenience.

Reasoning	Reduction	Program
Ex. 21		$(\lambda x^{\mathsf{Bool}}.x)\mathsf{true}$
Ex. 22		$\lambda x^{\mathsf{Bool}}.\mathsf{if}\ x\ \mathsf{then\ false\ else\ true}$
Ex. 23		$(\lambda f^{\mathsf{Bool}\to\mathsf{Bool}}.f\mathsf{true})(\lambda x^{\mathsf{Bool}}.\mathsf{false})$
Ex. 24	Ex. 5	$\mathsf{gensym}()$
Ex. 25	Ex. 7	$\mathsf{gensym}() = \mathsf{gensym}()$
Ex. 26		$\lambda y.\mathsf{gensym}()$
Ex. 27	Ex. 15	$\mathsf{let}\ x = \mathsf{gensym}()\ \mathsf{in}\ \lambda y.x$
Ex. 28		$\lambda x^{\mathsf{Nm}}.x = x$
Ex. 29	Ex. 13	$\mathsf{let}\ x = \mathsf{gensym}()\ \mathsf{in}\ \lambda y.x = y$
Ex. 30		$\lambda x.x = \mathsf{gensym}()$
Ex. 31	Ex. 14	$\mathsf{let}\ x = \mathsf{gensym}()\ \mathsf{in}\ \langle x, \lambda y.x = y\rangle$
Ex. 32		$\mathsf{let}\ x, y = \mathsf{gensym}()\ \mathsf{in}\ \lambda f^{\mathsf{Nm}\to\mathsf{Nm}}.((fx = y) \wedge (x = fy))$
Ex. 33	Ex. 16	Chain_p
Ex. 34	Ex. 17	$\mathsf{InaccessChain}_p$
Ex. 35	Ex. 20	$\mathsf{let}\ x = \mathsf{gensym}()\ \mathsf{in}\ \lambda f^{\mathsf{Nm}\to\mathsf{Bool}}.\mathsf{let}\ y = \mathsf{gensym}()\ \mathsf{in}\ fx = fy$
Ex. 36	Ex. 19	$\mathsf{let}\ x = \mathsf{gensym}()\ \mathsf{in}\ \mathsf{let}\ y = \mathsf{gensym}()\ \mathsf{in}\ \lambda f^{\mathsf{Nm}\to\mathsf{Bool}}.fx = fy$

Example 21. The simple application of the identity function to the Boolean constant true clearly returns true. The program $(\lambda x^{\mathsf{Bool}}.x)\mathsf{true}$ is reasoned about as follows and guarantees the expected result.

1	$\mathbb{\Gamma}+\delta+x \Vdash \{\mathsf{T}\}\ x :_a \{x = a\}$	$[\textsc{Var}]_\nu$
2	$\mathbb{\Gamma} \Vdash \{\mathsf{T}\}\ \lambda x^{\mathsf{Bool}}.x :_c \{\forall\delta.\forall x \in (\delta).c \bullet x = a\{a = x\}\}$	$[\textsc{Lam}]_\nu$, *1*
3	$\mathbb{\Gamma} \Vdash \{\mathsf{T}\}\ \lambda x^{\mathsf{Bool}}.x :_c \{c \bullet \mathsf{true} = a\{a = \mathsf{true}\}\}$	$[\textsc{Conseq}]_\nu$, $(utc1)$, $(u1)$, *2*
4	$\mathbb{\Gamma}+c \Vdash \{c \bullet \mathsf{true} = a\{a = \mathsf{true}\}\}\ \mathsf{true} :_d \{c \bullet d = a\{a = \mathsf{true}\}\}$	$[\textsc{Const}]_\nu$
5	$\mathbb{\Gamma} \Vdash \{\mathsf{T}\}\ (\lambda x^{\mathsf{Bool}}.x)\mathsf{true} :_a \{a = \mathsf{true}\}$	$[\textsc{App}]_\nu$, *3*, *4*

This proof is very similar to Ex. 1 in the λ-logic. The same rules are applied along with similar axioms (except $(utc1)$).

Example 22. The STLC program which takes a Boolean and returns the negation of it as follows $M_{22} \stackrel{def}{=} \lambda x^{\mathsf{Bool}}.\mathsf{if}\ x\ \mathsf{then\ false\ else\ true}$ as follows.

1	$\mathbb{\Gamma}+x \Vdash \{x = \mathsf{true}\}\ x :_d \{x = \mathsf{true} = d\}$	$[\textsc{Var}]_\nu$
2	$\mathbb{\Gamma}+x \Vdash \{(x = \mathsf{true} = d)[\mathsf{true}/d]\}\ \mathsf{false} :_b \{b = \mathsf{false}\}$	$[\textsc{Const}]_\nu$
3	$\mathbb{\Gamma}+x \Vdash \{(x = \mathsf{true} = d)[\mathsf{false}/d]\}\ \mathsf{true} :_b \{\mathsf{F}\}$	$[\textsc{Const}]_\nu$
4	$\mathbb{\Gamma}+x \Vdash \{x = \mathsf{true}\}\ \mathsf{if}\ x\ \mathsf{then\ false\ else\ true} :_b \{b = \mathsf{false}\}$	$[\textsc{If}]_\nu$, *1*, *2*, *3*
5	$\mathbb{\Gamma}+x \Vdash \{\mathsf{T}\}\ \mathsf{if}\ x\ \mathsf{then\ false\ else\ true} :_b \{x = \mathsf{true} \to b = \mathsf{false}\}$	$[\wedge \to]_\nu$, *4*
6	$\mathbb{\Gamma}+x \Vdash \{x = \mathsf{false}\}\ x :_d \{x = \mathsf{false} = d\}$	$[\textsc{Var}]_\nu$
7	$\mathbb{\Gamma}+x \Vdash \{(x = \mathsf{false} = d)[\mathsf{false}/d]\}\ \mathsf{false} :_b \{\mathsf{F}\}$	$[\textsc{Const}]_\nu$
8	$\mathbb{\Gamma}+x \Vdash \{(x = \mathsf{false} = d)[\mathsf{true}/d]\}\ \mathsf{true} :_b \{b = \mathsf{true}\}$	$[\textsc{Const}]_\nu$
9	$\mathbb{\Gamma}+x \Vdash \{x = \mathsf{false}\}\ \mathsf{if}\ x\ \mathsf{then\ false\ else\ true} :_b \{b = \mathsf{true}\}$	$[\textsc{If}]_\nu$, *6*, *7*, *8*
10	$\mathbb{\Gamma}+x \Vdash \{\mathsf{T}\}\ \mathsf{if}\ x\ \mathsf{then\ false\ else\ true} :_b \{x = \mathsf{false} \to b = \mathsf{true}\}$	$[\wedge \to]_\nu$, *9*
11	$\mathbb{\Gamma}+x \Vdash \{\mathsf{T}\}\ \mathsf{if}\ x\ \mathsf{then\ false\ else\ true} :_b \{x = \neg x\}$	$[\wedge\text{-}\textsc{Post}]_\nu$, *5*, *10*
12	$\mathbb{\Gamma} \Vdash \{\mathsf{T}\}\ M_{22} :_a \{\forall\delta.\forall x^{\mathsf{Bool}} \in (\delta).a \bullet x = \neg x\}$	$[\textsc{Lam}]_\nu$, *11*
13	$\mathbb{\Gamma} \Vdash \{\mathsf{T}\}\ M_{22} :_a \{\forall x^{\mathsf{Bool}} \in (\emptyset).a \bullet x = \neg x\}$	$[\textsc{Conseq}]_\nu$, $(utc1)$, $(u1)$, *12*

The last line is not necessary but is equivalent to the line 12 via $(utc2)$ and $(u5)$ which allows quantification over Bool types to add/remove the LTCs freely from the quantifier.

This proof is very similar to Ex. 2 in the λ-logic. The same rules are applied along with similar axioms (except $(utc1)$).

Example 23. The STLC program $(\lambda f^{\mathsf{Bool}\to\mathsf{Bool}}.f\mathsf{true})(\lambda x^{\mathsf{Bool}}.\mathsf{false})$ which should clearly return false is reasoned about as follows.

1	Let $A_{23}(f) \equiv f \bullet \mathsf{true} = \mathsf{false}$	
2	Let $B_{23}(c) \equiv \forall\delta.\forall f \in (\delta).A_{23}(f) \to c \bullet f = \mathsf{false}$	
3	$\mathbb{\Gamma}+\delta+f \Vdash \{A_{23}(f)\}\ f :_g \{A_{23}(g)\}$	$[\textsc{Var}]_\nu$
4	$\mathbb{\Gamma}+\delta+f+g \Vdash \{A_{23}(g)\}\ \mathsf{true} :_h \{g \bullet h = \mathsf{false}\}$	$[\textsc{Const}]_\nu$
5	$\mathbb{\Gamma}+\delta+f \Vdash \{A_{23}(f)\}\ f\mathsf{true} :_a \{a = \mathsf{false}\}$	$[\textsc{App}]_\nu$, *3, 4*
6	$\mathbb{\Gamma} \Vdash \{\mathsf{T}\}\ \lambda f.f\mathsf{true} :_c \{B_{23}(c)\}$	$[\textsc{Lam}]_\nu$, *5*
7	$\mathbb{\Gamma}+c+\delta'+x \Vdash \{\mathsf{T}\}\ \mathsf{false} :_a \{a = \mathsf{false}\}$	$[\textsc{Const}]_\nu$
8	$\mathbb{\Gamma}+c \Vdash \{\mathsf{T}\}\ \lambda x^{\mathsf{Bool}}.\mathsf{false} :_d \{\forall\delta.\forall x \in (\delta').d \bullet x = \mathsf{false}\}$	$[\textsc{Lam}]_\nu$, *7*
9	$\mathbb{\Gamma}+c \Vdash \{\mathsf{T}\}\ \lambda x^{\mathsf{Bool}}.\mathsf{false} :_d \{A_{23}(d)\}$	$[\textsc{Conseq}]_\nu$, $(utc1)$, $(u1)$, *8*
10	$\mathbb{\Gamma}+c \Vdash \{B_{23}(c)\}\ \lambda x.\mathsf{false} :_d \{B_{23}(c)\ \wedge\ A_{23}(f)\}$	$[\textsc{Invar}]_\nu$, *9*
11	$\mathbb{\Gamma}+c \Vdash \{B_{23}(c)\}\ \lambda x.\mathsf{false} :_d \{c \bullet d = \mathsf{false}\}$	$[\textsc{Conseq}]_\nu$, *M.P., 10*
12	$\mathbb{\Gamma} \Vdash \{\mathsf{T}\}\ (\lambda f^{\mathsf{Bool}\to\mathsf{Bool}}.f\mathsf{true})(\lambda x^{\mathsf{Bool}}.\mathsf{false}) :_a \{a = \mathsf{false}\}$	$[\textsc{App}]_\nu$, *6, 11*

This proof is very similar to Ex. 3, and has a similar result.

Example 24. The important program $\mathsf{gensym}()$ is reasoned about in an LTC $\mathbb{\Gamma}$, as follows. In line 2, $(utc1)$ instantiates the post-condition to $b \bullet () = a\{a\#\mathbb{\Gamma}+b\}$ and in line 3 $(f2)$ removes the b from the LTC to ensure the post-condition satisfies the $\textsc{Thin}_{Syn}(b)$ requirement in $[\textsc{App}]_\nu$. This proof holds under any LTC $\mathbb{\Gamma}$ and will be used throughout future examples under many different LTCs.

1	$\mathbb{\Gamma} \Vdash \{\mathsf{T}\}\ \mathsf{gensym} :_b \{\forall\delta.b \bullet () = a\{a\#\delta\}\}$	$[\textsc{Gensym}]_\nu$
2	$\mathbb{\Gamma} \Vdash \{\mathsf{T}\}\ \mathsf{gensym} :_b \{b \bullet () = a\{a\#\mathbb{\Gamma}+b\}\}$	$[\textsc{Conseq}]_\nu$, *(utc1), 1*
3	$\mathbb{\Gamma} \Vdash \{\mathsf{T}\}\ \mathsf{gensym} :_b \{b \bullet () = a\{a\#\mathbb{\Gamma}\}\}$	$[\textsc{Conseq}]_\nu$, *(f2), 2*
4	$\mathbb{\Gamma}+b \Vdash \{b \bullet () = a\{a\#\mathbb{\Gamma}\}\}\ () :_c \{b \bullet c = a\{a\#\mathbb{\Gamma}\}\}$	$[\textsc{Const}]_\nu$
5	$\mathbb{\Gamma} \Vdash \{\mathsf{T}\}\ \mathsf{gensym}() :_a \{a\#\mathbb{\Gamma}\}$	$[\textsc{App}]_\nu$, *3, 4*

Example 25. The following program compares gensym() with another gensym(), clearly this returns false. This is reasoned about by applying Ex. 24 twice with different LTCs such that the second LTC contains the anchor of the first, as follows. In line 3, the axiom $(f1)$ allows the $a \neq b$ inequality to be derived from line 2 given $a \in \mathbb{\Gamma}{+}a$.

1	$\mathbb{\Gamma} \Vdash \{\mathsf{T}\}\ \mathsf{gensym}()\ :_a\ \{a\#\mathbb{\Gamma}\}$	*See Ex. 24*
2	$\mathbb{\Gamma}{+}a \Vdash \{\mathsf{T}\}\ \mathsf{gensym}()\ :_b\ \{b\#\mathbb{\Gamma}{+}a\}$	*See Ex. 24*
3	$\mathbb{\Gamma}{+}a \Vdash \{a\#\mathbb{\Gamma}\}\ \mathsf{gensym}()\ :_b\ \{a \neq b\}$	$[\textsc{Conseq}]_\nu$, $(f1)$, *2*
4	$\mathbb{\Gamma} \Vdash \{\mathsf{T}\}\ \mathsf{gensym}() = \mathsf{gensym}()\ :_u\ \{u = \mathsf{false}\}$	$[\textsc{Eq}]_\nu$, *3*

Example 26. Placing name generation inside an abstraction halts the production of fresh names until the function is applied.

1	$\mathbb{\Gamma}{+}\delta{+}y \Vdash \{\mathsf{T}\}\ \mathsf{gensym}()\ :_m\ \{m\#\mathbb{\Gamma}{+}\delta{+}y\}$	*See Ex. 24*
2	$\mathbb{\Gamma} \Vdash \{\mathsf{T}\}\ \lambda y.\mathsf{gensym}()\ :_u\ \{\forall\delta.\forall y \in (\delta).u \bullet y = m\{m\#\mathbb{\Gamma}{+}\delta{+}y\}\}$	$[\textsc{Lam}]_\nu$, *1*

If y is of type Unit then this specification is identical to that of gensym, given $(utc1)$ and $(u1)$ and $(f2)$ imply $(\forall\delta.\forall y \in (\delta).u \bullet y = m\{m\#\mathbb{\Gamma}{+}\delta{+}y\}) \rightarrow (u \bullet () = m\{m\#\mathbb{\Gamma}\})$, the post-condition of $[\textsc{Gensym}]_\nu$. If $y : \alpha \neq \mathsf{Unit}$ then applying this function to a value of type α ensures $\forall\delta.\forall y \in (\delta).$ are instantiated at the point of application, making m fresh from any name derivable from the LTC at the point of application.

Example 27. Generating a name outside an abstraction and returning that same name in the function is often compared to Ex. 26 [2, 62]. The difference is that Ex. 26 returns a fresh name each time it is applied, whereas once let x = gensym() in $\lambda y.x$ has been evaluated it will always return the same name when applied.

1	$\mathrm{Let}A_{27}(p) \stackrel{def}{=} \forall\delta.\forall y \in (\delta).u \bullet y = m\{m\#\mathbb{\Gamma} \wedge p = m\}$	
2	$\{x\#\mathbb{\Gamma}\}\ x\ :_m\ \{m\#\mathbb{\Gamma} \wedge x = m\}$	$[\textsc{Var}]_\nu$
3	$\{x\#\mathbb{\Gamma}\}\ \lambda y.x\ :_u\ \{A_{27}(x)\}$	$[\textsc{Lam}]_\nu$, *2*
4	$\{x\#\mathbb{\Gamma}\}\ \lambda y.x\ :_u\ \{\exists x' \in (u).A_{27}(x')\}$	$[\textsc{Conseq}]_\nu$, *3*
5	$\mathbb{\Gamma} \Vdash \{\mathsf{T}\}$ let x = gensym() in $\lambda y.x\ :_u\ \{\exists x' \in (u).A_{27}(x')\}$	$[\textsc{LetFresh}]_\nu$, *4*

Proof of line 4 above is shown below. Essentially this proves x is derivable from u and hence $A_{27}(x)$ can be written as $\exists x' \in (u).A_{27}(x')$, such that now $\exists x' \in (u).A_{27}(x')$-$\textsc{Thin}_{Syn}(x)$

holds which ensures $[\textsc{LetFresh}]_\nu$ can be used.

6	$A_{27}(x)$	
7	$A_{27}(x) \wedge A_{27}(x)$	$F.O.L.$
8	$A_{27}(x) \wedge \forall y \in (\emptyset).u \bullet y = m\{x = m\}$	$(utc1)$, $(u4)$
9	$A_{27}(x) \wedge \forall y \in (\emptyset).\exists x' \in (u{+}y).x = x'$	$(ex3)$
10	$A_{27}(x) \wedge \exists x' \in (u).x = x'$	$(ex4)$
11	$\exists x' \in (u).(A_{27}(x) \wedge x = x')$	$(ex2)$
12	$\exists x' \in (u).A_{27}(x')$	$(eq4)$

Example 28. The function that takes in a name and compares it to itself is as follows.

$$M_{28} \stackrel{def}{=} \lambda x^{\mathsf{Nm}}.x = x$$

This clearly returns true and can be reasoned about as such as follows.

1	$\{\mathsf{T}\}\ x :_b \{x = b\}$	$[\textsc{Var}]_\nu$
2	$\{x = b\}\ x :_c \{b = c\}$	$[\textsc{Var}]_\nu$
3	$\{\mathsf{T}\}\ x = x :_c \{d = \mathsf{true}\}$	$[\textsc{Eq}]_\nu$, *1, 2*
4	$\mathbb{\Gamma} \Vdash \{\mathsf{T}\}\ M_{28} :_a \{\forall \delta.\forall x^{\mathsf{Nm}} \in (\delta).a \bullet x = \mathsf{true}\}$	$[\textsc{Lam}]_\nu$, *3*

Example 29. Ex. 10 introduces the following program in order to demonstrate the subtlety of hidden names. The subtlety arises from the name being generated outside the λ-binder, but then being equated (= destructor) inside the λ-binder, hence the name can never be retrieved from the function, meaning it is hidden. Hence, any application of this function will always be to other names, meaning this function always returns false.

$$M_{29} \stackrel{def}{=} \mathsf{let}\ x = \mathsf{gensym}()\ \mathsf{in}\ \lambda y.x = y$$

The ν-logic is used to reason about M_{29} as follows.

1	$\{\mathsf{T}\}\ x = y :_m \{m = (x = y)\}$	$[\textsc{Eq}]_\nu$
2	$\{\mathsf{T}\}\ \lambda y.x = y :_u \{\forall\delta.\forall y \in (\delta).u \bullet y = (x = y)\}$	$[\textsc{Lam}]_\nu$, *1*
3	$\mathbb{\Gamma}{+}x \Vdash \{x\#\mathbb{\Gamma}\}\ \lambda y.x = y :_u \{x\#\mathbb{\Gamma} \wedge \forall\delta.\forall y \in (\delta).u \bullet y = (x = y)\}$	$[\textsc{Invar}]_\nu$, *2*
4	$\mathbb{\Gamma}{+}x \Vdash \{x\#\mathbb{\Gamma}\}\ \lambda y.x = y :_u \{\forall y \in (\mathbb{\Gamma}{+}u).u \bullet y = \mathsf{false}\}$	$[\textsc{Conseq}]_\nu$, *3*
5	$\mathbb{\Gamma} \Vdash \{\mathsf{T}\}\ M_{29} :_u \{\forall y \in (\mathbb{\Gamma}{+}u).u \bullet y = \mathsf{false}\}$	$[\textsc{LetFresh}]_\nu$, *4*
6	$\mathbb{\Gamma} \Vdash \{\mathsf{T}\}\ M_{29} :_u \{\forall\delta.\forall y^{\mathsf{Nm}} \in (\delta).u \bullet y = \mathsf{false}\}$	$[\textsc{Conseq}]_\nu$, $(utc4)$, *5*

To prove line 4 above the axioms are applied as follows.

7	$\mathbb{\Gamma}{+}x{+}u \Vdash x\#\mathbb{\Gamma} \wedge \forall\delta.\forall y \in (\delta).u \bullet y = (x = y)$	
8	$x\#\mathbb{\Gamma} \wedge \forall y \in (\mathbb{\Gamma}{+}x{+}u).u \bullet y = (x = y)$	$(utc1)$
9	$x\#\mathbb{\Gamma}{+}u \wedge \forall y \in (\mathbb{\Gamma}{+}x{+}u).\ u \bullet y = (x = y)$	$(f3)$
10	$x\#\mathbb{\Gamma}{+}u \wedge \forall y \in (\mathbb{\Gamma}{+}u).\ u \bullet y = (x = y)$	$(u4)$
11	$\forall y \in (\mathbb{\Gamma}{+}u).\ x\#\mathbb{\Gamma}{+}u{+}y \wedge u \bullet y = (x = y)$	$(f4)$
12	$\forall y \in (\mathbb{\Gamma}{+}u).\ x \neq y \wedge u \bullet y = (x = y)$	$(f1)$
13	$\forall y \in (\mathbb{\Gamma}{+}u).u \bullet y = \mathsf{false}$	$(e3)$

In lines 9 and 10, the freshness of x and the quantifier of y range over the same LTC, meaning y cannot quantify over x. Then using $(f4)$ this implies x is fresh from $\mathbb{\Gamma}{+}x{+}y$ and hence $(f4)$ implies $x \neq y$ in lines 11 and 12, which then obtains the result required.

Example 30. The function which takes a name and compares it to a fresh name clearly also returns false.

$$M_{30} \stackrel{def}{=} \lambda x^{\mathsf{Nm}}.x = \mathsf{gensym}()$$

The reasoning about M_{30} is as follows.

1	$\mathbb{\Gamma}+\delta+x \Vdash \{\mathsf{T}\}\ x :_b \{x = b\}$	$[\textsc{Var}]_\nu$
2	$\mathbb{\Gamma}+\delta+x+b \Vdash \{x = b\}\ \mathsf{gensym}() :_c \{c\#\mathbb{\Gamma}+\delta+x+b\}$	*See Example 24*
3	$\mathbb{\Gamma}+\delta+x+b \Vdash \{x = b\}\ \mathsf{gensym}() :_c \{b \neq c\}$	$[\textsc{Conseq}]_\nu$, $(f1)$, *2*
4	$\mathbb{\Gamma}+\delta+x \Vdash \{\mathsf{T}\}\ x = \mathsf{gensym}() :_c \{c = \mathsf{false}\}$	$[\textsc{Eq}]_\nu$, *1*, *3*
5	$\mathbb{\Gamma} \Vdash \{\mathsf{T}\}\ M_{30} :_a \{\forall\delta.\forall x^{\mathsf{Nm}} \in (\delta).a \bullet x = \mathsf{false}\}$	$[\textsc{Lam}]_\nu$, *4*

The final triple in line 5 contains the same pre/post-conditions as Ex. 29.

Example 31. To demonstrate the release of a hidden variable similar to Ex. 29 above but with the output being a pair with the function and the name as seen in Ex. 14:

$$M_{31} \stackrel{def}{=} \mathsf{let}\ x = \mathsf{gensym}()\ \mathsf{in}\ \langle x, \lambda y.x = y\rangle$$

The reasoning regarding M_{31} is as follows.

1	Let $A_{31}(p,q) \stackrel{def}{=} p\#\mathbb{\Gamma} \wedge \forall\delta.\forall y \in (\delta).q \bullet y = (p = y)$	
2	$\{x\#\mathbb{\Gamma}\}\ x :_b \{x = b \wedge x\#\mathbb{\Gamma}\}$	$[\textsc{Var}]_\nu$
3	$\{\mathsf{T}\}\ \lambda y.x = y :_c \{\forall\delta.\forall y \in (\delta).c \bullet y = (x = y)\}$	*See Example 29, Lines.1-2*
4	$\{x = b \wedge x\#\mathbb{\Gamma}\}\ \lambda y.x = y :_c \{x = b \wedge A_{31}(x,c)\}$	$[\textsc{Invar}]_\nu$, *3*
5	$\{x = b \wedge x\#\mathbb{\Gamma}\}\ \lambda y.x = y :_c \{(A_{31}(\pi_1(a),\pi_2(a)))\,[\langle b,c\rangle/a]\}$	$[\textsc{Conseq}]_\nu$, *See Below*
6	$\{x\#\mathbb{\Gamma}\}\ \langle x, \lambda y.x = y\rangle :_a \{A_{31}(\pi_1(a),\pi_2(a))\}$	$[\textsc{Pair}]_\nu$, *2*, *5*
7	$\mathbb{\Gamma} \Vdash \{\mathsf{T}\}\ M_{31} :_a \{A_{31}(\pi_1(a),\pi_2(a))\}$	$[\textsc{LetFresh}]_\nu$, *6*

The post-condition of line 6 is $\textsc{Thin}_{Syn}(x)$ ensuring the conditions of $[\textsc{LetFresh}]_\nu$ in line 7 are satisfied. The detailed proof of line 5 above is provided below.

8	$x = b \wedge A_{31}(x,c)$	
9	$A_{31}(b,c)$	$(eq4)$
10	$A_{31}(\pi_1(\langle b,c\rangle),\pi_2(\langle b,c\rangle))$	$(p1)$, $b = \pi_1(\langle b,c\rangle)$, $c = \pi_2(\langle b,c\rangle)$
11	$(A_{31}(\pi_1(a),\pi_2(a)))\,[\langle b,c\rangle/a]$	*Def. 51*

Example 32. An example from [68]:

$$M_{32} \stackrel{def}{=} \mathsf{let}\ x,y = \mathsf{gensym}()\ \mathsf{in}\ \lambda f^{\mathsf{Nm}\to\mathsf{Nm}}.(fx = y) \wedge (x = fy)$$

Where $(fx = y) \wedge (x = fy) \stackrel{def}{=} \mathsf{if}\ fx = y\ \mathsf{then}\ fx = y\ \mathsf{else}\ \mathsf{false}$. This should be equivalent to $\lambda f.\mathsf{false}$ as one might expect because the function f cannot distinguish x and y and so $(fx = y)$ and $(x = fy)$ must always return false. The program is reasoned about as follows.

1	Let: $G_{32} \equiv \mathsf{if}\ fx = y\ \mathsf{then}\ fx = y\ \mathsf{else}\ \mathsf{false}$	
2	Let: $F_{x,y,\mathbb{\Gamma}_0} \equiv x\#\mathbb{\Gamma}_0 \wedge y\#\mathbb{\Gamma}_0{+}x$	
3	Let: $B_f \equiv F_{x,y,\delta} \rightarrow f \bullet x = m\{f \bullet y = n\{n \neq x \vee m \neq y\}\}$	
4	$\{F_{x,y,\mathbb{\Gamma}} \wedge B_f\}\ fx :_m \{f \bullet y = n\{n \neq x \vee m \neq y\}\}$	$[\textsc{App}]_\nu$, $[\textsc{Var}]_\nu$
5	$\{f \bullet y = n\{n \neq x \vee m \neq y\}\}\ y :_e \{f \bullet y = n\{n \neq x \vee m \neq e\}\}$	$[\textsc{Var}]_\nu$
6	$\{F_{x,y,\mathbb{\Gamma}} \wedge B_f\}\ fx = y :_p \{f \bullet y = n\{n \neq x \vee p = \mathsf{false}\}\}$	$[\textsc{Eq}]_\nu$, *4*, *5*
7	$\{f \bullet y = n\{n \neq x\}\}\ fy :_n \{n \neq x\}$	$[\textsc{App}]_\nu$, $[\textsc{Var}]_\nu$
8	$\{n \neq x\}\ x :_k \{n \neq k\}$	$[\textsc{Var}]_\nu$
9	$\{(f \bullet y = n\{n \neq x \vee p = \mathsf{false}\})[\mathsf{true}/p]\}\ fy = x :_b \{b = \mathsf{false}\}$	$[\textsc{Eq}]_\nu$, *7*, *8*
10	$\{(f \bullet y = n\{n \neq x \vee p = \mathsf{false}\})[\mathsf{false}/p]\}\ \mathsf{false} :_b \{b = \mathsf{false}\}$	$[\textsc{Const}]_\nu$
11	$\{F_{x,y,\mathbb{\Gamma}} \wedge B_f\}\ G_{32} :_b \{b = \mathsf{false}\}$	$[\textsc{If}]_\nu$, *6*, *9*, *10*
12	$\{F_{x,y,\mathbb{\Gamma}}\}\ \lambda f.G_{32} :_a \{\forall\delta.\forall f \in (\delta).B_f \rightarrow a \bullet f = \mathsf{false}\}$	$[\textsc{Lam}]_\nu$, *11*
13	$\{F_{x,y,\mathbb{\Gamma}}\}\ \lambda f.G_{32} :_a \{\forall f \in (\mathbb{\Gamma}).a \bullet f = \mathsf{false}\}$	$[\textsc{Conseq}]_\nu$, $(u6)$, *12*
14	$\{x\#\mathbb{\Gamma}\}\ \mathsf{let}\ y = \mathsf{gensym}()\ \mathsf{in}\ \lambda f.G_{32} :_a \{\forall f \in (\mathbb{\Gamma}).a \bullet f = \mathsf{false}\}$	$[\textsc{Let}]_\nu$, *13*
15	$\{\mathsf{T}\}\ M_{32} :_a \{\forall f \in (\mathbb{\Gamma}).a \bullet f = \mathsf{false}\}$	$[\textsc{Let}]_\nu$, *14*
16	$\{\mathsf{T}\}\ M_{32} :_a \{\forall f \in (\mathbb{\Gamma}{+}a).a \bullet f = \mathsf{false}\}$	$[\textsc{Conseq}]_\nu$, $(u10)$, *15*
17	$\{\mathsf{T}\}\ M_{32} :_a \{\forall\delta.\forall f \in (\delta).a \bullet f = \mathsf{false}\}$	$[\textsc{Conseq}]_\nu$, $(utc5)$, *16*

The first part of the conditional is reasoned about in lines 4-6 where the true case is reasoned about in lines 7-9 and the false in line 10. The following lines 11-17 reason about the context the conditional occurs in. line 13 is proven by axiom $(u6)$ by construction of the axiom applied twice with the second application using symmetry of x and y as follows.

18	$\mathbb{\Gamma}+x+y+a \Vdash \forall\delta.\forall f \in (\delta).B_f \to a \bullet f = \mathsf{false}$	
19	$\to \forall f \in (\mathbb{\Gamma}).B_f \to a \bullet f = \mathsf{false}$	$(utc1)$, $(u4)$
20	$\to \forall f \in (\mathbb{\Gamma}).B_f \to a \bullet f = \mathsf{false}$ $\wedge\ x\#\mathbb{\Gamma} \wedge y\#\mathbb{\Gamma}+x \to \forall f \in (\mathbb{\Gamma}).f \bullet x = m\{m \neq y\}$ $(u6)$ $\wedge\ x\#\mathbb{\Gamma} \wedge y\#\mathbb{\Gamma}+x \to \forall f \in (\mathbb{\Gamma}).f \bullet y = n\{n \neq x\}$ $(u6)$	*Hold in* $\mathbb{\Gamma}+x+y+a$ *as* $\textsc{ExtInd}_{Syn}$
21	$\to \forall f \in (\mathbb{\Gamma}).B_f \to a \bullet f = \mathsf{false}$ $\wedge\ x\#\mathbb{\Gamma} \wedge y\#\mathbb{\Gamma}+x \ \to\ \forall f \in (\mathbb{\Gamma}).\ f \bullet x = m\{f \bullet y = n\{m \neq y \ \wedge\ n \neq x\}\}$	$(u3)$, $(e3)$, $F.O.L.$
22	$\to\ \forall f \in (\mathbb{\Gamma}).(B_f \to a \bullet f = \mathsf{false})\ \wedge\ B_f$	$(ex2)$, $F.O.L.$, $(u3)$

Example 33. The "Chain" example introduced in Ex. 16 [68] is a program that creates p number of fresh names and outputs a function which takes in name number x and outputs name number $x+1$ with the pth name looping back to the first name. The program Chain_p is defined as follows.

$$\begin{aligned}\mathsf{Chain}_p \stackrel{def}{=} \mathsf{let}\ x_0,\ ..., x_p = \mathsf{gensym}()\ \mathsf{in}\ \lambda x^{\mathsf{Nm}}.\ &\mathsf{if}\ x = x_0\ \mathsf{then}\ x_1\ \mathsf{else}\\ &\mathsf{if}\ x = x_1\ \mathsf{then}\ x_2\ \mathsf{else}\\ &...\\ &\mathsf{if}\ x = x_p\ \mathsf{then}\ x_0\ \mathsf{else}\ x_0\end{aligned}$$

The final conditional $\mathsf{if}\ x = x_p\ \mathsf{then}\ x_0\ \mathsf{else}\ x_0$ is clearly equivalent to x_0, hence the simplest informative example Chain_1 is equivalent to $\mathsf{Chain}'_1 \stackrel{def}{=} \mathsf{let}\ x_0, x_1 = \mathsf{gensym}()\ \mathsf{in}\ \lambda x.\mathsf{if}\ x = x_0\ \mathsf{then}\ x_1\ \mathsf{else}\ x_0$ and is reasoned about below.

1	Let: $N_{33} \stackrel{def}{=} \lambda x.\mathsf{if}\ x = x_0\ \mathsf{then}\ x_1\ \mathsf{else}\ x_0$	
2	Let: $F_p^{\#} \stackrel{def}{=} x_0 \# \mathbb{\Gamma} \wedge x_1 \# \mathbb{\Gamma} + x_0 \wedge \ldots \wedge x_p \# \mathbb{\Gamma} + x_0 + \ldots + x_{p-1}$	
3	Let: $A_{33}(a) \stackrel{def}{=} \exists x_0' \in (a).\exists x_1' \in (a + x_0').\forall \delta.\forall x \in (\delta).\ x \neq x_0' \rightarrow a \bullet x = x_0'$ $\wedge x = x_0' \rightarrow a \bullet x = x_1'$	
4	$\{F_1^{\#} \wedge x = x_0\}\ x = x_0 :_b \{b = \mathsf{true}\}$	$[\textsc{Eq}]_\nu$
5	$\{(b = \mathsf{true})[\mathsf{true}/b]\}\ x_1 :_c \{c = x_1\}$	$[\textsc{Var}]_\nu$
6	$\{(b = \mathsf{true})[\mathsf{false}/b]\}\ x_0 :_c \{c = x_1\}$	$[\textsc{Var}]_\nu$
7	$\{F_1^{\#} \wedge x = x_0\}\ \mathsf{if}\ x = x_0\ \mathsf{then}\ x_1\ \mathsf{else}\ x_0 :_c \{c = x_1\}$	$[\textsc{If}]_\nu$, *4, 5, 6*
8	$\{F_1^{\#}\}\ N_{33} :_a \{\forall \delta.\forall x \in (\delta).x = x_0 \rightarrow a \bullet x = x_1\}$	$[\textsc{Lam}]_\nu$, *7*
9	$\{F_1^{\#} \wedge x \neq x_0\}\ x = x_0 :_b \{b = \mathsf{false}\}$	$[\textsc{Eq}]_\nu$
10	$\{(b = \mathsf{false})[\mathsf{true}/b]\}\ x_1 :_c \{c = x_0\}$	$[\textsc{Var}]_\nu$
11	$\{(b = \mathsf{false})[\mathsf{false}/b]\}\ x_0 :_c \{c = x_0\}$	$[\textsc{Var}]_\nu$
12	$\{F_1^{\#} \wedge x \neq x_0\}\ \mathsf{if}\ x = x_0\ \mathsf{then}\ x_1\ \mathsf{else}\ x_0 :_c \{c = x_0\}$	$[\textsc{If}]_\nu$, *9, 10, 11*
13	$\{F_1^{\#}\}\ N_{33} :_a \{\forall \delta.\forall x \in (\delta).x \neq x_0 \rightarrow a \bullet x = x_0\}$	$[\textsc{Lam}]_\nu$, *12*
14	$\{F_1^{\#}\}\ N_{33} :_a \{\ \forall \delta.\forall x \in (\delta).x \neq x_0 \rightarrow a \bullet x = x_0\ \}$ $\wedge \forall \delta.\forall x \in (\delta).x = x_0 \rightarrow a \bullet x = x_1$	$[\wedge\text{-}\textsc{Post}]_\nu$, *8, 13*
15	$\{F_1^{\#}\}\ N_{33} :_a \{A_{33}(a)\}$	$[\textsc{Conseq}]_\nu$, *See Below*
16	$\{F_0^{\#}\}\ \mathsf{let}\ x_1 = \mathsf{gensym}()\ \mathsf{in}\ N_{33} :_a \{A_{33}(a)\}$	$[\textsc{LetFresh}]_\nu$, *15*
17	$\{\mathsf{T}\}\ \mathtt{Chain}_1' :_a \{A_{33}(a)\}$	$[\textsc{LetFresh}]_\nu$, *16*

The case where $x = x_0$ is reasoned about in lines 4-8 such that the function outputs x_1, whilst the case where $x \neq x_0$ is reasoned about in lines 9-13 such that the output is x_0. These are combined in line 14 which are then manipulated into the correct form in lines 14-17. Proof of line 15 is seen below. The proof manipulates the initial post-condition in line 14 into $A_{33}(a)$, such that $A_{33}(a)$-$\textsc{Thin}_{Syn}(x_1)$ and $A_{33}(a)$-$\textsc{Thin}_{Syn}(x_0)$ to satisfy the $[\textsc{LetFresh}]_\nu$ rules on lines 16 and 17 respectively.

18	$\forall\delta.\forall x \in (\delta).x \neq x_0 \rightarrow a \bullet x = x_0$ $\wedge\forall\delta.\forall x \in (\delta).x = x_0 \rightarrow a \bullet x = x_1$	
19	$\forall\delta.\forall x \in (\delta).\ x \neq x_0 \rightarrow a \bullet x = x_0\ \wedge x = x_0 \rightarrow a \bullet x = x_1$	$(utc3)$, $(u3)$
20	$\forall\delta.\forall x \in (\delta).\ x \neq x_0 \rightarrow a \bullet x = x_0\ \wedge x = x_0 \rightarrow a \bullet x = x_1$ $\wedge\ \forall\delta.\forall x \in (\delta).\ x \neq x_0 \rightarrow a \bullet x = x_0\ \wedge x = x_0 \rightarrow a \bullet x = x_1$ $\wedge\ \forall\delta.\forall x \in (\delta).\ x \neq x_0 \rightarrow a \bullet x = x_0\ \wedge x = x_0 \rightarrow a \bullet x = x_1$	*F.O.L.*
21	$\forall\delta.\forall x \in (\delta).\ x \neq x_0 \rightarrow a \bullet x = x_0\ \wedge x = x_0 \rightarrow a \bullet x = x_1$ $\wedge\ \forall x \in (\emptyset).\ x \neq x_0 \rightarrow a \bullet x = x_0\ \wedge x = x_0 \rightarrow a \bullet x = x_1$ $\wedge\ a \bullet x_0 = x_1$	$(utc1)$, $(u4)$
22	$\forall\delta.\forall x \in (\delta).\ x \neq x_0 \rightarrow a \bullet x = x_0\ \wedge x = x_0 \rightarrow a \bullet x = x_1$ $\wedge\ \forall x \in (\emptyset).\ a \bullet x = x_0$ $\wedge\ a \bullet x_0 = x_1$	$(u8)$, *M.P.*
23	$\forall\delta.\forall x \in (\delta).\ x \neq x_0 \rightarrow a \bullet x = x_0\ \wedge x = x_0 \rightarrow a \bullet x = x_1$ $\wedge\ (\forall x \in (\emptyset).\ \exists x'_0 \in (a{+}x).a \bullet x = x'_0)\quad \wedge\ \exists x'_1 \in (a{+}x_0).x'_1 = x_1$	$(ex3)$
24	$\forall\delta.\forall x \in (\delta).\ x \neq x_0 \rightarrow a \bullet x = x_0\ \wedge x = x_0 \rightarrow a \bullet x = x_1$ $\wedge\ \exists x'_0 \in (a).x_0 = x'_0\ \wedge\ \exists x'_1 \in (a{+}x_0).x'_1 = x_1$	$(ex4)$
25	$\exists x'_0 \in (a).\exists x'_1 \in (a{+}x'_0).\ \forall\delta.\forall x \in (\delta).\ x \neq x_0 \rightarrow a \bullet x = x_0$ $\wedge x = x_0 \rightarrow a \bullet x = x_1$ $\wedge\ x_0 = x'_0 \wedge\ x_1 = x'_1$	$(ex2)$
26	$\exists x'_0 \in (a).\exists x'_1 \in (a{+}x'_0).\ \forall\delta.\forall x \in (\delta).\ x \neq x'_0 \rightarrow a \bullet x = x'_0$ $\wedge x = x'_0 \rightarrow a \bullet x = x'_1$	$(eq4)$

In essence this proves that x_0 and x_1 can be derived from a and $a{+}x_0$ respectively and hence the axiom $(eq4)$ can be used to state $\exists x'_0 \in (a).\exists x'_1 \in (a)....$ to ensure that the formula is now $\textsc{Thin}_{Syn}(x_0, x_1)$. Although line 26 suffices in the proof above, ideally line 26 would read $\exists x'_0 \in (a).\exists x'_1 \in (a)....$ however further development of the axioms is required for this and is not introduced here.

A similar but more complex derivation for the program $\texttt{Chain}_p$ for $p > 1$ is not discussed here. The "lasso" example from Ex. 18 is also expected to hold similarly without any complications.

Example 34. The inaccessible chain is introduced in Ex. 17. The program $\mathsf{InaccessChain}_p$ is defined for a general p such that it produces a chain of names similar to Chain_p without access to any name in the chain, hence the output is equivalent to simply outputting a fresh name.

For simplicity, the program with $p = 1$ is reasoned about below but a similar reasoning strategy is expected to hold for $p \geq 2$.

$$\mathsf{InaccessChain}_1 \stackrel{def}{=} \mathsf{let}\ x_0, x_1 = \mathsf{gensym}()\ \mathsf{in}\ \lambda x^{\mathsf{Nm}}.\mathsf{if}\ x = x_0\ \mathsf{then}\ x_0\ \mathsf{else}\ x_1$$

Given $\mathsf{let}\ x_1, x_0 = \mathsf{gensym}()\ \mathsf{in}\ M \cong_{\xi}^{\mathring{\mathsf{a}}(M)} \mathsf{let}\ x_0, x_1 = \mathsf{gensym}()\ \mathsf{in}\ M$ due to Nominal determinacy, this ensures $\mathsf{InaccessChain}_1$ is identical to the example reasoned about below.

1	Let: $N_{34} \equiv \mathsf{if}\ x = x_0\ \mathsf{then}\ x_0\ \mathsf{else}\ x_1$	
2	Let: $F_{\#} \equiv x_1 \# \mathbb{\Gamma} \wedge x_0 \# \mathbb{\Gamma} + x_1$	
3	Let: $A_{34} \equiv \exists x_1' \in (a).\forall x \in (\mathbb{\Gamma} + a + x_1').a \bullet x = m\{x_1 \# \mathbb{\Gamma} \wedge m = x_1'\}$	
4	Let: $B_{34} \equiv \exists x_1' \in (a).\forall \delta.\forall x \in (\delta).a \bullet x = m\{x_1' \# \mathbb{\Gamma} \wedge m = x_1'\}$	
5	$\{\mathsf{T}\}\ x = x_0 :_b \{b = (x = x_0)\}$	$[\textsc{Eq}]_\nu$
6	$\{x = x_0\}\ x_0 :_c \{c = x_0\}$	$[\textsc{Var}]_\nu$
7	$\{\mathsf{T}\}\ x_0 :_c \{x = x_0 \to c = x_0\}$	$[\wedge \to]_\nu$, *6*
8	$\{x \neq x_0\}\ x_1 :_c \{x \neq x_0 \to c = x_1\}$	$[\textsc{Var}]_\nu$
9	$\{\mathsf{T}\}\ x_1 :_c \{x \neq x_0 \to c = x_1\}$	$[\wedge \to]_\nu$, *8*
10	$\{\mathsf{T}\}\ \mathsf{if}\ x = x_0\ \mathsf{then}\ x_0\ \mathsf{else}\ x_1 :_c \left\{ \begin{array}{l} x = x_0 \to c = x_0 \\ \wedge x \neq x_0 \to c = x_1 \end{array} \right\}$	$[\textsc{If}]_\nu$, *5, 7, 9*
11	$\{\mathsf{T}\}\ \lambda x^{\mathsf{Nm}}.N_{34} :_a \left\{ \forall \delta.\forall x \in (\delta).a \bullet x = c \left\{ \begin{array}{l} x = x_0 \to c = x_0 \\ \wedge x \neq x_0 \to c = x_1 \end{array} \right\} \right\}$	$[\textsc{Lam}]_\nu$, *10*
12	$\{F_{\#}\}\ \lambda x^{\mathsf{Nm}}.N_{34} :_a \left\{ F_{\#} \wedge \forall \delta.\forall x \in (\delta).a \bullet x = c \left\{ \begin{array}{l} x = x_0 \to c = x_0 \\ \wedge x \neq x_0 \to c = x_1 \end{array} \right\} \right\}$	$[\textsc{Invar}]_\nu$, *11*
13	$\{F_{\#}\}\ \lambda x^{\mathsf{Nm}}.N_{34} :_a \{A_{34}\}$	$[\textsc{Conseq}]_\nu$, *See Below*
14	$\{x_1 \# \mathbb{\Gamma}\}\ \mathsf{let}\ x_0 = \mathsf{gensym}()\ \mathsf{in}\ \lambda x^{\mathsf{Nm}}.N_{34} :_a \{A_{34}\}$	$[\textsc{LetFresh}]_\nu$, *13*
15	$\{\mathsf{T}\}\ \mathsf{let}\ x_1, x_0 = \mathsf{gensym}()\ \mathsf{in}\ \lambda x^{\mathsf{Nm}}.N_{34} :_a \{A_{34}\}$	$[\textsc{LetFresh}]_\nu$, *14*
16	$\{\mathsf{T}\}\ \mathsf{let}\ x_1, x_0 = \mathsf{gensym}()\ \mathsf{in}\ \lambda x^{\mathsf{Nm}}.N_{34} :_a \{B_{34}\}$	$[\textsc{Conseq}]_\nu$, $(utc4)$

The conditional is reasoned about in line 10, producing two outputs dependant on whether the name at x_0 is the input. line 13 above is proven below. The two cases $x = x_0$ and $x \neq x_0$ are forced into the latter case when the quantification for x is restricted to just the LTC $\mathbb{\Gamma}+x_1$ in line 19, which is then taken advantage of to show that all names derived from $\mathbb{\Gamma}+x_1$ are fresh from x_0 and hence the output of ax is always x_1. The second half of this proof, lines 21-26, takes advantage of the fact that x_1 can be derived from a to manipulate the formula to a $\textsc{Thin}_{Syn}(x_0, x_1)$ form. This manipulation is required to ensure the thinness requirement of the $[\textsc{LetFresh}]_\nu$ rules used in lines 10 and 11 are satisfied.

17	$F_\# \wedge \forall\delta.\forall x \in (\delta).a \bullet x = c \left\{ \begin{array}{l} x = x_0 \rightarrow c = x_0 \\ \wedge x \neq x_0 \rightarrow c = x_1 \end{array} \right\}$	
18	$F_\# \ \wedge \ \forall\delta.\forall x \in (\delta).x \neq x_0 \rightarrow a \bullet x = x_1$	*F.O.L.*
19	$x_1\#\mathbb{\Gamma} \ \wedge \ x_0\#\mathbb{\Gamma}+x_1 \ \wedge \ \forall x \in (\mathbb{\Gamma}+x_1).x \neq x_0 \rightarrow a \bullet x = x_1$	$(utc1)$, $(u4)$
20	$x_1\#\mathbb{\Gamma} \ \wedge \ \forall x \in (\mathbb{\Gamma}+x_1).x_0\#\mathbb{\Gamma}+x_1+x \wedge x \neq x_0 \rightarrow a \bullet x = x_1$	$(f4)$
21	$x_1\#\mathbb{\Gamma} \ \wedge \ \forall x \in (\mathbb{\Gamma}+x_1).a \bullet x = x_1$	$(f1)$, *M.P.*
22	$x_1\#\mathbb{\Gamma} \ \wedge \ \forall x \in (\mathbb{\Gamma}+x_1).a \bullet x = x_1$ $\wedge\forall x \in (\emptyset).\exists x_1' \in (a+x).x_1 = x_1'$	*F.O.L.*, $(utc1)$, $(u4)$, $(ex3)$
23	$x_1\#\mathbb{\Gamma} \ \wedge \ \forall x \in (\mathbb{\Gamma}+x_1+a).a \bullet x = x_1$ $\wedge\forall x \in (\emptyset).\exists x_1' \in (a+x).x_1 = x_1'$	$(u9)$
24	$x_1\#\mathbb{\Gamma} \ \wedge \ \forall x \in (\mathbb{\Gamma}+x_1+a).a \bullet x = x_1 \ \wedge \ \exists x_1' \in (a).x_1 = x_1'$	$(ex4)$
25	$\exists x_1' \in (a).x_1'\#\mathbb{\Gamma} \wedge \forall x \in (\mathbb{\Gamma}+a+x_1').a \bullet x = x_1'$	$(u4)$, $(ex2)$, $(eq4)$
26	$\exists x_1' \in (a).\forall x \in (\mathbb{\Gamma}+a+x_1').a \bullet x = m\{x_1'\#\mathbb{\Gamma} \wedge m = x_1'\}$	$(u2)$, $(u3)$, $(e3)$

Example 35. The alternative "Hard' 'example introduced in Ex. 20 is the program M'_H as follows.

$$M'_H \stackrel{def}{=} \mathsf{let}\ x = \mathsf{gensym}()\ \mathsf{in}\ \lambda f^{\mathsf{Nm}\to\ \mathsf{Bool}}.\mathsf{let}\ y = \mathsf{gensym}()\ \mathsf{in}\ fx = fy$$

1	Let: $G_{35} \stackrel{def}{=} \lambda f^{\mathsf{Nm}\to\ \mathsf{Bool}}.\mathsf{let}\ y = \mathsf{gensym}()\ \mathsf{in}\ fx = fy$	
2	Let: $F_x \equiv x\#\mathbb{\Gamma}$ and $F_{x,y} \equiv F_x \wedge y\#\mathbb{\Gamma}{+}x{+}f$	
3	Let: $B_f(D) \equiv \forall\delta_x.\forall x \in (\delta_x).x\#\mathbb{\Gamma} \to f \bullet x = c\{\forall\delta_y.\forall y \in (\delta_y).y\#\mathbb{\Gamma}{+}x \to D\}$	
4	Let: $A_{35}(a) \stackrel{def}{=} \forall\delta.\forall f \in (\delta).\ B_f(f \bullet y = d\{c = d\}) \to a \bullet f = \mathsf{true}$ $\wedge\ B_f(f \bullet y = d\{c \neq d\}) \to a \bullet f = \mathsf{false}$	
5	$\{f \bullet x = c\{f \bullet y = d\{c = d\}\}\}\ fx :_c \{f \bullet y = d\{c = d\}\}$	$[\textsc{App}]_\nu$, $[\textsc{Var}]_\nu$
6	$\{f \bullet y = d\{c = d\}\}\ fy :_d \{c = d\}$	$[\textsc{App}]_\nu$, $[\textsc{Var}]_\nu$
7	$\{f \bullet x = c\{f \bullet y = d\{c = d\}\}\}\ fx = fy :_b \{b = \mathsf{true}\}$	$[\textsc{Eq}]_\nu$, *5, 6*
8	$\{F_{x,y} \wedge B_f(f \bullet y = d\{c = d\})\}\ fx = fy :_b \{b = \mathsf{true}\}$	$[\textsc{Conseq}]_\nu$, $(utc1)$, $(u1)$, *M.P.*
9	$\{F_x \wedge B_{f1}(f \bullet y = d\{c = d\})\}\ \mathsf{let}\ y = \mathsf{gensym}()\ \mathsf{in}\ fx = fy :_b \{b = \mathsf{true}\}$	$[\textsc{Let}]_\nu$, *8*
10	$\{F_x\}\ G_{35} :_a \{\forall\delta.\forall f \in (\delta).B_f(f \bullet y = d\{c = d\}) \to a \bullet f = \mathsf{true}\}$	$[\textsc{Lam}]_\nu$, *9*
11	$\{F_x\}\ G_{35} :_a \{\forall\delta.\forall f \in (\delta).B_f(f \bullet y = d\{c \neq d\}) \to a \bullet f = \mathsf{false}\}$	*See 5-10 above*
12	$\{F_x\}\ G_{35} :_a \{A_{35}(a)\}$	$[\wedge\text{-}\textsc{Post}]_\nu$, *10, 11*
13	$\mathbb{\Gamma} \Vdash \{\mathsf{T}\}\ M'_H :_a \{A_{35}(a)\}$	$[\textsc{Let}]_\nu$, *12*

The final line 13, states that for any function f when applied to the fresh names x and y and the results are equated, such that: if the equating always returns true then this clearly implies $a \bullet f = \mathsf{true}$, and if the equating always returns false then this clearly implies $a \bullet f = \mathsf{false}$. This doesn't cover the case where some equating fx and fy return true whilst others return false, however if more information is known about the function f then this can be used, but in this general case this is the most general informative statement possible.

Example 36. The "Hard" example from the literature is the program that creates two fresh names and applies some function to these names, such that the function should not be able to distinguish these names as it has no access to them, as follows.

$$M_H \stackrel{def}{=} \mathsf{let}\ x = \mathsf{gensym}()\ \mathsf{in}\ \mathsf{let}\ y = \mathsf{gensym}()\ \mathsf{in}\ \lambda f^{\mathsf{Nm}\to\ \mathsf{Bool}}.fx = fy$$

One may typically assume that this function is equivalent to $\lambda f^{\mathsf{Nm}\to\mathsf{Bool}}.\mathsf{true}$, as the function f never has access to the names at x and y and so can never distinguish such names. Applying the function f to the two names must then output the same Boolean constant and hence the equality must return true. Typically this is the case, however there is a context introduced below as $C_H[\cdot]$, in which M_H fills the hole and at some point in the evaluation M_H (or its derived value) is applied to a function and the output is actually false. This does not mean the equality $M_H \cong^{\emptyset}_{(\mathsf{Nm}\to\mathsf{Bool})\to\mathsf{Bool}} \lambda f^{\mathsf{Nm}\to\mathsf{Bool}}.\mathsf{true}$ is false, but that in some situations the output of the function does not equate true and hence this should be represented as such in the logic. The logical reasoning about this program can be seen as follows.

1	Let: $F_{x,y} \equiv x\#\mathbb{\Gamma} \wedge y\#\mathbb{\Gamma}{+}x$	
2	Let: $B_f \equiv f \bullet x = c\{f \bullet y = d\{c = d\}\}$	
3	$\{B_f\}\ fx :_c \{f \bullet y = d\{c = d\}\}$	$[\textsc{App}]_\nu$, $[\textsc{Var}]_\nu$
4	$\{f \bullet y = d\{c = d\}\}\ fy :_d \{c = d\}$	$[\textsc{App}]_\nu$, $[\textsc{Var}]_\nu$
5	$\{F_{x,y} \wedge F_{x,y} \to B_f\}\ fx = fy :_b \{b = \mathsf{true}\}$	$[\textsc{Eq}]_\nu$, *3*, *4*
6	$\{F_{x,y}\}\ \lambda f.fx = fy :_a \{\forall\delta.\forall f \in (\delta).(F_{x,y} \to B_f) \to a \bullet f = \mathsf{true}\}$	$[\textsc{Lam}]_\nu$, *5*
7	$\{F_{x,y}\}\ \lambda f.fx = fy :_a \{\forall f \in (\mathbb{\Gamma}).(F_{x,y} \to B_f) \to a \bullet f = \mathsf{true}\}$	$[\textsc{Conseq}]_\nu$, $(utc1)$, $(u4)$
8	$\{F_{x,y}\}\ \lambda f.fx = fy :_a \{\forall f \in (\mathbb{\Gamma}).\ a \bullet f = \mathsf{true}\}$	$[\textsc{Conseq}]_\nu$, $(u7)$
9	$\{x\#\mathbb{\Gamma}\}\ \mathsf{let}\ y = \mathsf{gensym}()\ \mathsf{in}\ \lambda f.fx = fy :_a \{\forall f \in (\mathbb{\Gamma}).a \bullet f = \mathsf{true}\}$	$[\textsc{Let}]_\nu$
10	$\mathbb{\Gamma} \Vdash \{\mathsf{T}\}\ M_H :_a \{\forall f^{\mathsf{Nm}\to\ \mathsf{Bool}} \in (\mathbb{\Gamma}).a \bullet f = \mathsf{true}\}$	$[\textsc{Let}]_\nu$
11	$\mathbb{\Gamma} \Vdash \{\mathsf{T}\}\ M_H :_a \{\forall f^{\mathsf{Nm}\to\ \mathsf{Bool}} \in (\mathbb{\Gamma}{+}a).a \bullet f = \mathsf{true}\}$	$[\textsc{Conseq}]_\nu$, $(u10)$
12	$\mathbb{\Gamma} \Vdash \{\mathsf{T}\}\ M_H :_a \{\forall\delta.\forall f^{\mathsf{Nm}\to\ \mathsf{Bool}} \in (\delta).a \bullet f = \mathsf{true}\}$	$[\textsc{Conseq}]_\nu$, $(utc5)$

The post-condition in line 6 is not $\textsc{Thin}_{Syn}(x \text{ or } y)$ (or $\textsc{Thin}_{Sem}(x \text{ or } y)$) and hence

cannot be used in the $[\textsc{Let}]_\nu$ rule and hence lines 7 and 8 remove the x and y dependence. However the post-condition in line 10 only quantifies over the functions f derivable from the LTC $\mathbb{\Gamma}$, which cannot be used to reason about the program $C_H[M_H]$. This requires quantification derived from the LTC $\mathbb{\Gamma}+a$ or preferably any future LTC δ. Axiom $(u10)$ extends the restricting LTC in the f quantifier and hence $(utc5)$ can now be used to introduce the $\forall\delta.$ from which f is now derived. This triple ensures that any future application of a to some future derived f returns true.

The reduction evaluation of $C_H[M_H]$ (defined below), can be seen in Ex. 19.

$$C_H[\cdot] \stackrel{def}{=} \mathsf{let}\ g = [\cdot]\ \mathsf{in}\ \mathsf{let}\ h = \lambda y^{\mathsf{Nm}}.g(\lambda z^{\mathsf{Nm}}.y = z)\ \mathsf{in}\ gh$$

In the case of $C_H[M_H]$, if M_H evaluates to V_H, then two internal applications of V_H return false, however the most external application returns true. This is not captured in the reasoning for $C_H[M_H]$ which uses the reasoning above in lines 1-12 as follows.

13	Let: $C(g) \equiv \forall\delta.\forall f^{\mathsf{Nm}\to\ \mathsf{Bool}} \in (\delta).g \bullet f = \mathsf{true}$	
14	$\{\mathsf{T}\}\ M_H :_g \{C(g)\}$	*See Line.12 above*
15	$\{\mathsf{T}\}\ \lambda v.g(\lambda w.v = w) :_h \{\mathsf{T}\}$	$[\textsc{Lam}]_\nu$, *Trivial*
16	$\{C(g)\}\ \lambda v.g(\lambda w.v = w) :_h \{g \bullet h = \mathsf{true}\}$	$[\textsc{Invar}]_\nu$, $[\textsc{Conseq}]_\nu$, $(utc1)$, $(u1)$, *15*
17	$\{g \bullet h = \mathsf{true}\}\ gh :_a \{a = \mathsf{true}\}$	$[\textsc{App}]_\nu$, $[\textsc{Var}]_\nu$
18	$\{C(g)\}\ \mathsf{let}\ h = \lambda v.g(\lambda w.v = w)\ \mathsf{in}\ gh :_a \{a = \mathsf{true}\}$	$[\textsc{Let}]_\nu$, *16, 17*
19	$\{\mathsf{T}\}\ C_H[M_H] :_a \{a = \mathsf{true}\}$	$[\textsc{Let}]_\nu$, *14, 18*

7.1 Summary

Numerous examples have been reasoned about using the ν-logic. Programs in the STLC can be reasoned about using this logic as seen in Ex. 21-23, with the results that one would expect. All interesting programs introduced in Sec. 3.1.2 are reasoned about effectively; including the "Chain" example and the "Hard" example with the expected results. Some programs which were not reasoned about include the more complicated versions of Chain_p, Lasso_p and $\mathsf{InaccessChain}_{(p,i)}$, although these are expected to hold similarly to the smaller p cases that were reasoned about. This covers most key programs in the literature and some new programs that have caused issue over the construction of the logic. Many axioms are introduced for specific examples, thus a more general set of axioms that allow for reasoning about more complicated examples, is of course desirable.

Chapter 8

Conclusion

In the introduction the question was posed:

> Can a simple sequential language with names be reasoned about using a Hoare-style program logic in a simple manner that allows for compositional reasoning?

This thesis has affirmatively answered this question. A program logic for the ν_{GS}-calculus is introduced to reason compositionally about this simple program language with fresh name generation and name comparison.

The program logic introduced in Chapt. 4 builds on the idea that names are generated and used in a sequential manner such that if a name only appears hidden in a function, then any future use of the name must be derived from the function and hence cannot reveal the hidden name. This motivates the introduction of a new logical quantifier which does not quantify over all values, but instead quantifies over all values derivable from a set of variables. This ensures that names that have been previously generated and appear in the "state" can be used only in the form they are created. The state is represented by LTCs, which are introduced to define the variables from which quantification restricts over.

The application of gensym produces a name which is fresh from the current nameset, however future applications of gensym need to produce a name which is fresh from that future state's nameset. Similarly, quantification restricted to only the current set of variables (or state) is insufficient. A new quantification over all future states (which extend the current state) is introduced which denotes the future state by a TCV. The TCV allows for the naming of the future state, which can be used within the LTC to restrict quantification. Deriving values from future states maintains the hidden names but allows access to future derived values (which themselves cannot reveal hidden names).

The LTCs reflect the extension element of TCVs and variables through the requirement of order. The ordering implies that if a variable is added to an LTC then it must have

been derived from the LTC, and for a TCV to be added to an LTC it must represent an extension of the LTC.

These concepts are formally introduced in the logic in Chapt. 4, which includes the syntax for these new constructors. The logic is typed using LTCs, inspired by the typing rules of the λ-logic. The logical formulae are reasoned about via axioms and axiom schemas that convey the intentions described above, often inspired by their traditional F.O.L. counterparts. Some new axioms are introduced regarding the restricting LTC.

Hoare triples for the ν-calculus are restricted to static syntax, which ensures names are not used directly in the logic, but referred to only via variables. The logic of triples is introduced through rules which capture the operational semantics of the new name-related operators, yet also maintain the reasoning about the STLC operators.

A model based on the ordered LTC is introduced such that semantics of logical formulae and Hoare triples match the intended definitions. The soundness proof of the ν-logic with respect to the model and semantics introduced is provided. No claim or proof of any form of completeness for this logic is made.

Most interesting examples found in the literature [52, 62, 2, 68] are reasoned about compositionally using the ν-logic. The properties of these programs match the derived triples for each program.

8.1 Directions for Future Work

The work in this thesis has various directions in which it could be extended or developed. These are discussed in the following sections.

8.1.1 Generalisations of the Axioms

The logical axioms in Sec. 4.5 cover the cases required in the reasoning examples in Chapt. 7. However this logic for axioms may be further abstracted to a more complete set of axioms. For instance, axioms $(u9)$ and $(u10)$ represent similar cases of extending the LTC in a universal restricted quantification, and axioms $(u6)$ and $(u7)$ also obtain similar forms, taking fresh names x and y and a function that accepts names. A more general form of these axioms may exist which prove these axioms but also capture more cases. The axioms inspired by the F.O.L. axioms such as $(u1)$, $(u2)$, $(u3)$, $(ex1)$, $(ex2)$ are most likely in their most general form. However it is expected that many of the other axioms may have potential to be generalised.

It is possible that the separation of $\forall x \in (\mathbb{\Gamma}).A$ into $\forall x.\ x \in \mathbb{\Gamma} \rightarrow A$ may facilitate

the search for a more general set of axioms for the ν-logic, where $x \in \mathbb{\Gamma}$ is a separate logical formulae meaning "derived". The splitting of quantification and "derived" may prove fruitful if a more general set of axioms are found for "derived" alongside the standard F.O.L. axioms.

The ν-logic presented here holds only for static syntax. Extending this logic to all programs (i.e. those which include names directly) would be of interest. Complications are likely to arise through the introduction of names into the logical expressions which are not currently considered here.

8.1.2 Related Logics

It would be interesting to investigate hybrid logic and nominal logic in the context of the ν-logic. There are clear similarities in both logics which may give further insights into the ν-logic and vice-versa.

Hybrid Logics

The quantification over LTCs in the ν-logic, is essentially the model operator $\Box$ in modal (or temporal) logic with the additional ability to name the future state. This idea is not new. Hybrid logics extend modal and temporal logic with the idea of naming states and the ability to revert back to these states [10, 9, 8]. The original ideas for temporal hybrid logic are ascribed to Arthur N. Prior [55, 56] and more recent developments to hybrid logics are ascribed to Peter Blackburn, et al. [10, 9, 8].

Hybrid logic builds on standard modal logic including the operator $\Box$. The extension includes the $\downarrow_\delta A$ logical operator, which names the current state δ (or any variable) which may be used in A. Hybrid logic also has the ability to revert back to particular (already defined) states in the form of the logical operator $@_\delta A$, meaning at state δ, then A holds. Originally, hybrid logic does not do much with these states except evaluate them for validity.

A similarity between the ν-logic and the hybrid logic may exist. The obvious starting point for a translation is $\forall \delta.A \sim \Box \downarrow_\delta A$, stating for any future state $\Box$, if the state is named δ via $\downarrow_\delta$, then A holds using the variable δ. Similarly the translation $\forall x \in (\mathbb{\Gamma}).A \sim @_{\mathbb{\Gamma}} \forall x.A$ states that reverting back to state $\mathbb{\Gamma}$ via $@_{\mathbb{\Gamma}}$ then quantifying over all values in this state $\forall x$, then A holds. This later translation requires care for the definition of universal quantification, as normally the universal quantifier would quantify over all names available, which is not the intended result as hidden names still need to remain hidden.

The ν-calculus has the ability to extend and remove TCVs and variables from the LTCs. A more fundamental approach to LTCs may exist. This cannot be represented in original versions of hybrid logics but with further research this may be a possibility.

Nominal Logic

In 2019 Andrew Pitts and Murdoch Gabbay were awarded the Alonzo Church Award for their introduction of nominal techniques in [50, 22, 51]. Nominal techniques use nominal sets to provide a mathematical theory and logic for most key concepts regarding names. The nominal logic [50], bears many similarities to the ν-logic and ideas from nominal logic are likely to progress the ν-logic further. In nominal logic the new logical quantifier $\mathsf{N}x.A$ states that for any fresh name x the logical formula A holds. This resembles the ν-logic formula $\forall x^{\mathsf{Nm}} \in (\emptyset).A$, which states that for any fresh name x, A holds. Further work could explore the axioms of nominal logic in the setting of the ν-calculus, or alternatively introducing aspect of the ν-logic (such as "derived" $x \in \mathbb{\Gamma}$) to nominal logic.

8.1.3 Mechanisation of Proofs

Although the soundness proof of the ν-logic can be seen in Chapt. 6, proving this in an automated theorem prover, such as Coq or Isabelle/HOL, would provide a cast iron guarantee of the proof.

8.1.4 Full Proof of Conservativity

A sketch of the proof of conservativity has been provided in Lem. 119, however a full proof would clarify that the ν-logic is a conservative extension of the λ-logic. Although this is expected to hold, this is left for future work.

8.1.5 Applications of Names

My initial reason for studying names was their use in meta-programming, although this is not the only application of names. The ν-logic could be adapted and applied to other programming language constructs which use names. The use of names in meta-programming could be reasoned about using ν-logic and the combination with other logical constructors would be an interesting avenue of research.

Applying the ν-logic to uses of names such as references to a local state may be of interest. Comparing the result to the Local-logic introduced in Sec. 2.2.3 would also be of interest. Other applications of names include objects (as in Java), exceptions, channels

(as in the π-calculus), cryptographic keys (as in the cryptographic lambda calculus) and more. These are all potential applications of the ν-logic with unknown consequences.

Bibliography

[1] S. Abramsky, D. R. Ghica, A. S. Murawski, C.-H. L. Ong, and I. D. B. Stark. Nominal Games and Full Abstraction for the Nu–Calculus. In *Proc. LICS*, pages 150–159, 2004. 63

[2] Nick Benton and Vasileios Koutavas. A Mechanized Bisimulation for the Nu-Calculus. Technical Report MSR-TR-2008-129, Microsoft, 2008. 49, 53, 55, 59, 60, 62, 164, 167, 182

[3] Karl S Berg, Soraya Delgado, Rae Okawa, Steven R Beissinger, and Jack W Bradbury. Contact calls are used for individual mate recognition in free-ranging green-rumped parrotlets, forpus passerinus. *Animal Behaviour*, 81(1):241–248, 2011. 2

[4] Martin Berger, Kohei Honda, and Nobuko Yoshida. Logics for imperative higher-order functions with aliasing and local state: Thee completeness results. 34, 39, 40

[5] Martin Berger, Kohei Honda, and Nobuko Yoshida. A Logical Analysis of Aliasing for Higher-Order Imperative Functions. In *Proc. ICFP*, pages 280–293, 2005. 40, 82, 149, 152

[6] Martin Berger and Laurence Tratt. Program Logics for Homogeneous Generative Run-Time Meta-Programming. *Logical Methods in Computer Science (LMCS)*, 11(1:5), 2015. 28

[7] Lars Birkedal and Aleš Bizjak. Lecture notes on IRIS: Higher-order concurrent separation logic. *Aarhus University*, 2020. 40

[8] Patrick Blackburn. Representation, reasoning, and relational structures: a hybrid logic manifesto. *Logic Journal of the IGPL*, 8(3):339–365, 2000. 183

[9] Patrick Blackburn and Jerry Seligman. Hybrid languages. *Journal of Logic, Language and Information*, 4(3):251–272, 1995. 72, 183

[10] Patrick Blackburn and Miroslava Tzakova. Hybrid languages and temporal logic. *Logic journal of IGPL*, 7(1), 1999. 183

[11] Cristiano Calcagno. *Semantic and logical properties of stateful programming*. PhD thesis, University of Genoa, 2002. 22, 23

[12] Alonzo Church. A set of postulates for the foundation of logic. *Annals of mathematics*, pages 346–366, 1932. 9

[13] Alonzo Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5(2):56–68, 1940. 10, 14, 17

[14] Stephen A. Cook. Soundness and completeness of an axiom system for program verification. *SIAM J. Comput.*, 7(1):70–90, 1978. 34

[15] Erik Crank and Matthias Felleisen. Parameter-passing and the lambda calculus. In *Proceedings of the 18th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 233–244, 1991. 13

[16] Haskell B Curry. An analysis of logical substitution. *American journal of mathematics*, 51(3):363–384, 1929. 9

[17] Harold Pancho Eliott and Martin Berger. A Program Logic for Fresh Name Generation. In *Proc. FSEN*, page TBC, 2021. v

[18] Matthias Felleisen and Robert Hieb. The Revised Report on the Syntactic Theories of Sequential Control State. *TCS*, 103:235–271, 1992. 64

[19] Maribel Fernandez, Murdoch J Gabbay, and Ian Mackie. Nominal rewriting systems. In *Proceedings of the 6th ACM SIGPLAN international conference on Principles and practice of declarative programming*, pages 108–119, 2004. 57

[20] Robert W. Floyd. Assigning Meaning to Programs. In *Proc. Symposia in Applied Mathematics*, pages 19–32, 1967. 18

[21] Gottlob Frege. Über sinn und bedeutung (translated as "sense and reference"). *Zeitschrift für Philosophie und philosophische Kritik*, 100:25–50, 1892. 3

[22] Murdoch J Gabbay and Andrew M Pitts. A new approach to abstract syntax with variable binding. *Formal aspects of computing*, 13(3):341–363, 2002. 46, 57, 184

[23] Kurt Gödel. *On undecidable propositions of formal mathematics systems*. Institute for Advanced Study, 1934. 9, 34

[24] C. A. R. Hoare. An Axiomatic Basis of Computer Programming. *CACM*, 12(10), 1969. 18

[25] Kohei Honda. Elementary structures in process theory (1): Sets with renaming. *Mathematical Structures in Computer Science*, 10(5):617–663, 2000. 57

[26] Kohei Honda, Martin Berger, and Nobuko Yoshida. Descriptive and Relative Completeness of Logics for Higher-Order Functions. In *Proc. ICALP*, pages 360–371, 2006. 34

[27] Kohei Honda and Nobuko Yoshida. Game theoretic analysis of call-by-value computation. In *International Colloquium on Automata, Languages, and Programming*, pages 225–236. Springer, 1997. 63

[28] Kohei Honda and Nobuko Yoshida. A compositional logic for polymorphic higher-order functions. In *Proc. PPDP'04*, pages 191–202. ACM Press, 2004. 23, 28, 30, 34, 40, 41, 149, 152

[29] Samin S Ishtiaq and Peter W O'hearn. BI as an assertion language for mutable data structures. In *Proceedings of the 28th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 14–26, 2001. 22

[30] Vincent M Janik. Whistle matching in wild bottlenose dolphins (tursiops truncatus). *Science*, 289(5483):1355–1357, 2000. 2

[31] Alan Jeffrey and Julian Rathke. Towards a theory of bisimulation for local names. In *Proceedings. 14th Symposium on Logic in Computer Science (Cat. No. PR00158)*, pages 56–66. IEEE, 1999. 62

[32] Stephanie L King and Vincent M Janik. Bottlenose dolphins can use learned vocal labels to address each other. *Proceedings of the National Academy of Sciences*, 110(32):13216–13221, 2013. 2

[33] Stephen C Kleene and J Barkley Rosser. The inconsistency of certain formal logics. *Annals of Mathematics*, pages 630–636, 1935. 10

[34] Vasileios Koutavas. *Reasoning about imperative and higher-order programs*. PhD thesis, Northeastern University, 2008. 62

[35] Saul A Kripke. Naming and necessity. In *Semantics of natural language*, pages 253–355. Springer, 1972. 3

[36] Steffen Lösch and Andrew M. Pitts. Relating two semantics of locally scoped names. In *CSL*, 2011. 42, 65, 66, 67

[37] John McCarthy, Michael I Levin, Paul W Abrahams, Daniel J Edwards, and Timothy P Hart. *LISP 1.5 programmer's manual*. MIT press, 1965. 43

[38] Elliot Mendelson. *Introduction to Mathematical Logic*. Wadsworth Inc., 1987. 26, 76, 77

[39] Robin Milner. A theory of type polymorphism in programming. *Journal of Computer and System Sciences*, 17(3):348–375, 1978. 15

[40] Robin Milner. Functions As Processes. In *Proc. ICALP*, pages 167–180, New York, NY, USA, 1990. Springer-Verlag New York, Inc. 42

[41] James Hiram Morris Jr. *Lambda-calculus models of programming languages*. PhD thesis, Massachusetts Institute of Technology, 1969. 17

[42] Aleksandar Nanevski, Amal Ahmed, Greg Morrisett, and Lars Birkedal. Abstract predicates and mutable adts in hoare type theory. In *European Symposium on Programming*, pages 189–204. Springer, 2007. 41

[43] Aleksandar Nanevski, Greg Morrisett, and Lars Birkedal. Hoare type theory, polymorphism and separation1. *Journal of Functional Programming*, 18(5-6):865–911, 2008. 41

[44] Aleksandar Nanevski and Greg Gregory Morrisett. Dependent type theory of stateful higher-order functions. 2005. 41

[45] Roger M Needham and Sape J Mullender. Names. *Distributed systems*, 2:315–327, 1989. 3

[46] Martin Odersky. *A syntactic theory of local names*. Yale University. Department of Computer Science, 1993. 42, 63, 64

[47] Martin Odersky. A Functional Theory of Local Names. In *Proc. POPL*, pages 48–59, 1994. 63, 66

[48] Peter O'Hearn, John Reynolds, and Hongseok Yang. Local reasoning about programs that alter data structures. In *International Workshop on Computer Science Logic*, pages 1–19. Springer, 2001. 22

[49] Peter W O'Hearn. Incorrectness logic. *Proceedings of the ACM on Programming Languages*, 4(POPL):1–32, 2019. 22

[50] A. M. Pitts. Nominal logic, a first order theory of names and binding. *Information and Computation*, 186:165–193, 2003. 46, 57, 184

[51] A. M. Pitts. *Nominal Sets: Names and Symmetry in Computer Science*. CUP, 2013. 57, 184

[52] Andrew M. Pitts and Ian David Bede Stark. Observable Properties of Higher Order Functions that Dynamically Create Local Names, or What's new? In *MFCS*, 1993. iv, 42, 48, 49, 53, 55, 57, 60, 61, 62, 164, 182

[53] Gordon Plotkin. Call-By-Name, Call-By-Value, and the λ-Calculus. *TCS*, 1(2):125–159, 1975. 66

[54] Emil L Post. Finite combinatory processes-formulation 1. *The journal of symbolic logic*, 1(3):103–105, 1936. 9

[55] Arthur Prior. Past, present and future. *Revue Philosophique de la France Et de l'Etranger*, 157:476–476, 1967. 72, 183

[56] Arthur Prior. *Papers on Time and Tense*. Oxford University Press UK, 1968. 183

[57] Yann Régis-Gianas and François Pottier. A hoare logic for call-by-value functional programs. In *International Conference on Mathematics of Program Construction*, pages 305–335. Springer, 2008. 41

[58] John C Reynolds. Intuitionistic reasoning about shared mutable data structure. *Millennial perspectives in computer science*, 2(1):303–321, 2000. 22

[59] John C. Reynolds. Separation logic: a logic for shared mutable data structures. In *Proc. LICS'02*, pages 55–74, 2002. 22, 40

[60] Marcin Sabok, Sam Staton, Dario Stein, and Michael Wolman. Probabilistic programming semantics for name generation. *arXiv preprint arXiv:2007.08638*, 2020. 63

[61] Moses Schönfinkel. Über die bausteine der mathematischen logik. *Mathematische annalen*, 92(3):305–316, 1924. 9

[62] Ian Stark. *Names and Higher Order Functions*. PhD thesis, University of Cambridge, 1994. Technical report 363, Univ. of Cambridge Computer Laboratory. 46, 49, 59, 60, 62, 167, 182

[63] Ian Stark. Names, equations, relations: Practical ways to reason about new. *Fundamenta Informaticae*, 33(4):369–396, 1998. 60, 61, 62

[64] Eijiro Sumii and Benjamin C Pierce. A bisimulation for type abstraction and recursion. *ACM SIGPLAN Notices*, 40(1):63–74, 2005. 62

[65] W. W. Tait. Intensional interpretations of functionals of finite type i. *The Journal of Symbolic Logic*, 32(2):198–212, 1967. 61

[66] Alan Mathison Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London mathematical society*, 2(1):230–265, 1937. 9

[67] Nikos Tzevelekos. Full abstraction for nominal general references. In *22nd Annual IEEE Symposium on Logic in Computer Science (LICS 2007)*, pages 399–410. IEEE, 2007. 63

[68] Nikos Tzevelekos. Program equivalence in a simple language with state. *Computer Languages, Systems and Structures*, 38:181–198, 2012. 46, 49, 52, 53, 55, 59, 60, 62, 63, 164, 170, 172, 182

[69] Dominique Unruh. Quantum relational Hoare logic. *Proceedings of the ACM on Programming Languages*, 3(POPL):1–31, 2019. 22

[70] Christian Urban and Christine Tasson. Nominal techniques in isabelle/hol. In *International Conference on Automated Deduction*, pages 38–53. Springer, 2005. 57

[71] Mingsheng Ying. Floyd–Hoare logic for quantum programs. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 33(6):1–49, 2012. 22

[72] Nobuko Yoshida, Kohei Honda, and Martin Berger. Logical reasoning for higher-order functions with local state. In *Proc. Fossacs*, LNCS, pages 361–377, 2007. 28, 34, 35, 36, 37, 39, 40, 73, 81, 82, 149, 152

[73] Yu Zhang. *Logical relations for names*. PhD thesis, Masters thesis, University of Paris, 2002. 62

[74] Yu Zhang and David Nowak. Logical relations for dynamic name creation. In *International Workshop on Computer Science Logic*, pages 575–588. Springer, 2003. 60, 62

Appendix A

Deferred Proofs

A.1 Lemmas for Soundness of Syntactic Properties Implying Semantic Properties

To prove that A-ExtInd$_{Syn}$ implies A-ExtInd$_{Sem}$, two lemmas are first introduced to factor out the two most complicated cases: $A^{-\delta} - \text{ExtInd}_{Syn} \rightarrow \forall\delta.\forall x \in (\delta).A^{-\delta} - \text{ExtInd}_{Syn}$ and

Lemma 120 (Constructing ExtInd$_{Syn}$formulae from $\forall\delta.\forall x \in (\delta).$)**.**

$$\forall A.\ A^{-\delta} - \text{ExtInd}_{Syn} \rightarrow \forall\delta.\forall x \in (\delta).A^{-\delta} - \text{ExtInd}_{Syn}$$

Proof.

1	Assume $A^{-\delta}$ -ExtInd$_{Syn}$	
2	$\leftrightarrow\ \forall\, \mathbb{\Gamma}, \xi_x^{\mathbb{\Gamma}+x:\alpha}, \xi_x'^{\mathbb{\Gamma}',x:\alpha}.\ (\mathbb{\Gamma}+x:\alpha \Vdash A\ \wedge\ \xi_x \preccurlyeq^\star \xi_x') \rightarrow (\xi_x \models A \leftrightarrow \xi_x' \models A)$	
3	Show: $\forall\, \mathbb{\Gamma}, \xi^{\mathbb{\Gamma}}, \xi'^{\mathbb{\Gamma}'}.\ \mathbb{\Gamma} \Vdash \forall\delta.\forall x \in (\delta).A^{-\delta}\ \wedge\ \xi \preccurlyeq^\star \xi' \rightarrow \begin{pmatrix} \xi \models \forall\delta.\forall x \in (\delta).A \\ \leftrightarrow\ \xi' \models \forall\delta.\forall x \in (\delta).A \end{pmatrix}$	
4	Hence assume: $\mathbb{\Gamma}, \xi^{\mathbb{\Gamma}}, \xi'^{\mathbb{\Gamma}'}$ s.t. $\mathbb{\Gamma} \Vdash \forall\delta.\forall x \in (\delta).A\ \wedge\ \xi \preccurlyeq^\star \xi'$	
5	$\rightarrow$: Show that: $\xi \models \forall\delta.\forall x \in (\delta).A \rightarrow \xi' \models \forall\delta.\forall x \in (\delta).A$	*Line.7 below*
6	$\leftarrow$: Show that: $\xi' \models \forall\delta.\forall x \in (\delta).A \rightarrow \xi \models \forall\delta.\forall x \in (\delta).A$	*Line.13 below*

$\to$:

7	Let: $\xi_{1d} \equiv \xi_1 \cdot \delta : \mathbb{\Gamma}_1 \backslash_{-TCV}$ and $\xi'_{2d} \equiv \xi'_2 \cdot \delta : \mathbb{\Gamma}_2 \backslash_{-TCV}$	
8	$\xi \preccurlyeq^\star \xi' \ \wedge \ \xi \models \forall \delta . \forall x \in (\delta) . A$	
9	$\leftrightarrow \ \xi \preccurlyeq^\star \xi' \ \wedge \ \forall \xi_1^{\mathbb{\Gamma}_1} . \ \xi \preccurlyeq^\star \xi_1 \ \to \ \forall M, V. \ M \overset{[\delta,\ \xi_{1d}]}{\rightsquigarrow} V \ \to \ \xi_{1d} \cdot x : V \models A$	*Sem.*$\forall \delta . \forall x \in (\delta).$
10	$\leftrightarrow \ \xi \preccurlyeq^\star \xi' \ \wedge \ \forall \xi_1^{\mathbb{\Gamma}_1} . \ \xi \preccurlyeq^\star \xi_1 \ \to \ \forall M, V. \ M \overset{[\delta,\ \xi_{1d}]}{\rightsquigarrow} V \ \to \ \xi_1 \cdot x : V \models A$	*Lem. 105*
11	$\to \ \xi \preccurlyeq^\star \xi' \ \wedge \ \forall \xi_2'^{\mathbb{\Gamma}'_2} . \ \xi' \preccurlyeq^\star \xi'_2 \ \to \ \forall M, V. \ M \overset{[\delta,\ \xi'_{2d}]}{\rightsquigarrow} V \ \to \ \xi'_2 \cdot x : V \models A$	*select* ξ'_2 *s.t.* $\xi' \preccurlyeq^\star \xi'_2$
12	$\leftrightarrow \ \xi' \models \forall \delta . \forall x \in (\delta) . A$	*Lem. 105*, *Sem.*$\forall \delta . \forall x \in (\delta).$

$\leftarrow$:

13	Let: $\xi'_{2d} \equiv \xi'_2 \cdot \delta : \mathbb{\Gamma}_2 \backslash_{-TCV}$ and $\xi_{1d} \equiv \xi_1 \cdot \delta : \mathbb{\Gamma}_1 \backslash_{-TCV}$ and $\xi'_{1d} \equiv \xi_1 \cdot \delta : \mathbb{\Gamma}'_1 \backslash_{-TCV}$	
14	$\xi \preccurlyeq^\star \xi' \ \wedge \ \xi' \models \forall \delta . \forall x \in (\delta) . A$	
15	$\leftrightarrow \ \forall \xi_2'^{\mathbb{\Gamma}'_2} . \ \xi' \preccurlyeq^\star \xi'_2 \ \rightarrow \ \forall M, V. \ M \overset{[\delta,\ \xi'_{2d}]}{\rightsquigarrow} V \ \rightarrow \ \xi'_{2d} \cdot x : V \models A$	*Sem.*$\forall \delta . \forall x \in (\delta)$.
16	$\leftrightarrow \ \forall \xi_2'^{\mathbb{\Gamma}'_2} . \ \xi' \preccurlyeq^\star \xi'_2 \ \rightarrow \ \forall M. \ M \overset{[\delta,\ \xi'_{2d}]}{\rightsquigarrow} V \ \rightarrow \ \xi'_2 \cdot x : V \models A$	*Lem. 105*
17	$\leftrightarrow \ \forall \xi_1^{\mathbb{\Gamma}_1} . \ \xi \preccurlyeq^\star \xi_1$ $\rightarrow \ \forall \xi_2'^{\mathbb{\Gamma}'_2} . \xi' \preccurlyeq^\star \xi'_2 \ \rightarrow \ \forall M, V. \ M \overset{[\delta,\ \xi'_{2d}]}{\rightsquigarrow} V \ \rightarrow \ \xi'_2 \cdot x : V \models A$	*Intro* ξ_1
18	$\leftrightarrow \ \forall (\xi \cdot \tilde{\xi}_1)^{\mathbb{\Gamma}_1} . \ \xi \preccurlyeq^\star \xi \cdot \tilde{\xi}_1$ $\rightarrow \ \forall (\xi \cdot \tilde{\xi}' \cdot \tilde{\xi}'_2)^{\mathbb{\Gamma}'_2} . \ \ \xi \cdot \tilde{\xi}' \preccurlyeq^\star \xi \cdot \tilde{\xi}' \cdot \tilde{\xi}'_2$ $\rightarrow \ \forall M, V. \ M \overset{[\delta,\ \xi'_{2d}]}{\rightsquigarrow} V \ \rightarrow \ \xi'_{2d} \cdot x : V \models A$	*write* $\xi' \equiv \xi \cdot \tilde{\xi}'$ $\xi_1 \equiv \xi \cdot \tilde{\xi}_1$ $\xi'_2 \equiv \xi' \cdot \tilde{\xi}'_2$
19	$\leftrightarrow \ \forall \tilde{\xi}_1 . \ \xi \preccurlyeq^\star (\xi \cdot \tilde{\xi}_1)^{\mathbb{\Gamma}_1} \ \wedge \ \xi \cdot \tilde{\xi}' \preccurlyeq^\star (\xi \cdot \tilde{\xi}' \cdot \tilde{\xi}_1)^{\mathbb{\Gamma}'_1}$ $\rightarrow \ \forall M, V. \ M \overset{[\delta,\ \xi'_{1d}]}{\rightsquigarrow} V \ \rightarrow \ \xi \cdot \tilde{\xi}' \cdot \tilde{\xi}_1 \cdot x : V \models A$	*Select* $\tilde{\xi}'_2$ *as* $\tilde{\xi}_1$ $\mathbb{\Gamma}'_2 \equiv \mathbb{\Gamma}'_1$ *F.O.L.*
20	$\rightarrow \ \forall \tilde{\xi}_1 . \ \mathring{\mathrm{a}}(\tilde{\xi}_1) \cap \mathring{\mathrm{a}}(\tilde{\xi}') \subseteq \mathring{\mathrm{a}}(\xi)$ $\wedge \ \xi \preccurlyeq^\star (\xi \cdot \tilde{\xi}_1)^{\mathbb{\Gamma}_1} \ \wedge \ \xi \cdot \tilde{\xi}' \preccurlyeq^\star (\xi \cdot \tilde{\xi}' \cdot \tilde{\xi}_1)^{\mathbb{\Gamma}'_1}$ $\rightarrow \ \forall M, V. M \overset{[\delta,\ \xi'_{1d}]}{\rightsquigarrow} V \ \rightarrow \ \xi \cdot \tilde{\xi}' \cdot \tilde{\xi}_1 \cdot x : V \models A$	*Subset of* $\forall \tilde{\xi}_1$
21	$\leftrightarrow \ \forall \tilde{\xi}_1 . \ \mathring{\mathrm{a}}(\tilde{\xi}_1) \cap \mathring{\mathrm{a}}(\tilde{\xi}') \subseteq \mathring{\mathrm{a}}(\xi)$ $\wedge \ \xi \preccurlyeq^\star (\xi \cdot \tilde{\xi}_1)^{\mathbb{\Gamma}_1} \ \wedge \ \xi \cdot \tilde{\xi}_1 \preccurlyeq^\star (\xi \cdot \tilde{\xi}_1 \cdot \tilde{\xi}')^{\mathbb{\Gamma}'_1}$ $\rightarrow \ \forall M, V. \ M \overset{[\delta,\ \xi'_{1d}]}{\rightsquigarrow} V \ \rightarrow \ \xi \cdot \tilde{\xi}_1 \cdot \tilde{\xi}' \cdot x : V \models A$	*Lem. 104*
22	$\rightarrow \ \forall \tilde{\xi}_1 . \ \mathring{\mathrm{a}}(\tilde{\xi}_1) \cap \mathring{\mathrm{a}}(\tilde{\xi}') \subseteq \mathring{\mathrm{a}}(\xi)$ $\wedge \ \xi \preccurlyeq^\star (\xi \cdot \tilde{\xi}_1)^{\mathbb{\Gamma}_1} \ \wedge \ \xi \cdot \tilde{\xi}_1 \preccurlyeq^\star (\xi \cdot \tilde{\xi}_1 \cdot \tilde{\xi}')^{\mathbb{\Gamma}'_1}$ $\rightarrow \ \forall M, V. M \overset{[\delta,\ \xi_{1d}]}{\rightsquigarrow} V \ \rightarrow \ \xi \cdot \tilde{\xi}_1 \cdot \tilde{\xi}' \cdot x : V \models A$	*Subset of possible M 's* $\mathbb{\Gamma}'_1 \Vdash \mathbb{\Gamma}_1$ *Lem. 102*
23	$\rightarrow \ \forall \xi_1 . \ \mathring{\mathrm{a}}(\xi_1) \cap \mathring{\mathrm{a}}(\tilde{\xi}') \subseteq \mathring{\mathrm{a}}(\xi)$ $\wedge \ \xi \preccurlyeq^\star \xi_1^{\mathbb{\Gamma}_1} \ \wedge \ \xi_1 \preccurlyeq^\star (\xi_1 \cdot \tilde{\xi}')^{\mathbb{\Gamma}'_1}$ $\rightarrow \ \forall M, V. M \overset{[\delta,\ \xi_{1d}]}{\rightsquigarrow} V \ \rightarrow \ \xi_1 \cdot x : V \models A$	*Lem. 104* A -ExtInd$_{Syn}$
24	$\rightarrow \ \forall \xi_1 . \ \xi \preccurlyeq^\star \xi_1^{\mathbb{\Gamma}_1} \rightarrow \ \forall M, V. \ M \overset{[\delta,\ \xi_{1d}]}{\rightsquigarrow} V \ \rightarrow \ \xi_1 \cdot x : V \models A$	*Nominal Det. Def. 25*
25	$\leftrightarrow \ \forall \xi_1 . \ \xi \preccurlyeq^\star \xi_1^{\mathbb{\Gamma}_1} \ \rightarrow \ \forall M, V. \ M \overset{[\delta,\ \xi_{1d}]}{\rightsquigarrow} V \ \rightarrow \ \xi_{1d} \cdot x : V \models A$	*Lem. 105,* $A^{-\delta}$
26	$\leftrightarrow \ \xi \models \forall \delta . \forall x \in (\delta) . A$	*Sem.*$\forall \delta . \forall x \in (\mathbb{\Gamma} + \delta)$.

line 24 holds as any name appearing in ξ_1 in line 24 which doesn't appear in ξ_1 in line 23 can be introduced through a fresh name in ξ_1 from line 23 and swapping the names for

the required fresh name. □

Lemma 121 (The formula used in the reasoning for $\lambda x.\mathsf{gensym}()$ is $\textsc{ExtInd}_{Sem}$).

$$\forall\delta.\forall x \in (\delta).f \bullet x = b\{b\#\delta{+}x\}\text{-}\textsc{ExtInd}_{Sem}$$

Proof.

$$\forall\, \xi^{\mathbb{\Gamma}}, \xi'.\xi \preccurlyeq^{\star} \xi' \rightarrow \xi \models \forall\delta.\forall x \in (\delta).f{\bullet}x = b\{b\#\delta{+}x\} \leftrightarrow \xi' \models \forall\delta.\forall x \in (\delta).f{\bullet}x = b\{b\#\delta{+}x\}$$

Assume some $\mathbb{\Gamma}$ such that $\mathbb{\Gamma} \Vdash \forall\delta.\forall x \in (\delta).f \bullet x = b\{b\#\delta{+}x\}$. Assume some $\xi^{\mathbb{\Gamma}_d}$, ξ' such that $\mathbb{\Gamma} \triangleright \xi$ and $\xi \preccurlyeq^{\star} \xi'$ then prove the $\leftrightarrow$ as follows.

Extending $(\xi \models \forall\delta.\forall x \in (\delta).f{\bullet}x = b\{b\#\delta{+}x\} \rightarrow \xi' \models \forall\delta.\forall x \in (\delta).f{\bullet}x = b\{b\#\delta{+}x\})$:

1	Assume $\xi \models \forall\delta.\forall x \in (\delta).f \bullet x = b\{b\#\delta{+}x\}$	
2	$\leftrightarrow\ \forall\, \xi_1^{\mathbb{\Gamma}_1}.\xi \preccurlyeq^{\star} \xi_1 \rightarrow \xi_1 \cdot \delta : \mathbb{\Gamma}_1 \backslash_{-TCV} \models \forall x \in (\delta).f \bullet x = b\{b\#\delta{+}x\}$	*Sem.*$\forall\delta$.
3	$\rightarrow\ \forall\, \xi_1^{\mathbb{\Gamma}_1}.\xi \preccurlyeq^{\star} \xi' \preccurlyeq^{\star} \xi_1 \rightarrow \xi_1 \cdot \delta : \mathbb{\Gamma}_1 \backslash_{-TCV} \models \forall x \in (\delta).f \bullet x = b\{b\#\delta{+}x\}$	*Subset* $\forall\,\xi_1$
4	$\rightarrow\ \forall\, \xi_1^{\mathbb{\Gamma}_1}.\xi' \preccurlyeq^{\star} \xi_1 \rightarrow \xi_1 \cdot \delta : \mathbb{\Gamma}_1 \backslash_{-TCV} \models \forall x \in (\delta).f \bullet x = b\{b\#\delta{+}x\}$	*Remove* $\xi \preccurlyeq^{\star}$
5	$\rightarrow\ \xi' \models \forall\delta.\forall x \in (\delta).f \bullet x = b\{b\#\delta{+}x\}$	

Contracting $(\xi \models \forall\delta.\forall x \in (\delta).f \bullet x = b\{b\#\delta{+}x\} \leftarrow \xi' \models \forall\delta.\forall x \in (\delta).f \bullet x = b\{b\#\delta{+}x\})$:

6	Assume $\xi' \models \forall\delta.\forall x \in (\delta).f \bullet x = b\{b\#\delta{+}x\}$	
7	Let: $\xi'_{1d} \equiv \xi_1^{\mathbb{\Gamma}_1} \cdot \delta : \mathbb{\Gamma}_1 \backslash_{-TCV}$ $\quad (\equiv \xi \cdot \tilde{\xi}' \cdot \tilde{\xi}_1 \cdot \delta : (\mathbb{\Gamma}{+}\tilde{\mathbb{\Gamma}}'{+}\tilde{\mathbb{\Gamma}}_1)\backslash_{-TCV})$ and ξ'_{1d}	
8	$\leftrightarrow \forall \xi_1^{\mathbb{\Gamma}_1}.\ \xi' \preccurlyeq^\star \xi_1$ $\rightarrow \forall M_x, V_x. M_x \overset{[\delta,\ \xi'_{1d}]}{\rightsquigarrow} V_x \rightarrow \xi'_{1d} \cdot x : V_x \models f \bullet x = b\{b\#\delta{+}x\}$	*Sem.*$\forall\delta.$, $\forall x \in (\delta).$
9	$\leftrightarrow \forall \tilde{\xi}_1^{\tilde{\mathbb{\Gamma}}_1}.\ \xi \cdot \tilde{\xi}' \preccurlyeq^\star \xi \cdot \tilde{\xi}' \cdot \tilde{\xi}_1$ $\rightarrow \forall M_x, V_x.\ M_x \overset{[\delta,\ \xi'_{1d}]}{\rightsquigarrow} V_x$ $\rightarrow \xi'_{1d} \cdot x : V_x \models f \bullet x = b\{b\#\delta{+}x\}$	$\xi'^{\mathbb{\Gamma}'} \equiv \xi^{\mathbb{\Gamma}} \cdot \tilde{\xi}'^{\tilde{\mathbb{\Gamma}}'}$ $\xi_1^{\mathbb{\Gamma}_1} \equiv \xi^{\mathbb{\Gamma}} . \tilde{\xi}'^{\tilde{\mathbb{\Gamma}}'} . \tilde{\xi}_1^{\tilde{\mathbb{\Gamma}}_1}$
10	$\rightarrow \forall \tilde{\xi}_1^{\tilde{\mathbb{\Gamma}}_1}.\ \xi \cdot \tilde{\xi}' \preccurlyeq^\star \xi \cdot \tilde{\xi}' \cdot \tilde{\xi}_1 \wedge \xi \preccurlyeq^\star \xi \cdot \tilde{\xi}_1$ $\wedge\ \mathring{a}(\tilde{\xi}_1) \cap \mathring{a}(\tilde{\xi}') \subseteq \mathring{a}(\xi)$ $\rightarrow \forall M_x, V_x.\ M_x \overset{[\delta,\ \xi'_{1d}]}{\rightsquigarrow} V_x$ $\rightarrow \xi'_{1d} \cdot x : V_x \models f \bullet x = b\{b\#\delta{+}x\}$	*Subset* $\forall \tilde{\xi}_1$ *only* $\tilde{\xi}_1$ *s.t.* $\xi \preccurlyeq^\star \xi \cdot \tilde{\xi}_1$
11	$\rightarrow \forall \tilde{\xi}_1^{\tilde{\mathbb{\Gamma}}_1}.\ \xi \cdot \tilde{\xi}' \preccurlyeq^\star \xi \cdot \tilde{\xi}' \cdot \tilde{\xi}_1 \wedge \xi \preccurlyeq^\star \xi \cdot \tilde{\xi}_1$ $\wedge\ \mathring{a}(\tilde{\xi}_1) \cap \mathring{a}(\tilde{\xi}') \subseteq \mathring{a}(\xi)$ $\rightarrow \forall M_x, V_x.\ M_x \overset{[\mathbb{\Gamma}{+}\tilde{\mathbb{\Gamma}}'{+}\tilde{\mathbb{\Gamma}}_1,\ \xi'_{1d}\backslash\delta]}{\rightsquigarrow} V_x$ $\rightarrow \xi'_{1d} \cdot x : V_x \models f \bullet x = b\{b\#\delta{+}x\}$	*Lem. 102* $[\![\delta]\!]_{\xi_{1d}} \equiv \mathbb{\Gamma}{+}\tilde{\mathbb{\Gamma}}'{+}\tilde{\mathbb{\Gamma}}_1$
12	Let: $\xi_2 \equiv \xi \cdot \tilde{\xi}_1 \cdot \delta : (\mathbb{\Gamma}{+}\tilde{\mathbb{\Gamma}}_1)\backslash_{-TCV}$	
13	$\rightarrow \forall \tilde{\xi}_1^{\tilde{\mathbb{\Gamma}}_1}.\ \xi \cdot \tilde{\xi}' \preccurlyeq^\star \xi \cdot \tilde{\xi}' \cdot \tilde{\xi}_1 \wedge \xi \preccurlyeq^\star \xi \cdot \tilde{\xi}_1$ $\wedge\ \mathring{a}(\tilde{\xi}_1) \cap \mathring{a}(\tilde{\xi}') \subseteq \mathring{a}(\xi)$ $\rightarrow \forall M_x, V_x.M_x \overset{[\delta,\ \xi_2]}{\rightsquigarrow} V_x \rightarrow \xi'_{1d} \cdot x : V_x \models f \bullet x = b\{b\#\delta{+}x\}$	*Subset* $\forall M_x$ $\xi_2\backslash\delta \subseteq \xi_{1d}\backslash\delta$
14	$\rightarrow \forall \tilde{\xi}_1^{\tilde{\mathbb{\Gamma}}_1}.\ \xi \cdot \tilde{\xi}' \preccurlyeq^\star \xi \cdot \tilde{\xi}' \cdot \tilde{\xi}_1 \wedge \xi \preccurlyeq^\star \xi \cdot \tilde{\xi}_1$ $\wedge\ \mathring{a}(\tilde{\xi}_1) \cap \mathring{a}(\tilde{\xi}') \subseteq \mathring{a}(\xi)$ $\rightarrow \forall M_x, V_x.M_x \overset{[\delta,\ \xi_2]}{\rightsquigarrow} V_x \rightarrow \xi_2 \cdot x : V_x \models f \bullet x = b\{b\#\delta{+}x\}$	*See below, Line.18*
15	$\rightarrow \forall \tilde{\xi}_1^{\tilde{\mathbb{\Gamma}}_1}.\ \xi \preccurlyeq^\star \xi \cdot \tilde{\xi}_1 \rightarrow \xi'_{1d} \models \forall x \in (\delta).f \bullet x = b\{b\#\delta{+}x\}$	*Nominal Det. Def. 25*
16	$\rightarrow \forall \xi_1^{\mathbb{\Gamma}_1}.\xi \preccurlyeq^\star \xi_1 \rightarrow \xi'_{1d} \models f \bullet x = b\{b\#\delta{+}x\}$	$\xi_1^{\mathbb{\Gamma}_1} \equiv \xi^{\mathbb{\Gamma}} \cdot \tilde{\xi}_1^{\tilde{\mathbb{\Gamma}}_1}$
17	$\rightarrow \xi \models \forall\delta.f \bullet x = b\{b\#\delta{+}x\}$	*Sem.*$\forall\delta.$

Proof of

$$\forall\, \xi^{\mathbb{\Gamma}}, \tilde{\xi}'^{\mathbb{\Gamma}'}, \tilde{\xi}_1^{\tilde{\mathbb{\Gamma}}_1}.\ \xi\cdot\tilde{\xi}' \preccurlyeq^\star \xi\cdot\tilde{\xi}'\cdot\tilde{\xi}_1 \ \wedge\ \xi \preccurlyeq^\star \xi\cdot\tilde{\xi}_1$$
$$\wedge\ \mathring{a}(\tilde{\xi}_1)\cap\mathring{a}(\tilde{\xi}')\subseteq\mathring{a}(\xi)$$
$$\rightarrow\ \xi_{1d} \equiv \xi\cdot\tilde{\xi}'\cdot\tilde{\xi}_1\cdot\delta : (\mathbb{\Gamma}+\tilde{\mathbb{\Gamma}}'+\tilde{\mathbb{\Gamma}}_1)\backslash_{-TCV}$$
$$\wedge\ \xi_2 \equiv \xi\cdot\tilde{\xi}_1\cdot\delta : (\mathbb{\Gamma}+\tilde{\mathbb{\Gamma}}_1)\backslash_{-TCV}$$
$$\wedge\ \forall\, M_x, V_x.\ M_x \overset{[\mathbb{\Gamma}+\tilde{\mathbb{\Gamma}}_1,\ \xi\cdot\tilde{\xi}_1]}{\rightsquigarrow} V_x$$
$$\rightarrow\ \xi_{1d}\cdot x : V_x \models f\bullet x = b\{b\#\delta+x\}$$
$$\rightarrow\ \xi_2\cdot x : V_x \models f\bullet x = b\{b\#\delta+x\}$$

18	Assume some $\xi^{\mathbb{\Gamma}}$, $\tilde{\xi}'^{\mathbb{\Gamma}'}$, $\tilde{\xi}_1^{\tilde{\mathbb{\Gamma}}_1}$ s.t. $\xi\cdot\tilde{\xi}' \preccurlyeq^\star \xi\cdot\tilde{\xi}'\cdot\tilde{\xi}_1$, $\xi \preccurlyeq^\star \xi\cdot\tilde{\xi}_1$ and $\mathring{a}(\tilde{\xi}_1)\cap\mathring{a}(\tilde{\xi}')\subseteq\mathring{a}(\xi)$ then:	
19	Assume some M_x, V_x s.t. $M_x \overset{[\mathbb{\Gamma}+\tilde{\mathbb{\Gamma}}_d,\ \xi\cdot\tilde{\xi}_1]}{\rightsquigarrow} V_x$ then:	
20	$\mathring{a}(\tilde{\xi}_1)\cap\mathring{a}(\tilde{\xi}')\subseteq\mathring{a}(\xi) \rightarrow \xi\cdot\tilde{\xi}' \preccurlyeq^\star \xi\cdot\tilde{\xi}'\cdot\tilde{\xi}_1 \leftrightarrow \xi\cdot\tilde{\xi}_1 \preccurlyeq^\star \xi\cdot\tilde{\xi}'\cdot\tilde{\xi}_1$	*Lem. 104*
21	$\rightarrow \xi\cdot\tilde{\xi}_1\cdot b : \mathsf{n}_b \preccurlyeq^\star \xi\cdot\tilde{\xi}'\cdot\tilde{\xi}_1\cdot b : \mathsf{n}_b$	*Lem. 104* $\mathsf{n}_b \notin \mathring{a}(\xi\cdot\tilde{\xi}'\cdot\tilde{\xi}_1)$
22	Let $\xi_{3x}^{\mathbb{\Gamma}_{3x}} \equiv \xi\cdot\tilde{\xi}'\cdot\tilde{\xi}_1\cdot\delta : (\mathbb{\Gamma}+\tilde{\mathbb{\Gamma}}'+\tilde{\mathbb{\Gamma}}_1)\backslash_{-TCV}\cdot x : V_x$ and $\xi_{3xb} \equiv (\xi_{3x}\backslash\delta)\cdot b : \mathsf{n}_b$	
23	Let $\xi_{2x}^{\mathbb{\Gamma}_{2x}} \equiv \xi^{\mathbb{\Gamma}}\cdot\tilde{\xi}_1\cdot\delta : (\mathbb{\Gamma}+\tilde{\mathbb{\Gamma}}_1)\backslash_{-TCV}\cdot x : V_x$	
24	Assume: $\xi \preccurlyeq^\star \xi\cdot\tilde{\xi}' \ \wedge\ \xi\cdot\tilde{\xi}' \preccurlyeq^\star \xi\cdot\tilde{\xi}'\cdot\tilde{\xi}_1 \ \wedge\ \xi \preccurlyeq^\star \xi\cdot\tilde{\xi}_1$	
25	Assume: $\xi_{3x} \models f\bullet x = b\{b\#\delta+x\}$	
26	$\leftrightarrow\ \exists\, \mathsf{n}_b.\ fx \overset{[\mathbb{\Gamma}_{3x},\ \xi_{3x}]}{\rightsquigarrow} \mathsf{n}_b \ \wedge\ \neg\ \exists\, M_b.M_b \overset{[\delta+x,\ \xi_{3x}\cdot b:\mathsf{n}_b]}{\rightsquigarrow} \mathsf{n}_b$	*Sem.*$\bullet = \{\}$, #
27	$\leftrightarrow\ \exists\, \mathsf{n}_b.\ fx \overset{[\mathbb{\Gamma}_{2x},\ \xi_{2x}]}{\rightsquigarrow} \mathsf{n}_b \ \wedge\ \neg\ \exists\, M_b.M_b \overset{[\delta+x,\ \xi_{3x}\cdot b:\mathsf{n}_b]}{\rightsquigarrow} \mathsf{n}_b$	$f, x \in [\![\mathbb{\Gamma}_{2x}]\!]_{\xi_{2x}} \subseteq [\![\mathbb{\Gamma}_{3x}]\!]_{\xi_{3x}}$, *Lem. 102*
28	$\leftrightarrow\ \exists\, \mathsf{n}_b.\ fx \overset{[\mathbb{\Gamma}_{2x},\ \xi_{2x}]}{\rightsquigarrow} \mathsf{n}_b \ \wedge\ \neg\ \exists\, M_b.M_b \overset{[\mathbb{\Gamma}_{3x},\ \xi_{3xb}]}{\rightsquigarrow} \mathsf{n}_b$	$[\![\delta+x]\!]_{\xi_{3x}} = \mathbb{\Gamma}_{3x}$, *Lem. 102*
29	$\rightarrow\ \exists\, \mathsf{n}_b.\ fx \overset{[\mathbb{\Gamma}_{2x},\ \xi_{2x}]}{\rightsquigarrow} \mathsf{n}_b \ \wedge\ \neg\ \exists\, M_b.M_b \overset{[\mathbb{\Gamma}_{2x},\ \xi_{3xb}]}{\rightsquigarrow} \mathsf{n}_b$	*Subset*, $\mathbb{\Gamma}_{3x} \Vdash \mathbb{\Gamma}_{2x}$
30	$\rightarrow\ \exists\, \mathsf{n}_b.\ fx \overset{[\mathbb{\Gamma}_{2x},\ \xi_{2x}]}{\rightsquigarrow} \mathsf{n}_b \ \wedge\ \neg\ \exists\, M_b.M_b \overset{[\mathbb{\Gamma}_{2x},\ (\xi_{2x}\backslash\delta)\cdot b:\mathsf{n}_b]}{\rightsquigarrow} \mathsf{n}_b$	*Lem. 96, Line.20*
31	$\rightarrow\ \exists\, \mathsf{n}_b.\ fx \overset{[\mathbb{\Gamma}_{2x},\ \xi_{2x}]}{\rightsquigarrow} \mathsf{n}_b \ \wedge\ \neg\ \exists\, M_b.M_b \overset{[\delta+x,\ \xi_{2x}\cdot b:\mathsf{n}_b]}{\rightsquigarrow} \mathsf{n}_b$	$[\![\delta+x]\!]_{\xi_{2x}} = \mathbb{\Gamma}_{2x}$, *Lem. 102*
32	$\rightarrow \xi_{2x} \models f\bullet x = b\{b\#\delta+x\}$	

□

Lemma 122 ($\textsc{ExtInd}_{Syn}$ implies $\textsc{ExtInd}_{Sem}$)**.**

$$A\text{-}\textsc{ExtInd}_{Syn} \rightarrow A\text{-}\textsc{ExtInd}_{Sem}$$

Proof. Proof by induction on the structure of A, knowing that A is constructed using the definition of $\textsc{ExtInd}_{Syn}$ in Def. 54. Using Lem. 99 (i.e. $\mathbb{\Gamma}_0 \subseteq \mathbb{\Gamma} \rightarrow [\![\mathbb{\Gamma}_0]\!]_\xi \equiv [\![\mathbb{\Gamma}_0]\!]_{\xi'}$) and Lem. 100 (i.e. $[\![e]\!]_\xi \equiv [\![e]\!]_{\xi'}$) the proof builds on the structure of Def. 54 as follows.

1. Base Formulas:

$A \equiv \mathsf{T}, \mathsf{F}$ Clearly hold.

$A \equiv e\#\mathbb{\Gamma}_0$ $\mathbb{\Gamma} \Vdash e\#\mathbb{\Gamma}_0$ implies $\mathbb{\Gamma} \Vdash e : \mathsf{Nm}$ and $\mathbb{\Gamma} \Vdash \mathbb{\Gamma}_0$.
Lem. 100 implies $[\![e]\!]_\xi \equiv [\![e]\!]_{\xi'}$.
Lem. 99 implies $[\![\mathbb{\Gamma}_0]\!]_\xi \equiv [\![\mathbb{\Gamma}_0]\!]_{\xi'}$.
Lem. 102 implies $\exists M.\ M \overset{[\mathbb{\Gamma}_0,\ \xi]}{\rightsquigarrow} [\![e]\!]_\xi \leftrightarrow \exists M.\ M \overset{[\mathbb{\Gamma}_0,\ \xi']}{\rightsquigarrow} [\![e]\!]_{\xi'}$

2. Core Inductive Cases:

$A \equiv \neg A_1$ Holds from I.H. on A_1 as $\xi \models A_1 \leftrightarrow \xi' \models A_1$
hence $\xi \not\models A_1 \leftrightarrow \xi' \not\models A_1$, hence $\xi \models \neg A_1 \leftrightarrow \xi' \models \neg A_1$.

$A \equiv A_1 \wedge A_2$ Holds from I.H. on A_1 and A_2, as $\xi \models A_i \leftrightarrow \xi' \models A_i$ implies
$\xi \models A_1 \wedge A_2 \leftrightarrow \xi \models A_1 \wedge \xi \models A_2 \leftrightarrow \xi' \models A_1 \wedge \xi' \models A_2 \leftrightarrow \xi' \models A_1 \wedge A_2$.

$A \equiv A_1 \vee A_2$ Derivable given $A \vee B \equiv \neg(\neg A \wedge \neg B)$

$A \equiv A_1 \rightarrow A_2$ Derivable given $A \rightarrow B \equiv \neg(A \wedge \neg B)$

$A \equiv e \bullet e' = m\{A_1\}$ Given $\mathring{a}(V_x) \cap \mathring{a}(\xi') \subseteq \mathring{a}(\xi)$:
Lem. 102 implies $ee' \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_m \leftrightarrow ee' \overset{[\mathbb{\Gamma},\ \xi']}{\rightsquigarrow} V_m$.
Lem. 104 implies $\xi \preccurlyeq^\star \xi' \wedge ee' \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_m \rightarrow \xi \cdot m : V_m \preccurlyeq^\star \xi' \cdot m : V_m$
Induction on A_1 implies $\xi \cdot m : V_m \models A_1 \leftrightarrow \xi' \cdot m : V_m \models A_1$.
Hence $\xi \models e \bullet e' = m\{A_1\}$.
$\leftrightarrow \exists V_m.\ ee' \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_m \wedge \xi \cdot m : V_m \models A_1$
$\leftrightarrow \exists V_m.\ ee' \overset{[\mathbb{\Gamma},\ \xi']}{\rightsquigarrow} V_m \wedge \xi' \cdot m : V_m \models A_1$
$\leftrightarrow \xi' \models e \bullet e' = m\{A_1\}$

$A \equiv \forall x \in (\mathbb{\Gamma}_0).A_1$ Lem. 99 implies $[\![\mathbb{\Gamma}_0]\!]_\xi \equiv [\![\mathbb{\Gamma}_0]\!]_{\xi'}$
Given $\mathring{a}(V_x) \cap \mathring{a}(\xi') \subseteq \mathring{a}(\xi)$:
Lem. 102 implies $M \overset{[\mathbb{\Gamma}_0,\ \xi]}{\rightsquigarrow} V_x \leftrightarrow M \overset{[\mathbb{\Gamma}_0,\ \xi']}{\rightsquigarrow} V_x$
Lem. 104 implies $\xi \preccurlyeq^\star \xi' \wedge M_x \overset{[\mathbb{\Gamma}_0,\ \xi]}{\rightsquigarrow} V_x \rightarrow \xi \cdot x : V_x \preccurlyeq^\star \xi' \cdot x : V_x$
Induction on A_1 implies $\xi \cdot x : V_x \models A_1 \leftrightarrow \xi' \cdot x : V_x \models A_1$

Hence

$\xi \models \forall x \in (\mathbb{\Gamma}_0).A_1$

$\leftrightarrow \ \forall\, M, V.M \overset{[\mathbb{\Gamma}_0,\ \xi]}{\rightsquigarrow} V \ \rightarrow\ \xi \cdot x : V \models A_1$ Sem.$\forall x \in (\mathbb{\Gamma}_0)$.

$\leftrightarrow \ \forall\, M, V.M \overset{[\mathbb{\Gamma}_0,\ \xi']}{\rightsquigarrow} V \ \rightarrow\ \xi \cdot x : V \models A_1$ $[\![\mathbb{\Gamma}_0]\!]_\xi \equiv [\![\mathbb{\Gamma}_0]\!]_{\xi'}$ Lem. 102

$\leftrightarrow \ \forall\, M, V.M \overset{[\mathbb{\Gamma}_0,\ \xi']}{\rightsquigarrow} V \ \rightarrow\ \xi' \cdot x : V \models A_1$ I.H. on A_1

$\leftrightarrow \ \xi' \models \forall x \in (\mathbb{\Gamma}_0).A_1$ Sem.$\forall x \in (\mathbb{\Gamma}_0)$.

$A \equiv \exists x \in (\mathbb{\Gamma}_0).A_1$ Derivable from $\exists x \in (\mathbb{\Gamma}_0).A_1 \equiv \neg\forall x \in (\mathbb{\Gamma}_0).\neg A_1$

3. $\forall\delta$.-Inductive Cases:

$A \equiv \forall\delta.A_1$ holds by I.H. on A_1, Knowing that A_1 is δ-free then:

$\xi \models \forall\delta.A_1 \rightarrow \xi' \models \forall\delta.A_1$ clearly holds through the semantics.

$\xi' \models \forall\delta.A_1$

$\leftrightarrow \ \forall\, \xi_1^{\mathbb{\Gamma}_1}.\xi' \preccurlyeq^\star \xi_1 \rightarrow \xi_1 \cdot \delta : \mathbb{\Gamma}_1 \models A_1$ Sem.$\forall\delta$.

$\rightarrow \ \forall\, \xi_1^{\mathbb{\Gamma}_1}.\xi' \preccurlyeq^\star \xi_1 \rightarrow \xi_1 \models A_1$ Lem. 105 $A_1^{-\delta}$

$\rightarrow \ \xi' \models A_1$ $\forall\, \xi_1 \rightarrow \xi'$

$\rightarrow \ \xi \models A_1$ I.H.

$\rightarrow \ \forall\, \xi_2^{\mathbb{\Gamma}_2}.\xi \preccurlyeq^\star \xi_2 \ \rightarrow\ \xi_2 \models A_1$ I.H.

$\rightarrow \ \forall\, \xi_2^{\mathbb{\Gamma}_2}.\xi \preccurlyeq^\star \xi_2 \ \rightarrow\ \xi_2 \cdot \delta : \mathbb{\Gamma}_2 \models A_1$ Lem. 105 $A_1^{-\delta}$

$\rightarrow \ \xi \models \forall\delta.A_1$ Sem.$\forall\delta$.

$A \equiv \forall\delta.\forall x \in (\delta).A_1$ holds by I.H. on $A_1^{-\delta}$, Lem. 120

Let: $\xi_{1d}^{\mathbb{\Gamma}_{1d}} \equiv \xi_1 \cdot \delta : \mathbb{\Gamma}_1$ and $\xi_{1x}^{\mathbb{\Gamma}_{1x}} \equiv \xi_1 \cdot x : V_x$ and $\xi_{1dx}^{\mathbb{\Gamma}_{1dx}} \equiv \xi_{1dx} \cdot x : V_x$

$\xi \models \forall\delta.\forall x \in (\delta).A_1$

$\leftrightarrow \ \forall\, \xi_1^{\mathbb{\Gamma}_1}.\xi \preccurlyeq^\star \xi_1 \rightarrow \forall\, M, V_x.M \overset{[\delta,\ \xi_{1d}]}{\rightsquigarrow} V_x \rightarrow \xi_{1dx} \models A_1$ Sem.$\forall\delta.\forall x \in (\delta)$.

$\rightarrow \ \forall\, \xi_1^{\mathbb{\Gamma}_1}.\xi \preccurlyeq^\star \xi' \preccurlyeq^\star \xi_1 \rightarrow \forall\, M, V_x.M \overset{[\delta,\ \xi_{1d}]}{\rightsquigarrow} V_x \rightarrow \xi_{1dx} \models A_1$ Subset $\forall\, \xi_1$

$\leftrightarrow \ \xi' \models \forall\delta.\forall x \in (\delta).A_1$ Sem.$\forall\delta.\forall x \in (\delta)$.

For any j or $'$ version of these then let:

$\xi_{jd}^{\mathbb{\Gamma}_{jd}} \equiv \xi_j^{\mathbb{\Gamma}_j} \cdot \delta : \mathbb{\Gamma}_j \backslash_{-TCV}$ and $\xi_{jx}^{\mathbb{\Gamma}_{jx}} \equiv \xi_j \cdot x : V_x$ and $\xi_{jdx}^{\mathbb{\Gamma}_{jdx}} \equiv \xi_{jd} \cdot x : V_x$

Let: $\xi_d'^{\mathbb{\Gamma}'_d} \equiv \xi_2' \cdot \delta : \mathbb{\Gamma}_2 \backslash_{-TCV}$ and $\xi_{2d}'^{\mathbb{\Gamma}'_{2d}} \equiv \xi_2' \cdot \delta : \mathbb{\Gamma}_1' \backslash_{-TCV}$ and $\xi_{2x}' \equiv \xi_2' \cdot x : V_x$ and $\xi_1' \equiv \xi_2' \cdot x : V_x$

Write $\xi' \equiv \xi \cdot \tilde{\xi}'$, $\xi_1 \equiv \xi \cdot \tilde{\xi}_1$, $\xi_2' \equiv \xi' \cdot \tilde{\xi}_2'$, $\xi_1'^{\mathbb{\Gamma}'_1} \equiv \xi \cdot \tilde{\xi}_1 \cdot \tilde{\xi}'$

$$
\begin{array}{lll}
\xi' \models \forall \delta. \forall x \in (\delta). A_1 & & \\
\leftrightarrow \ \forall\, \xi_2^{\prime \mathbb{\Gamma}'_2}. \xi' \preccurlyeq^\star \xi'_2 \rightarrow \forall\, M, V_x. M \overset{[\delta,\ \xi'_{2d}]}{\rightsquigarrow} V_x \rightarrow \xi'_{2dx} \models A_1 & & \text{Sem.} \forall \delta. \forall x \in (\delta). \\
\leftrightarrow \ \forall\, \xi_2^{\prime \mathbb{\Gamma}'_2}. \xi' \preccurlyeq^\star \xi'_2 \rightarrow \forall\, M, V_x. M \overset{[\delta,\ \xi'_{2d}]}{\rightsquigarrow} V_x \rightarrow \xi'_{2x} \models A_1 & & \text{Lem. 105 } A_1^{-\delta} \\
\leftrightarrow \ \forall\, \xi_1^{\mathbb{\Gamma}_1}. \xi \preccurlyeq^\star \xi_1 \rightarrow & & \text{Introduce } \xi_1 \\
\quad \forall\, \xi_2^{\prime \mathbb{\Gamma}'_2}. \xi' \preccurlyeq^\star \xi'_2 \rightarrow \forall\, M, V_x. M \overset{[\delta,\ \xi'_{2d}]}{\rightsquigarrow} V_x \rightarrow \xi'_{2x} \models A_1 & & \\
\rightarrow \ \forall\, \tilde{\xi}_1^{\tilde{\mathbb{\Gamma}}_1}.\ \xi \preccurlyeq^\star \xi_1 \wedge \xi' \preccurlyeq^\star \xi'_1 & & \text{Select } \tilde{\xi}'_2 \equiv \tilde{\xi}_1 \\
\quad \rightarrow \forall\, M, V_x. M \overset{[\delta,\ \xi'_{1d}]}{\rightsquigarrow} V_x \rightarrow \xi'_{1dx} \models A_1 & & \\
\rightarrow \ \forall\, \tilde{\xi}_1^{\tilde{\mathbb{\Gamma}}_1}.\ \xi \preccurlyeq^\star \xi_1 \wedge \xi' \preccurlyeq^\star \xi'_1 \wedge \mathring{\mathsf{a}}(\xi_1) \cap \mathring{\mathsf{a}}(\xi') \subseteq \mathring{\mathsf{a}}(\xi) & & \text{Subset of } \forall\, \xi'_1 \\
\quad \rightarrow \forall\, M, V_x. M \overset{[\delta,\ \xi'_{1d}]}{\rightsquigarrow} V_x \rightarrow \xi'_{1dx} \models A_1 & & \\
\rightarrow \ \forall\, \tilde{\xi}_1^{\tilde{\mathbb{\Gamma}}_1}.\ \xi \preccurlyeq^\star \xi_1 \wedge \xi_1 \preccurlyeq^\star \xi'_1 \wedge \mathring{\mathsf{a}}(\xi_1) \cap \mathring{\mathsf{a}}(\xi') \subseteq \mathring{\mathsf{a}}(\xi) & & \text{Lem. 104} \\
\quad \rightarrow \forall\, M, V_x. M \overset{[\delta,\ \xi'_{1d}]}{\rightsquigarrow} V_x \rightarrow \xi'_{1dx} \models A_1 & & \\
\rightarrow \ \forall\, \tilde{\xi}_1^{\tilde{\mathbb{\Gamma}}_1}.\ \xi \preccurlyeq^\star \xi_1 \wedge \xi_1 \preccurlyeq^\star \xi'_1 \wedge \mathring{\mathsf{a}}(\xi_1) \cap \mathring{\mathsf{a}}(\xi') \subseteq \mathring{\mathsf{a}}(\xi) & & \text{Subset of possible} \\
\quad \rightarrow \forall\, M, V_x.\ M \overset{[\delta,\ \xi_{1d}]}{\rightsquigarrow} V_x \wedge \mathring{\mathsf{a}}(V_x) \cap \mathring{\mathsf{a}}(\xi'_1) \subseteq \mathring{\mathsf{a}}(\xi_1) & & M\text{'s \& } V_x\text{'s} \\
\quad \rightarrow \xi'_{1dx} \models A_1 & & \\
\rightarrow \ \forall\, \tilde{\xi}_1^{\tilde{\mathbb{\Gamma}}_1}.\ \xi \preccurlyeq^\star \xi_1 \wedge \xi_1 \preccurlyeq^\star \xi'_1 \wedge \mathring{\mathsf{a}}(\xi_1) \cap \mathring{\mathsf{a}}(\xi') \subseteq \mathring{\mathsf{a}}(\xi) & & \text{Lem. 104,} \\
\quad \rightarrow \forall\, M, V_x.\ M \overset{[\delta,\ \xi_{1d}]}{\rightsquigarrow} V_x \wedge \mathring{\mathsf{a}}(V_x) \cap \mathring{\mathsf{a}}(\xi'_1) \subseteq \mathring{\mathsf{a}}(\xi_1) & & A_1 \text{ -ExtInd}_{Sem} \\
\quad \rightarrow \xi_{1dx} \models A_1 & & \text{Lem. 105, } A_1^{-\delta} \\
\rightarrow \ \forall\, \xi_1^{\mathbb{\Gamma}_1}.\ \xi \preccurlyeq^\star \xi_1 \rightarrow \forall\, M, V_x.\ M \overset{[\delta,\ \xi_{1d}]}{\rightsquigarrow} V_x \rightarrow \xi_{1dx} \models A_1 & & \text{Nominal determinacy} \\
& & \text{Def. 25} \\
\rightarrow \ \xi \models \forall \delta. \forall x \in (\delta). A_1 & & \text{Sem.} \forall \delta. \forall x \in (\delta).
\end{array}
$$

4. Two specific cases:

$\forall \delta. f \bullet () = b\{b \# \delta\}$ This holds given Lem. 121 as simply stating $x : \mathsf{Unit}$ ensures this holds. The full proof follows a similar reasoning as the proof for Lem. 121 but with the slight simplification without the $\forall x \in (\delta).$-complication.

$\forall \delta. \forall x \in (\delta). f \bullet x = b\{b \# \delta + x\}$ Proven in Lem. 121.

□

Lemma 123 (Syntactically thin formulae implies semantically thin)**.**

$$\forall\, \mathbb{\Gamma}, A, x.\ \mathbb{\Gamma} \backslash x \Vdash A \rightarrow (A\text{-Thin}_{Syn}(x) \rightarrow A\text{-Thin}_{Sem}(x))$$

Proof. This proof assumes that all models ξ and $\xi\backslash x$ are well-constructed models.

1. If $\mathbb{\Gamma}\backslash x \Vdash A$ and $\mathbb{\Gamma} \Vdash x : \alpha_{\text{-}(\mathsf{Nm},\to)}$ then A $\textsc{Thin}_{Sem}(x)$:
   Given $[\![e]\!]_\xi \equiv [\![e]\!]_{\xi\cdot x:V_x^{\alpha\text{-}(\mathsf{Nm},\to)}}$ and $[\![\mathbb{\Gamma}_1]\!]_\xi \equiv [\![\mathbb{\Gamma}_1]\!]_{\xi\cdot x:V_x^{\alpha\text{-}(\mathsf{Nm},\to)}}$ the proof holds easily.
   Lem. 81, Lem. 82, Lem. 83.

2. The assertions $A \equiv \mathsf{T}, \mathsf{F}, e = e',\ e \neq e'$ are all free from x ($x \notin \mathsf{fv}(A)$) then A $\textsc{Thin}_{Sem}(x)$:

– T, F: clearly hold.

– $e = e'$: hold as follows.

1	$\xi \models e = e'$	
2	$\leftrightarrow$ $[\![e]\!]_\xi \cong_\alpha^{\mathring{a}(\xi)} [\![e']\!]_\xi$	*Sem.*$=$
3	$\leftrightarrow$ $[\![e]\!]_{\xi\backslash x} \cong_\alpha^{\mathring{a}(\xi)} [\![e']\!]_{\xi\backslash x}$	$\mathbb{\Gamma}\backslash x \Vdash e : \alpha \to [\![e]\!]_\xi \equiv [\![e]\!]_{\xi\backslash x}$
4	$\leftrightarrow$ $[\![e]\!]_{\xi\backslash x} \cong_\alpha^{\mathring{a}(\xi\backslash x)} [\![e']\!]_{\xi\backslash x}$	*Lem. 32*
5	$\leftrightarrow$ $\xi\backslash x \models e = e'$	

– $e \neq e'$: Same proof as above as proof is $\leftrightarrow$.

3. If A_1 $\textsc{Thin}_{Syn}(x)$ and A_2 $\textsc{Thin}_{Syn}(x)$ then by I.H. A_1 $\textsc{Thin}_{Sem}(x)$ and A_2 $\textsc{Thin}_{Sem}(x)$
   Then it is necessary to prove that $A_1 \wedge A_2$, $A_1 \vee A_2$, $e \bullet e' = m\{A_1\}$, $\forall y \in (\mathbb{\Gamma}_1).A_1$, $\exists y^{\alpha\text{-}(\mathsf{Nm},\to)} \in (\mathbb{\Gamma}_1).A_1$, $\exists y \in (\mathbb{\Gamma}_1\backslash_{-TCV}).A_1$
   are all $\textsc{Thin}_{Sem}(x)$

– $A_1 \wedge A_2$, $A_1 \vee A_2$: The proofs are trivial by induction and hence are omitted.

– $e \bullet e' = m\{A_1\}$: holds by I.H. on A_1 as follows.

1	$\xi \models e \bullet e' = m\{A_1\}$	
2	$\leftrightarrow$ $\exists V_m.\ ee' \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_m$ $\wedge$ $\xi \cdot m : V_m \models A_1$	*I.H.*
3	$\to$ $\exists V_m.\ ee' \overset{[\mathbb{\Gamma},\ \xi\backslash x]}{\rightsquigarrow} V_m$	$[\![ee']\!]_\xi \equiv [\![ee']\!]_{\xi\backslash x}$, *Sem.* $\to$
	$\wedge$ $\xi\backslash x \cdot m : V_m \models A_1$	$(\mathring{a}(V_m) \cap \mathring{a}(V_x) \subseteq \mathring{a}(\xi\backslash x))$
4	$\to$ $\xi\backslash x \models e \bullet e' = m\{A_1\}$	

– $\forall y \in (\mathbb{\Gamma}_1).A_1$: holds assuming A_1 $\textsc{Thin}_{Syn}(x)$ as follows.

1	$\xi \models \forall y \in (\mathbb{\Gamma}_1).A_1$	
2	$\leftrightarrow\ \forall\, M_y, V_y. M_y \overset{[\mathbb{\Gamma}_1,\ \xi]}{\rightsquigarrow} V_y \rightarrow \xi \cdot y : V_y \models A_1$	$Sem.\forall y \in (\mathbb{\Gamma}_1).A_1$
3	$\rightarrow\ \forall\, M_y, V_y.\ \mathring{a}(M_y) = \emptyset \wedge [\![\mathbb{\Gamma}_1]\!]_\xi \vdash M_y : \alpha$ $\wedge\ (\mathring{a}(\xi), M_y\xi) \Downarrow (\mathring{a}(\xi), G',\ V_y)$ $\rightarrow\ \xi \cdot y : V_y \models A_1$	$Sem.\overset{[,\ ]}{\rightsquigarrow}$
4	$\rightarrow\ \forall\, M_y, V_y.\ \mathring{a}(M_y) = \emptyset \wedge [\![\mathbb{\Gamma}_1]\!]_{\xi\backslash x} \vdash M_y : \alpha$ $\wedge\ (\mathring{a}(\xi), M_y\xi) \Downarrow (\mathring{a}(\xi), G',\ V_y)$ $\rightarrow\ \xi \cdot y : V_y \models A_1$	$[\![\mathbb{\Gamma}_1]\!]_{\xi\backslash x} \subseteq [\![\mathbb{\Gamma}_1]\!]_\xi$
5	$\rightarrow\ \forall\, M_y, V_y.\ \mathring{a}(M_y) = \emptyset \wedge [\![\mathbb{\Gamma}_1]\!]_{\xi\backslash x} \vdash M_y : \alpha$ $\wedge\ (\mathring{a}(\xi\backslash x), M_y\xi\backslash x) \Downarrow (\mathring{a}(\xi\backslash x), G',\ V_y)$ $\rightarrow\ \xi \cdot y : V_y \models A_1$	$x \notin \mathsf{fv}([\![\mathbb{\Gamma}_1]\!]_{\xi\backslash x}),$ $Sem. \rightarrow$ $(\mathring{a}(V_y) \cap \mathring{a}(V_x) \subseteq \mathring{a}(\xi\backslash x))$
6	$\rightarrow\ \forall\, M_y, V_y. M_y \overset{[\mathbb{\Gamma}_1,\ \xi\backslash x]}{\rightsquigarrow} V_y \rightarrow (\xi \cdot y : V_y) \models A_1$	$Sem.\overset{[,\ ]}{\rightsquigarrow}$
7	$\rightarrow\ \forall\, M_y, V_y. M_y \overset{[\mathbb{\Gamma}_1,\ \xi\backslash x]}{\rightsquigarrow} V_y \rightarrow (\xi \cdot y : V_y)\backslash x \models A_1$	$I.H.$
8	$\rightarrow\ \forall\, M_y, V_y. M_y \overset{[\mathbb{\Gamma}_1,\ \xi\backslash x]}{\rightsquigarrow} V_y \rightarrow \xi\backslash x \cdot y : V_y \models A_1$	$Def.\xi\backslash x$
9	$\rightarrow\ \xi\backslash x \models \forall y \in (\mathbb{\Gamma}_1).A_1$	$Sem.\forall y \in (\mathbb{\Gamma}_1).A_1$

– $\exists y^{\alpha\text{-}(\mathsf{Nm},\to)} \in (\mathbb{\Gamma}_1).A_1$: holds as follows.

1	$\xi \models \exists y^{\alpha\text{-}(\mathsf{Nm},\to)} \in (\mathbb{\Gamma}_1).A_1$		
2	$\leftrightarrow$	$\exists M_y, V_y. M_y \overset{[\mathbb{\Gamma}_1,\ \xi]}{\rightsquigarrow} V_y \ \wedge\ \xi \cdot y : V_y \models A_1$	*Sem.*$\exists y \in (\mathbb{\Gamma}_1).A_1$
3	$\to$	$\exists M_y, V_y.\ \mathring{a}(M_y) = \emptyset \ \wedge\ [\![\mathbb{\Gamma}_1]\!]_\xi \vdash M_y : \alpha$ $\wedge\ (\mathring{a}(\xi), M_y\xi) \Downarrow (\mathring{a}(\xi), G',\ V_y)$ $\wedge\ \xi \cdot y : V_y \models A_1$	*Sem.*$\overset{[,\ ]}{\rightsquigarrow}$
4	$\to$	$\exists M_y, V_y.\ \mathring{a}(M_y) = \emptyset \ \wedge\ [\![\mathbb{\Gamma}_1]\!]_{\xi\backslash x} \vdash M_y : \alpha$ $\wedge\ (\mathring{a}(\xi), M_y\xi) \Downarrow (\mathring{a}(\xi), G',\ V_y)$ $\wedge\ \xi \cdot y : V_y \models A_1$	$V_y^{\alpha\text{-}(\mathsf{Nm},\to)}$ *equally derivable from any TC, Lem. 80*
5	$\to$	$\exists M_y, V_y.\ \mathring{a}(M_y) = \emptyset \ \wedge\ [\![\mathbb{\Gamma}_1]\!]_{\xi\backslash x} \vdash M_y : \alpha$ $\wedge\ (\mathring{a}(\xi\backslash x), M_y\xi\backslash x) \Downarrow (\mathring{a}(\xi\backslash x), G',\ V_y)$ $\wedge\ \xi \cdot y : V_y \models A_1$	$x \notin \mathsf{fv}([\![\mathbb{\Gamma}_1]\!]_{\xi\backslash x})$, *Sem.* $\to$ $(\mathring{a}(V_y) \cap \mathring{a}(V_x) \subseteq \mathring{a}(\xi\backslash x))$
6	$\to$	$\exists M_y, V_y. M_y \overset{[\mathbb{\Gamma}_1,\ \xi\backslash x]}{\rightsquigarrow} V_y \ \wedge\ \xi \cdot y : V_y \models A_1$	*Sem.*$\overset{[,\ ]}{\rightsquigarrow}$
7	$\to$	$\exists M_y, V_y. M_y \overset{[\mathbb{\Gamma}_1,\ \xi\backslash x]}{\rightsquigarrow} V_y \ \wedge\ (\xi \cdot y : V_y)\backslash x \models A_1$	*I.H.*
8	$\to$	$\exists M_y, V_y. M_y \overset{[\mathbb{\Gamma}_1,\ \xi\backslash x]}{\rightsquigarrow} V_y \ \wedge\ \xi\backslash x \cdot y : V_y \models A_1$	*Def.*$\xi\backslash x$
9	$\to$	$\xi\backslash x \models \exists y \in (\mathbb{\Gamma}_1).A_1$	*Sem.*$\exists y \in (\mathbb{\Gamma}_1).A_1$

– $\exists y \in (\mathbb{\Gamma}_1 \backslash_{-TCV}).A_1$: holds as follows.

1	$\xi \models \exists y \in (\mathbb{\Gamma}_1 \backslash_{-TCV}).A_1$	
2	$\leftrightarrow \exists M_y, V_y. M_y \overset{[\mathbb{\Gamma}_1 \backslash_{-TCV},\ \xi]}{\rightsquigarrow} V_y \wedge \xi \cdot y : V_y \models A_1$	$Sem.\exists y \in (\mathbb{\Gamma}_1 \backslash_{-TCV}).A_1$
3	$\rightarrow \exists M_y, V_y.\ \mathring{a}(M_y) = \emptyset \wedge [\![\mathbb{\Gamma}_1 \backslash_{-TCV}]\!]_\xi \vdash M_y : \alpha$ $\wedge (\mathring{a}(\xi), M_y\xi) \Downarrow (\mathring{a}(\xi), G',\ V_y)$ $\wedge \xi \cdot y : V_y \models A_1$	$Sem.\overset{[,]}{\rightsquigarrow}$
4	$\rightarrow \exists M_y, V_y.\ \mathring{a}(M_y) = \emptyset \wedge [\![\mathbb{\Gamma}_1 \backslash_{-TCV}]\!]_{\xi\backslash x} \vdash M_y : \alpha$ $\wedge (\mathring{a}(\xi), M_y\xi) \Downarrow (\mathring{a}(\xi), G',\ V_y)$ $\wedge \xi \cdot y : V_y \models A_1$	$[\![\mathbb{\Gamma}_0]\!]_{\xi\backslash x} \equiv [\![\mathbb{\Gamma}_0]\!]_\xi$
5	$\rightarrow \exists M_y, V_y.\ \mathring{a}(M_y) = \emptyset \wedge [\![\mathbb{\Gamma}_1 \backslash_{-TCV}]\!]_{\xi\backslash x} \vdash M_y : \alpha$ $\wedge (\mathring{a}(\xi\backslash x), M_y\xi\backslash x) \Downarrow (\mathring{a}(\xi\backslash x), G',\ V_y)$ $\wedge \xi \cdot y : V_y \models A_1$	$x \notin \mathsf{fv}([\![\mathbb{\Gamma}_1]\!]_{\xi\backslash x})$, $Sem. \rightarrow$ $(\mathring{a}(V_y) \cap \mathring{a}(V_x) \subseteq \mathring{a}(\xi\backslash x))$
6	$\rightarrow \exists M_y, V_y. M_y \overset{[\mathbb{\Gamma}_1 \backslash_{-TCV},\ \xi\backslash x]}{\rightsquigarrow} V_y \wedge (\xi \cdot y : V_y)\backslash x \models A_1$	$I.H.$
7	$\rightarrow \exists M_y, V_y. M_y \overset{[\mathbb{\Gamma}_1 \backslash_{-TCV},\ \xi\backslash x]}{\rightsquigarrow} V_y \wedge \xi\backslash x \cdot y : V_y \models A_1$	$Def.\xi\backslash x$
8	$\rightarrow \exists M_y, V_y. M_y \overset{[\mathbb{\Gamma}_1 \backslash_{-TCV},\ \xi\backslash x]}{\rightsquigarrow} V_y \wedge (\xi \cdot y : V_y)\backslash x \models A_1$	$Sem.\overset{[,]}{\rightsquigarrow}$
9	$\rightarrow \xi\backslash x \models \exists y \in (\mathbb{\Gamma}_1 \backslash_{-TCV}).A_1$	$Sem.\exists y \in (\mathbb{\Gamma}_1 \backslash_{-TCV}).A_1$

The formula $\exists y \in (\mathbb{\Gamma}_1).A_1$ is not $\textsc{Thin}_{Syn}(x)$ for all cases of $\mathbb{\Gamma}_0$ as $\mathbb{\Gamma}_0$ may contain a TCV δ which could be used in M_y hence be explicitly dependant on x. This means the two previous cases state two clear cases for $\mathbb{\Gamma}$ (or the type of y) where formulae of this form are guaranteed to be $\textsc{Thin}_{Syn}(x)$.

4: If A_1 $\textsc{Thin}_{Syn}(x)$ and $\delta \notin \mathsf{ftcv}(A_1)$ then $\forall\delta.A_1$ $\textsc{Thin}_{Sem}(x)$:

Holds by I.H. on A_1 as follows

1	$\xi \models \forall\delta.A_1$	
2	$\leftrightarrow\ \forall\, \xi'^{\mathbb{\Gamma}'}.\ \xi \preccurlyeq^\star \xi' \ \rightarrow\ \xi' \cdot \delta : \mathbb{\Gamma}' \models A_1$	*Sem.*$\forall\delta$.
3	$\leftrightarrow\ \forall\, \xi'^{\mathbb{\Gamma}'}.\ \xi \preccurlyeq^\star \xi' \ \rightarrow\ \xi' \models A_1^{-\delta}$	*Lem. 105*
4	$\leftrightarrow\ \forall\, \xi'^{\mathbb{\Gamma}'}.\ \xi \preccurlyeq^\star \xi' \ \rightarrow\ \xi'\backslash x \models A_1^{-\delta}$	*I.H.*
5	$\rightarrow\ \forall\, \xi'\backslash x^{\mathbb{\Gamma}'\backslash x}.\ \xi\backslash x \preccurlyeq^\star \xi'\backslash x \ \rightarrow\ \xi'\backslash x \models A_1^{-\delta}$	*Subset of* $\forall\, \xi'^{\mathbb{\Gamma}'}$ *as* $[\![\mathbb{\Gamma}\backslash x]\!]_{\xi\backslash x} \subseteq [\![\mathbb{\Gamma}]\!]_\xi$ *and* $\xi\backslash x \subseteq \xi$
6	$\rightarrow\ \forall\, \xi''^{\mathbb{\Gamma}''}.\ \xi\backslash x \preccurlyeq^\star \xi'' \ \rightarrow\ \xi'' \models A_1^{-\delta}$	*Rename* $\xi'' \equiv \xi'\backslash x$
7	$\rightarrow\ \forall\, \xi''^{\mathbb{\Gamma}''}.\ \xi\backslash x \preccurlyeq^\star \xi'' \ \rightarrow\ \xi'' \cdot \delta : \mathbb{\Gamma}'' \models A_1^{-\delta}$	*Lem. 105*
8	$\rightarrow\ \xi\backslash x \models \forall\delta.A_1^{-\delta}$	*Sem.*$\forall\delta$.

5: If A_1 $\textsc{Thin}_{Syn}(x)$ then $\forall\delta.\forall y^{\alpha_y} \in (\delta).A_1^{-\delta}$ $\textsc{Thin}_{Sem}(x)$: Holds by I.H. on A_1 as follows.

1	Assume A_1 $\textsc{Thin}_{Syn}(x)$	
2	i.e. assume $\forall\, \mathbb{\Gamma}, \delta, y^{\alpha_y}.\ \delta{+}y : \alpha_y \backslash x \Vdash A_1 \rightarrow \forall\, \xi_{iy}^{\delta+y:\alpha_y}.\ \xi_{iy} \models A_1 \rightarrow \xi_{iy}\backslash x \models A_1$	
3	Assume $\mathbb{\Gamma}$, $\xi^{\mathbb{\Gamma}}$ s.t. $\mathbb{\Gamma}\backslash x \Vdash \forall\delta.\forall y \in (\delta).A_1$ and $\xi \models \forall\delta.\forall y \in (\delta).A_1$	
4	$\rightarrow$ $\mathbb{\Gamma}\backslash x{+}\delta{+}y : \alpha_y \Vdash A_1$ $\wedge\ \forall\, \xi_1^{\mathbb{\Gamma}_1}.\xi \preccurlyeq^\star \xi_1 \rightarrow \forall\, M_y^{\alpha_y}, V_y.\ M_y \overset{[\delta,\ \xi_1\cdot\delta:\mathbb{\Gamma}_1]}{\rightsquigarrow} V_y$ $\rightarrow\ \xi_1 \cdot \delta : \mathbb{\Gamma}_1 \cdot y : V_y \models A_1$	*Typing rules,* *Sem.*$\forall$*.,* $\forall \in ()$*.*
5	$\rightarrow \forall\, \xi_1^{\mathbb{\Gamma}_1}.\xi \preccurlyeq^\star \xi_1 \rightarrow \forall\, M_y^{\alpha_y}, V_y.\ M_y \overset{[\delta,\ \xi_1\cdot\delta:\mathbb{\Gamma}_1]}{\rightsquigarrow} V_y$ $\rightarrow\ \xi_1 \cdot y : V_y \models A_1$	$A_1^{-\delta}$*, Lem. 105*
6	$\rightarrow \forall\, \xi_1^{\mathbb{\Gamma}_1}.\xi \preccurlyeq^\star \xi_1 \rightarrow \forall\, M_y^{\alpha_y}, V_y.\ M_y \overset{[\delta,\ \xi_1\cdot\delta:\mathbb{\Gamma}_1]}{\rightsquigarrow} V_y$ $\rightarrow\ (\xi_1 \cdot y : V_y)\backslash x \models A_1$	*I.H., Line.2*
7	$\rightarrow \forall\, \xi_1^{\mathbb{\Gamma}_1}.\xi \preccurlyeq^\star \xi_1 \rightarrow \forall\, M_y^{\alpha_y}, V_y.\ M_y \overset{[\delta,\ \xi_1\cdot\delta:\mathbb{\Gamma}_1]}{\rightsquigarrow} V_y$ $\rightarrow\ (\xi_1 \cdot \delta : \mathbb{\Gamma}_1 \cdot y : V_y)\backslash x \models A_1$	$A_1^{-\delta}$*, Lem. 105*
8	$\rightarrow \forall\, \xi_1^{\mathbb{\Gamma}_1}.\xi \preccurlyeq^\star \xi_1 \rightarrow \forall\, M_y^{\alpha_y}, V_y.\ \xi_{1d} \equiv \xi_1 \cdot \delta : \mathbb{\Gamma}_1$ $M_y \overset{[\delta,\ \xi_1 d\backslash x]}{\rightsquigarrow} V_y$ $\rightarrow\ (\xi_d \cdot y : V_y)\backslash x \models A_1$	$[\![\delta]\!]_{\xi_{1d}\backslash x} \subseteq [\![\delta]\!]_{\xi_{1d}}$ $x \notin \mathsf{fv}(M) \rightarrow M\xi \equiv M(\xi\backslash x)$ *Sem.* $\rightarrow$
9	$\rightarrow \forall\, \xi_1^{\mathbb{\Gamma}_1}.\xi \preccurlyeq^\star \xi_1 \rightarrow (\xi_1 \cdot \delta : \mathbb{\Gamma}_1)\backslash x \models \forall y \in (\delta).A_1$	*Sem.*$\forall \in ()$*.*
10	$\rightarrow \forall\, \xi_2^{\mathbb{\Gamma}_2}.\ \xi \preccurlyeq^\star \xi_2 \cdot x : V_x \rightarrow (\xi_2 \cdot x : V_x \cdot \delta : \mathbb{\Gamma}_1)\backslash x \models \forall y \in (\delta).A_1$	$\xi_1 \equiv \xi_2 \cdot x : V_x$
11	$\rightarrow \forall\, \xi_3^{\mathbb{\Gamma}_3}.\ \xi\backslash x \preccurlyeq^\star \xi_3 \rightarrow \xi_3 \cdot \delta : \mathbb{\Gamma}_3 \models \forall y \in (\delta).A_1$	*Subset of* $\forall\, \xi_2$ *extending* $\xi\backslash x$
12	$\rightarrow\ \xi\backslash x \models \forall\delta.\forall y \in (\delta).A_1$	*Sem.*$\forall\delta$*.*
13	Hence: $\forall\, \mathbb{\Gamma}.\ \mathbb{\Gamma}\backslash x \Vdash \forall\delta.\forall y \in (\delta).A_1$ $\rightarrow\ \forall\, \xi^{\mathbb{\Gamma}}.\ \xi \models \forall\delta.\forall y \in (\delta).A_1 \rightarrow \xi\backslash x \models \forall\delta.\forall y \in (\delta).A_1$	*Lines.3-12*
14	Hence: $\forall\delta.\forall y \in (\delta).A_1$ $\textsc{Thin}_{Sem}(x)$	

Hence all cases from Def. 55 are satisfied and hence the lemma holds. □

Again it is emphasised that $y\#\mathbb{\Gamma}_1$ *is* $\textsc{Thin}_{Syn}(x)$ *given* $y\#\mathbb{\Gamma}_1 \overset{\text{def}}{=} \forall z^{\mathsf{Nm}} \in (\mathbb{\Gamma}_1).z \neq y$ *and this is* $\textsc{Thin}_{Syn}(x)$ *hence also* $\textsc{Thin}_{Sem}(x)$.

A.2 Soundness of the Extra Core Rules

Here are the proofs for the core derivation rules of the ν-logic not included in Sec. 6.3.

A.2.1 Soundness of $[\textsc{Pair}]_\nu$

$$\frac{\{A\}\ M :_m \{B\} \quad \{B\}\ N :_n \{C[\langle m, n\rangle/u]_{\mathbb{\Gamma}+m+n}\} \quad C\ \textsc{Thin}_{Syn}(x)}{\{A\}\ \langle M, N\rangle :_u \{C\}}\ [\textsc{Pair}]_\nu$$

Proof. Clearly Lem. 113 ensures A-$\textsc{Thin}_{Syn}(x)$ implies A-$\textsc{Thin}_{Sem}(x)$.

1	Let: $\xi_m^{\mathbb{\Gamma}+m} \equiv \xi \cdot m : V_m$ and $\xi_{mn}^{\mathbb{\Gamma}+m+n} \equiv \xi_m \cdot n : V_n$	
2	Assume $\mathbb{\Gamma}$ s.t. $\mathbb{\Gamma} \Vdash \{A\}\ \langle M, N\rangle :_u \{C[\langle m,n\rangle/u]_{\mathbb{\Gamma}+m+n}\}$	
3	$\to$ $\mathbb{\Gamma} \Vdash \{A\}\ M :_m \{B\}$ $\wedge$ $\mathbb{\Gamma}+m \Vdash \{B\}\ N :_n \{C[\langle m,n\rangle/u]_{\mathbb{\Gamma}+m+n}\}$ $\wedge$ $\mathbb{\Gamma} \vdash N : \alpha$	
4	$\forall\, \mathbb{\Gamma}', \xi'^{\mathbb{\Gamma}_0}. \mathbb{\Gamma}' \triangleright \xi'$ $\to$ $\xi' \models A \to M \overset{[\mathbb{\Gamma}',\ \xi']}{\rightsquigarrow} V_m$ $\wedge$ $\xi' \cdot m : V_m \models B$	*I.H.(1)*
5	$\forall\, \mathbb{\Gamma}', \xi_m'^{\mathbb{\Gamma}_0+m}. \mathbb{\Gamma}'+m \triangleright \xi_m'$ $\to$ $\xi_m' \models B \to N \overset{[\mathbb{\Gamma}'+m,\ \xi_m']}{\rightsquigarrow} V_n$ $\wedge$ $\xi_m' \cdot n : V_n \models C[\langle m,n\rangle/u]_{\mathbb{\Gamma}'+m+n}$	*I.H.(2)*
6	Assume $\xi^{\mathbb{\Gamma}_0}$ s.t. $\mathbb{\Gamma} \triangleright \xi$ $\wedge$ $\xi \models A$	
7	$\to$ $M \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_m$ $\wedge$ $\xi \cdot m : V_m \models B$	*I.H.(1)*
8	$\to$ $M \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_m$ $\wedge$ $N \overset{[\mathbb{\Gamma}+m,\ \xi_m]}{\rightsquigarrow} V_n$ $\wedge$ $\xi_{mn} \models C[\langle m,n\rangle/u]_{\mathbb{\Gamma}_{mn}}$	*I.H.(2)*
9	$\to$ $M \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_m$ $\wedge$ $N \overset{[\mathbb{\Gamma}+m,\ \xi_m]}{\rightsquigarrow} V_n$ $\wedge$ $\exists\, V_u.\ \langle m,n\rangle \overset{[\mathbb{\Gamma}+m+n,\ \xi_{mn}]}{\rightsquigarrow} V_u$ $\wedge$ $\xi_{mn} \cdot u : V_u \models C$	*Sem.*$[e/x]$, $u \notin \mathsf{dom}(\xi_{mn})$
10	$\to$ $M \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_m$ $\wedge$ $N \overset{[\mathbb{\Gamma}+m,\ \xi_m]}{\rightsquigarrow} V_n$ $\wedge$ $\langle M, N\rangle \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_u$ $\wedge$ $\xi_{mn} \cdot u : V_u \models C$	*Op. Sem.(Pair)*
11	$\to$ $M \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_m$ $\wedge$ $N \overset{[\mathbb{\Gamma}+m,\ \xi_m]}{\rightsquigarrow} V_n$ $\wedge$ $\langle M, N\rangle \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_u$ $\wedge$ $\xi \cdot u : V_u \models C$	C $\textsc{Thin}_{Sem}$ (m,n)
12	$\to$ $\langle M, N\rangle \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_u$ $\wedge$ $\xi \cdot u : V_u \models C$	*Remove Excess*
13	$\to$ $\forall\, \xi^{\mathbb{\Gamma}_0}.\ \mathbb{\Gamma} \triangleright \xi$ $\to$ $\xi \models A \to \langle M, N\rangle \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_u$ $\wedge$ $\xi \cdot u : V_u \models C$	*Assumption, Line.5*
14	Hence: $\dfrac{\models \{A\}\ M :_m \{B\} \qquad \models \{B\}\ N :_n \{C[\langle m,n\rangle/u]_{\mathbb{\Gamma}_{mn}}\}}{\models \{A\}\ \langle M, N\rangle :_u \{C\}}$ C-$\textsc{Thin}_{Syn}(m,n)$	*Lines.1-12*

□

A.2.2 Soundness of $[\textsc{Proj}(i)]_\nu$

$$\frac{\{A\}\ M :_m \{C[\pi_i(m)/u]_{\mathbb{\Gamma}+m}\} \quad C\ \textsc{Thin}_{Syn}(m)}{\{A\}\ \pi_i(M) :_u \{C\}}\ [\textsc{Proj}(i)]_\nu$$

Proof. Clearly Lem. 113 ensures A-$\textsc{Thin}_{Syn}(x)$ implies A-$\textsc{Thin}_{Sem}(x)$.

1	Assume $\mathbb{\Gamma}$ s.t. $\mathbb{\Gamma} \Vdash \{A\}\ \pi_i(M) :_u \{C\}$	
2	$\to\ \mathbb{\Gamma} \Vdash \{A\}\ M :_m \{C[\pi_i(m)/u]_{\mathbb{\Gamma}+m}\}$	*Typing rules*
3	$\forall\, \xi^{\mathbb{\Gamma}_0}.\mathbb{\Gamma} \triangleright \xi\ \to\ \xi \models A \to M \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_m\ \wedge\ \xi \cdot m : V_m \models C[\pi_i(m)/u]_{\mathbb{\Gamma}+m}$	*I.H.(1)*
4	Assume $\xi^{\mathbb{\Gamma}_0}$ s.t. $\mathbb{\Gamma} \triangleright \xi\ \wedge\ \xi \models A$	
5	$\to M \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_m\ \wedge\ \xi \cdot m : V_m \models C[\pi_i(m)/u]_{\mathbb{\Gamma}+m}$	*I.H.(1)*
6	$\to\ M \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_m\ \wedge\ \pi_i(m) \overset{[\mathbb{\Gamma}+m,\ \xi\cdot m:V_m]}{\rightsquigarrow} V_u$ $\wedge\ \xi \cdot m : V_m \cdot u : V_u \models C$	*Sem.*$[e/x]$, $u \notin \mathsf{dom}(\xi \cdot m \cdot n)$
7	$\to\ M \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_m\ \wedge\ \pi_i(m) \overset{[\mathbb{\Gamma}+m,\ \xi\cdot m:V_m]}{\rightsquigarrow} V_u$ $\wedge\ \xi \cdot u : V_u \models C$	$C\ \ \textsc{Thin}_{Sem}\ (m)$
8	$\to\ \pi_i(M) \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_u\ \wedge\ \xi \cdot u : V_u \models C$	*Op. Sem.(Pair)*
9	$\to\ \forall\, \xi^{\mathbb{\Gamma}_0}.\ \mathbb{\Gamma} \triangleright \xi\ \to\ \xi \models A\ \to\ \pi_i(M) \overset{[\mathbb{\Gamma},\ \xi]}{\rightsquigarrow} V_u$ $\wedge\ \xi \cdot u : V_u \models C$	*Assumption, Line.4*
10	Hence: $\dfrac{\models \{A\}\ M :_m \{C[\pi_i(m)/u]_{\mathbb{\Gamma}+m}\} \quad C\ \textsc{Thin}_{Syn}(m)}{\models \{A\}\ \pi_i(M) :_u \{C\}}$	*Lines.1-10*

□

A.2.3 Soundness of $[\textsc{If}]_\nu$

$$\frac{\{A\}\ M :_m \{B\} \quad \{B[\mathsf{true}/m]\}\ N_1 :_u \{C\} \quad \{B[\mathsf{false}/m]\}\ N_2 :_u \{C\}}{\{A\}\ \mathsf{if}\ M\ \mathsf{then}\ N_1\ \mathsf{else}\ N_2 :_u \{C\}}\ [\textsc{If}]_\nu$$

Proof. The proof is standard, given that substitution is equivalent to standard substitution for b_i-values of type Bool. It holds trivially through the operational semantics of $\mathsf{if}\ M\ \mathsf{then}\ N_1\ \mathsf{else}\ N_2$ and the semantics of substitution in the logic. □

A.2.4 Soundness of $[\textsc{Neg}]_\nu$

$$\frac{\{A\}\ M :_m \{C[\neg m/u]\}}{\{A\}\ \neg M :_u \{C\}}\ [\textsc{Neg}]_\nu$$

Proof. The proof is standard, given that substitution is equivalent to substituting the negation of u for u. It holds trivially through the operational semantics of $\neg M$ and the semantics of substitution in the logic. □