Chiron: A Set Theory with Types, Undefinedness, Quotation, and Evaluation*

William M. Farmer†
McMaster University

28 December 2012

Abstract

Chiron is a derivative of von-Neumann-Bernays-Gödel (NBG) set theory that is intended to be a practical, general-purpose logic for mechanizing mathematics. Unlike traditional set theories such as Zermelo-Fraenkel (ZF) and NBG, Chiron is equipped with a type system, lambda notation, and definite and indefinite description. The type system includes a universal type, dependent types, dependent function types, subtypes, and possibly empty types. Unlike traditional logics such as first-order logic and simple type theory, Chiron admits undefined terms that result, for example, from a function applied to an argument outside its domain or from an improper definite or indefinite description. The most noteworthy part of Chiron is its facility for reasoning about the syntax of expressions. *Quotation* is used to refer to a set called a construction that represents the syntactic structure of an expression, and *evaluation* is used to refer to the value of the expression that a construction represents. Using quotation and evaluation, syntactic side conditions, schemas, syntactic transformations used in deduction and computation rules, and other such things can be directly expressed in Chiron. This paper presents the syntax and semantics of Chiron, some definitions and simple examples illustrating its use, a proof system for Chiron, and a notion of an interpretation of one theory of Chiron in another.

*Published as SQRL Report No. 38, McMaster University, 2007 (revised 2012). This research was supported by NSERC.

†Address: Department of Computing and Software, McMaster University, 1280 Main Street West, Hamilton, Ontario L8S 4K1, Canada. E-mail: wmfarmer@mcmaster.ca.

Contents

List of Tables

1 Introduction

The usefulness of a logic is often measured by its expressivity: the more that can be expressed in the logic, the more useful the logic is. By a *logic*, we mean a language (or a family of languages) that has a formal syntax and a precise semantics with a notion of logical consequence. (A logic may also have, but is not required to have, a proof system.) By this definition, a theory in a logic—such as Zermelo-Fraenkel (ZF) set theory in first-order order—is itself a logic. But what do we mean by *expressivity*? There are actually two notions of expressivity. The *theoretical expressivity* of a logic is the measure of what ideas can be expressed in the logic without regard to how the ideas are expressed. The *practical expressivity* of a logic is the measure of how readily ideas can be expressed in the logic.

To illustrate the difference between these two notions, let us compare two logics, standard first-order logic (FOL) and first-order logic without function symbols (FOL$^-$). Since functions can be represented using either predicate symbols or function symbols, FOL and FOL$^-$ clearly have exactly the same theoretical expressivity. For example, if three functions are represented as unary function symbols f, g, h in FOL, these functions can be represented as binary predicate symbols p_f, p_g, p_h in FOL$^-$. The statement that the third function is the composition of the first two functions is expressed in FOL by the formula

$$\forall\, x \,.\, h(x) = f(g(x)),$$

while it is expressed in FOL$^-$ by the more verbose formula

$$\forall\, x, z \,.\, p_h(x, z) \equiv \exists\, y \,.\, p_g(x, y) \wedge p_f(y, z).$$

The verbosity that comes from using predicate symbols to represent functions progressively increases as the complexity of statements about functions increases. Hence, FOL$^-$ has a significantly lower level of practical expressivity than FOL does.

Traditional general-purpose logics—such as predicate logics like first-order logic and simple type theory and set theories like ZF and von-Neumann-Bernays-Gödel (NBG) set theory—are primarily intended to be theoretical tools. They are designed to be used *in theory*, not *in practice*. They are thus very expressive theoretically, but not very expressive practically. For example, in the languages of ZF and NBG, there is no vocabulary for forming a term $f(a)$ that denotes the application of a set f representing a function to a set a representing an argument to f. Moreover, even if such an application operator were added to ZF or NBG, there is no special mechanism for handling

"undefined" applications. As a result, statements involving functions and undefinedness are much more verbose and indirect than they need to be, and reasoning about functions and undefinedness is usually performed in the metalogic instead of in the logic itself.

Chiron is a set theory that has a much higher level of practical expressivity than traditional set theories. It is intended to be a general-purpose logic that, unlike traditional logics, is designed to be used in practice. It integrates NBG set theory, elements of type theory, a scheme for handling undefinedness, and a facility for reasoning about the syntax of expressions. This paper presents the syntax and semantics of Chiron, some definitions and simple examples illustrating its use, a proof system for Chiron, and a notion of an interpretation of one theory of Chiron in another. A quicker, more informal presentation of the syntax and semantics of Chiron is found in [8].

The following is the outline of the paper. Section 2 gives an informal overview of Chiron. Section 3 presents Chiron's official syntax and an unofficial compact notation for Chiron. The semantics of Chiron is given in section 4. We also show in section 4 that there is a faithful semantic interpretation of NBG in Chiron. A large group of useful operators are defined in sections 5 and 6 including the operators needed for the substitution of a term for the occurrences of a free variable. Some of the practical expressivity of Chiron is illustrated by examples in section 7. Section 8 presents a proof system for Chiron and proves that it is sound and also complete in a restricted sense. Section 9 defines the notion of a semantic interpretation of one theory of Chiron in another. The paper concludes in section 10 with a brief summary and a list of future tasks. There are two appendices. The first presents two alternate semantics for Chiron based on value gaps, and the second gives an expanded definition of a proper expression.

2 Overview

This section gives an informal overview of Chiron. A formal definition of the syntax and semantics of Chiron is presented in subsequent sections.

2.1 NBG Set Theory

NBG set theory is closely related to the more well-known ZF set theory. The underlying logic of both NBG and ZF is first-order logic, and NBG and ZF both share the same intuitive model of the iterated hierarchy of sets. However, in contrast to ZF, variables in NBG range over both sets and proper

classes. Thus, the universe of sets V and total functions from V to V like the cardinality function can be represented as terms in NBG even though they are proper classes. There is a faithful interpretation of ZF in NBG [19, 22, 23]. This means that reasoning in ZF can be reduced to reasoning in NBG in a meaning-preserving way and that NBG is consistent iff ZF is consistent. A good introduction to NBG is found in [13] or [17].

Chiron is a derivative of NBG. It is an enhanced version of STMM [5], a version of NBG with types and undefinedness. Chiron has a much richer syntax and more complex semantics than NBG, but the models for Chiron contain exactly the same values (i.e., classes) as the models for NBG. Moreover, there is a faithful semantic interpretation of NBG in Chiron—which means that there is a meaning-preserving embedding of NBG in Chiron such that Chiron is a conservative extension of the image of NBG under the embedding. That is, Chiron adds new vocabulary and assumptions to NBG without compromising the underlying semantics of NBG, and hence, Chiron is satisfiable iff NBG is satisfiable.

2.2 Values

A *value* is a set, class, superclass, truth value, undefined value, or operation. A *class* is an element of a model of NBG set theory. Each class is a collection of classes. A *set* is a class that is a member of a class. A class is thus a collection of sets. A class is *proper* if it not a set. Intuitively, sets are "small" classes and proper classes are "big" classes. A *superclass* is a collection of classes that need not be a class itself. Summarizing, the domain D_{v} of sets is a proper subdomain of the domain D_{c} of classes, and D_{c} is a proper subdomain of the domain D_{s} of superclasses. D_{v} is the universal class (the class of all sets), and D_{c} is the universal superclass (the superclass of all classes).

There are two truth values, T representing *true* and F representing *false*. The truth values are not members of D_{s}. There is also an *undefined value* $\bot$ which serves as the value of various sorts of undefined terms such as undefined function applications and improper definite or indefinite descriptions. $\bot$ is not a member of $D_s \cup \{\mathrm{T}, \mathrm{F}\}$.

An *operation* is a mapping over superclasses, the truth values, and the undefined value. More precisely, for $n \geq 0$, an *n-ary operation* is a total mapping

$$\sigma : D_1 \times \cdots \times D_n \to D_{n+1}$$

where D_i is D_{s}, $D_{\mathrm{c}} \cup \{\bot\}$, or $\{\mathrm{T}, \mathrm{F}\}$ for all i with $1 \leq i \leq n+1$. An operation

is not a member of $D_s \cup \{\mathrm{T}, \mathrm{F}, \bot\}$. A *function* is a class of ordered pairs that represents a (possibly partial) mapping

$$f : D_{\mathrm{v}} \rightharpoonup D_{\mathrm{v}}.$$

Operations are not classes, but many operations can be represented by functions (which are classes).

2.3 Expressions

An *expression* is a tree whose leaves are *symbols*. There are four special sorts of expressions: *operators*, *types*, *terms*, and *formulas*. An expression is *proper* is it is one of these special sorts of expressions, and an expression is *improper* if it is not proper. Proper expressions denote values, while improper expressions are nondenoting (i.e., they do not denote anything).

Operators denote operations. Many sorts of syntactic entities can be formalized in Chiron as operators. Examples include logical connectives; individual constants, function symbols, and predicate symbols from first-order logic; type constants and type constructors including dependent type constructors; and definedness operators. Like a function or predicate symbol in first-order logic, an operator in Chiron is not useful unless it is applied.

Types are used to restrict the values of operators and variables and to classify terms by their values. They denote superclasses. Terms are used to describe classes. They denote classes or the undefined value $\bot$. A term is *defined* if it denotes a class and is *undefined* if it denotes $\bot$. Every term is assigned a type. Suppose a term a is assigned a type α and α denotes a superclass Σ_α. If a is defined, i.e., a denotes a class x, then x is in Σ_α. Formulas are used to make assertions. They denote truth values.

The proper expressions are categorized according to their first (leftmost) symbols:

1. Operator and operator applications (op, op-app).
2. Variables (var).
3. Type applications and dependent function types (type-app, dep-fun-type).
4. Function applications and abstractions (fun-app, fun-abs).
5. Conditional terms (if).
6. Existential quantifications (exists).

7. Definite and indefinite descriptions (def-des, indef-des).

8. Quotations and evaluations (quote, eval).

2.4 Dependent Function Types

A *dependent function type* is a type of the form

$$\gamma = (\mathsf{dep\text{-}fun\text{-}type}, (\mathsf{var}, x, \alpha), \beta)$$

where α and β are types. (Dependent function types are commonly known as *dependent product types*.) The type γ denotes a superclass of possibly partial functions. A function abstraction of the form

$$(\mathsf{fun\text{-}abs}, (\mathsf{var}, x, \alpha), b),$$

where b is a term of type β, is of type γ.

The dependent function type γ is a generalization of the more common function type $\alpha \to \beta$. If f is a term of type $\alpha \to \beta$ and a is a term of type α, then the application $f(a)$ is of type β—which does not depend on the value of a. In Chiron, however, if f is a term of type γ and a is a term of type α, then the term

$$(\mathsf{fun\text{-}app}, f, a),$$

the application of f to a, is of the type

$$(\mathsf{type\text{-}app}, \gamma, a),$$

the type formed by applying the type γ to a—which generally depends on the value of a.

2.5 Undefinedness

An expression is *undefined* if it has no prescribed meaning or if it denotes a value that does not exist. There are several sources of undefined expressions in Chiron:

- Nondenoting operator, type, and function applications.
- Nonexistent function abstractions.
- Improper definite and indefinite descriptions.
- Out of range variables and evaluations.

Undefined expressions are handled in Chiron according to the *traditional approach to undefinedness* [6]. The value of an undefined term is the undefined value $\perp$, but the value of an undefined type or formula is D_{c} (the universal superclass) or F, respectively. That is, the values for undefined types, terms, and formulas are D_{c}, $\perp$, and F, respectively. Commonly used in mathematical practice, the traditional approach to undefinedness enables statements involving partial functions and definite and indefinite descriptions to be expressed very concisely [6].

2.6 Quotation and Evaluation

A *construction* is a set that represents the syntactic structure of an expression. A term of the form (quote, e), where e is an expression, denotes the construction that represents e. Thus a proper expression e has two different meanings:

1. The *semantic meaning* of e is the value denoted by e itself.
2. The *syntactic meaning* of e is the construction denoted by (quote, e).

There are two ways to refer to a semantic meaning v. The first is to directly form a proper expression e not beginning with eval that denotes v. The second is to form a term a that denotes the construction that represents a proper expression e that denotes v and then form the type $(\mathsf{eval}, a, \mathsf{type})$, term $(\mathsf{eval}, a, \alpha)$, or formula $(\mathsf{eval}, a, \mathsf{formula})$ (depending on whether e is a type, a term assigned the type α, or a formula) which denotes v.

Likewise there are two ways to refer to a syntactic meaning c. The first is to directly form a term a not beginning with quote that denotes c. The second is to form an expression e such that the construction c represents the syntactic structure of e and then form the expression (quote, e) which denotes c.

For an expression e, the term (quote, e) denotes the syntactic meaning of e and is thus always defined (even when e is an undefined term or an improper (nondenoting) expression). However, a term $(\mathsf{eval}, a, \alpha)$, where α is a type, may be undefined.

op	type	term	formula
op-app	var	type-app	dep-fun-type
fun-app	fun-abs	if	exists
def-des	indef-des	quote	eval

Table 1: The Key Words of Chiron.

3 Syntax

This section presents the syntax of Chiron which is inspired by the S-expression syntax of the Lisp family of programming languages.

3.1 Expressions

Let $\mathcal{S}$ be a fixed countably infinite set of symbols and $\mathcal{K}$ be the set of the 16 symbols given in Table 1. Assume $\mathcal{K} \subseteq \mathcal{S}$. The members of $\mathcal{S}$ are the *symbols* of Chiron and the members of $\mathcal{K}$ are the *key words* of Chiron. The key words are used to classify expressions and identify different categories of expressions.

Let a *signature form* be a sequence $s_1, \ldots, s_{n+1}$ of symbols where $n \geq 0$ and each s_i is the symbol type, term, or formula. A *language* of Chiron is a pair $L = (\mathcal{O}, \theta)$ where:

1. $\mathcal{O}$ is a countable set of symbols such that (1) $\mathcal{O}$ and $\mathcal{S}$ are disjoint and (2) $\mathcal{O}_0 \subseteq \mathcal{O}$ where $\mathcal{O}_0$ is the set of the 18 symbols given in Table 2. The members of $\mathcal{O}$ are called *operator names* and the members of $\mathcal{O}_0$ are the *built-in operator names* of Chiron.

2. θ maps each $o \in \mathcal{O}$ to a signature form such that, for each $o \in \mathcal{O}_0$, $\theta(o)$ is the signature form assigned to o in Table 2.

Throughout this paper let $L = (\mathcal{O}, \theta)$ be a language of Chiron.

Let $L_i = (\mathcal{O}_i, \theta_i)$ be a language of Chiron for $i = 1, 2$. L_1 is a *sublanguage* of L_2 (and L_2 is an *extension* of L_1), written $L_1 \leq L_2$, if $\mathcal{O}_1 \subseteq \mathcal{O}_2$ and θ_1 is a subfunction of θ_2.

The two formation rules below inductively define the notion of an *expression* of L. $\mathbf{expr}_L[e]$ asserts that e is an expression of L.

Expr-1 (Atomic expression)

$$\frac{s \in \mathcal{S} \cup \mathcal{O}}{\mathbf{expr}_L[s]}$$

	Operator Name	Signature Form
1.	set	type
2.	class	type
3.	op-names	term
4.	lang	type
5.	expr-sym	type
6.	expr-op-name	term, type
7.	expr	term, type
8.	expr-op	term, type
9.	expr-type	term, type
10.	expr-term	term, type
11.	expr-term-type	term, term, type
12.	expr-formula	term, type
13.	in	term, term, formula
14.	type-equal	type, type, formula
15.	term-equal	term, term, type, formula
16.	formula-equal	formula, formula, formula
17.	not	formula, formula
18.	or	formula, formula, formula

Table 2: The Built-In Operator Names of Chiron.

Expr-2 (Compound expression)

$$\frac{\mathbf{expr}_L[e_1], \ldots, \mathbf{expr}_L[e_n]}{\mathbf{expr}_L[(e_1, \ldots, e_n)]}$$

where $n \geq 0$.

Hence, an expression is an S-expression (with commas in place of spaces) that exhibits the structure of a tree whose leaves are symbols in $\mathcal{S} \cup \mathcal{O}$. Let $\mathcal{E}_L$ denote the set of expressions of L.

A *proper expression* of L is an expression of L defined by the set of 13 formation rules below. A proper expression denotes a class, a truth value, the undefined value, or an operation. Each proper expression of L is assigned an expression. $\mathbf{p\text{-}expr}_L[e : e']$ asserts that $e \in \mathcal{E}_L$ is a proper expression of L to which the expression $e' \in \mathcal{E}_L$ is assigned. An *improper expression* of L is an expression of L that is not a proper expression of L. Improper expressions are nondenoting.

There are four sorts of proper expressions. An *operator* of L is a proper expression of L to which the expression op is assigned. A *type* of L is a proper expression of L to which the expression type is assigned. A *term* of L is a proper expression of L to which a type of L is assigned. And a *formula* of L is a proper expression of L to which the expression $\mathsf{formula}$ is assigned. When a is a term of L, α is a type of L, and $\mathbf{p\text{-}expr}_L[a : \alpha]$ holds, a is said to be a *term of type* α. As we mentioned earlier, operators denote operations, types denote superclasses, terms denote classes or the undefined value $\bot$, and formulas denote the truth values T and F.

$\mathbf{operator}_L[O]$ means $\mathbf{p\text{-}expr}_L[O : \mathsf{op}]$, $\mathbf{type}_L[\alpha]$ means $\mathbf{p\text{-}expr}_L[\alpha : \mathsf{type}]$, $\mathbf{term}_L[a]$ means $\mathbf{p\text{-}expr}_L[a : \alpha]$ for some type α of L, and $\mathbf{formula}_L[A]$ means $\mathbf{p\text{-}expr}_L[A : \mathsf{formula}]$. $\mathbf{term}_L[a : \alpha]$ means $\mathbf{p\text{-}expr}_L[a : \alpha]$ and $\mathbf{type}_L[\alpha]$, i.e., a is a term of type α. An expression k is a *kind* of L, written $\mathbf{kind}_L[k]$, if $k = \mathsf{type}$, $\mathbf{type}_L[k]$, or $k = \mathsf{formula}$. Thus kinds are the expressions assigned to types, terms, and formulas. A proper expression e of L is said to be an *expression of kind* k if $k = \mathsf{type}$ and e is a type, $\mathbf{type}_L[k]$ and e is a term of type k, or $k = \mathsf{formula}$ and e is a formula.

The following formation rules define the 13 proper expression categories of Chiron:

P-Expr-1 (Operator)

$$\frac{o \in \mathcal{O}, \mathbf{kind}_L[k_1], \ldots, \mathbf{kind}_L[k_{n+1}]}{\mathbf{operator}_L[(\mathsf{op}, o, k_1, \ldots, k_{n+1})]}$$

where $n \geq 0$; $\theta(o) = s_1, \ldots, s_{n+1}$; and $k_i = s_i = \mathsf{type}$, $\mathbf{type}_L[k_i]$ and $s_i = \mathsf{term}$, or $k_i = s_i = \mathsf{formula}$ for all i with $1 \leq i \leq n+1$.

P-Expr-2 (Operator application)

$$\frac{\mathbf{operator}_L[(\mathsf{op}, o, k_1, \ldots, k_{n+1})], \mathbf{expr}_L[e_1], \ldots, \mathbf{expr}_L[e_n]}{\mathbf{p\text{-}expr}_L[(\mathsf{op\text{-}app}, (\mathsf{op}, o, k_1, \ldots, k_{n+1}), e_1, \ldots, e_n) : k_{n+1}]}$$

where $n \geq 0$ and ($k_i = \mathsf{type}$ and $\mathbf{type}_L[e_i]$), ($\mathbf{type}_L[k_i]$ and $\mathbf{term}_L[e_i]$), or ($k_i = \mathsf{formula}$ and $\mathbf{formula}_L[e_i]$) for all i with $1 \leq i \leq n$.

P-Expr-3 (Variable)

$$\frac{x \in \mathcal{S}, \mathbf{type}_L[\alpha]}{\mathbf{term}_L[(\mathsf{var}, x, \alpha) : \alpha]}$$

P-Expr-4 (Type application)

$$\frac{\mathbf{type}_L[\alpha], \mathbf{term}_L[a]}{\mathbf{type}_L[(\mathsf{type\text{-}app}, \alpha, a)]}$$

P-Expr-5 (Dependent function type)

$$\frac{\mathbf{term}_L[(\mathsf{var}, x, \alpha)], \mathbf{type}_L[\beta]}{\mathbf{type}_L[(\mathsf{dep\text{-}fun\text{-}type}, (\mathsf{var}, x, \alpha), \beta)]}$$

P-Expr-6 (Function application)

$$\frac{\mathbf{term}_L[f : \alpha], \mathbf{term}_L[a]}{\mathbf{term}_L[(\mathsf{fun\text{-}app}, f, a) : (\mathsf{type\text{-}app}, \alpha, a)]}$$

P-Expr-7 (Function abstraction)

$$\frac{\mathbf{term}_L[(\mathsf{var}, x, \alpha)], \mathbf{term}_L[b : \beta]}{\mathbf{term}_L[(\mathsf{fun\text{-}abs}, (\mathsf{var}, x, \alpha), b) : (\mathsf{dep\text{-}fun\text{-}type}, (\mathsf{var}, x, \alpha), \beta)]}$$

P-Expr-8 (Conditional term)

$$\frac{\mathbf{formula}_L[A], \mathbf{term}_L[b : \beta], \mathbf{term}_L[c : \gamma]}{\mathbf{term}_L[(\mathsf{if}, A, b, c) : \delta]}$$

where $\delta = \begin{cases} \beta & \text{if } \beta = \gamma \\ (\mathsf{op\text{-}app}, (\mathsf{op}, \mathsf{class}, \mathsf{type})) & \text{otherwise} \end{cases}$

P-Expr-9 (Existential quantification)

$$\frac{\mathbf{term}_L[(\mathsf{var}, x, \alpha)], \mathbf{formula}_L[B]}{\mathbf{formula}_L[(\mathsf{exists}, (\mathsf{var}, x, \alpha), B)]}$$

P-Expr-10 (Definite description)

$$\frac{\mathbf{term}_L[(\mathsf{var}, x, \alpha)], \mathbf{formula}_L[B]}{\mathbf{term}_L[(\mathsf{def\text{-}des}, (\mathsf{var}, x, \alpha), B) : \alpha]}$$

P-Expr-11 (Indefinite description)

$$\frac{\mathbf{term}_L[(\mathsf{var}, x, \alpha)], \mathbf{formula}_L[B]}{\mathbf{term}_L[(\mathsf{indef\text{-}des}, (\mathsf{var}, x, \alpha), B) : \alpha]}$$

P-Expr-12 (Quotation)

$$\frac{\mathbf{expr}_L[e]}{\mathbf{term}_L[(\mathsf{quote}, e) : \mathsf{E}]}$$

where $\mathsf{E} = (\mathsf{op\text{-}app}, (\mathsf{op}, \mathsf{expr}, \mathsf{L}, \mathsf{type}), \ell)$ and L and ℓ are defined as in Table 3.

P-Expr-13 (Evaluation)

$$\frac{\mathbf{term}_L[a], \mathbf{kind}_L[k]}{\mathbf{p\text{-}expr}_L[(\mathsf{eval}, a, k) : k]}$$

Note: An expanded definition of a proper expression with 25 proper expression categories is given in appendix B.

Proposition 3.1.1 *The formation rules assign a unique expression of L to each proper expression of L.*

Unless stated otherwise, an *operator name*, *operator*, etc. is an *operator name*, *operator*, etc. of L. We will use $s, t, u, v, w, x, y, z, \ldots$ to denote symbols; $o, o', \ldots$ to denote operator names; $O, O', \ldots$ to denote operators; $\alpha, \beta, \gamma, \ldots$ to denote types; $a, b, c, \ldots$ to denote terms; $A, B, C, \ldots$ to denote formulas; and $k, k', \ldots$ to denote kinds.

The *length* of an expression $e \in \mathcal{E}_L$, written $|e|$, is defined recursively by the following statements:

1. If $s \in \mathcal{S} \cup \mathcal{O}$, $|s| = 1$.
2. If $e_1, \ldots, e_n \in \mathcal{E}_L$, $|(e_1, \ldots, e_n)| = 1 + |e_1| + \cdots |e_n|$.

Notice that $|(\,)| = 1$ and, in general, $|e|$ equals the number of symbols and parenthesis pairs occurring in e. The *complexity* of an expression $e \in \mathcal{E}_L$, written $c(e)$, is the pair $(m, n) \in \mathbf{N} \times \mathbf{N}$ of natural numbers such that:

1. m is the number of occurrences of the symbol eval in e that are not within a quotation.
2. n is the length of e.

For $c(e_1) = (m_1, n_1)$ and $c(e_2) = (m_2, n_2)$, $c(e_1) < c(e_2)$ means either $m_1 < m_2$ or ($m_1 = m_2$ and $n_1 < n_2$).

Let $O = (\mathsf{op}, o, k_1, \ldots, k_{n+1})$ be an operator. The operator name o is called the *name* of O, and the list $k_1, \ldots, k_{n+1}$ of kinds is called the

signature of O. O is said to be an *n-ary* operator because it is applied to n arguments in an operator application. O is a *type operator*, *term operator*, or *formula operator* if $k_{n+1} = \mathsf{type}$, $\mathbf{type}_L[k_{n+1}]$, or $k_{n+1} = \mathsf{formula}$, respectively. A *type constant* is a 0-ary type operator application of the form $(\mathsf{op\text{-}app}, (\mathsf{op}, o, \mathsf{type}))$. A *term constant* (or simply *constant*) of type α is a 0-ary term operator application of the form $(\mathsf{op\text{-}app}, (\mathsf{op}, o, \alpha))$. A *formula constant* is a 0-ary formula operator application of the form $(\mathsf{op\text{-}app}, (\mathsf{op}, o, \mathsf{formula}))$. Two operators are *similar* if their names are the same.

Let $o \in \mathcal{O}$ and $F = \theta(o)$. If F contains the symbol term, there will be many operators with the name o that have different signatures, i.e., there will be many operators with the name o that are similar to each other. However, each operator name will usually be assigned a *preferred signature*. If F does not contain the symbol term, o is assigned F as its preferred signature. Each built-in operator name is assigned a preferred signature. An operator formed from a built-in operator name and its preferred signature is called a *built-in operator* of Chiron. The operators in Table 3 are the built-in operators of Chiron.

Remark 3.1.2 A language $L = (\mathcal{O}, \theta)$ can be presented as a set L' of operators such that, for each symbol o, there is at most one operator in L' whose name is o. $\mathcal{O}$ is the set of operator names o such that $o \in \mathcal{O}$ iff o is the name of some operator in L', and θ is the function from $\mathcal{O}$ to signature forms such that, for all $(o :: k_1, \ldots, k_{n+1}) \in L'$, $\theta(o)$ is the signature form corresponding to $k_1, \ldots, k_{n+1}$. In addition, for each $(o :: k_1, \ldots, k_{n+1}) \in L'$, $k_1, \ldots, k_{n+1}$ is the preferred signature of o.

Let $a = (\mathsf{var}, x, \alpha)$ be a variable. x is called the *name* of a, and α is called the *type* of a. Two variables are similar if their names are the same.

An expression e is *eval-free* if all occurrences of the symbol eval in e are within a quotation. Notice that, if e is eval-free, then the complexity of e is $c(e) = (0, |e|)$. We will use $\alpha^{\mathrm{ef}}, \beta^{\mathrm{ef}}, \gamma^{\mathrm{ef}}, \ldots$; $a^{\mathrm{ef}}, b^{\mathrm{ef}}, c^{\mathrm{ef}}, \ldots$; and $A^{\mathrm{ef}}, B^{\mathrm{ef}}, C^{\mathrm{ef}}, \ldots$ to denote eval-free types, terms, and formulas, respectively.

A *subexpression* of an expression is defined inductively as follows:

1. If e is a proper expression, then e is a subexpression of itself.
2. If $e = (s, e_1, \ldots, e_n)$ is a proper expression such that s is not quote, then e_i is a subexpression of e for each proper expression e_i with $1 \leq i \leq n$.
3. If e is a subexpression of e' and e' is a subexpression of e'', then e is a subexpression of e''.

	Operator
1.	(op, set, type)
2.	(op, class, type)
3.	(op, op-names, V)
4.	(op, lang, type)
5.	(op, expr-sym, type)
6.	(op, expr-op-name, L, type)
7.	(op, expr, L, type)
8.	(op, expr-op, L, type)
9.	(op, expr-type, L, type)
10.	(op, expr-term, L, type)
11.	(op, expr-term-type, L, E_{ty}, type)
12.	(op, expr-formula, L, type)
13.	(op, in, V, C, formula)
14.	(op, type-equal, type, type, formula)
15.	(op, term-equal, C, C, type, formula)
16.	(op, formula-equal, formula, formula, formula)
17.	(op, not, formula, formula)
18.	(op, or, formula, formula, formula)
where:	
	V = (op-app, (op, set, type))
	C = (op-app, (op, class, type))
	ℓ = (op-app, (op, opnames, V))
	L = (op-app, (op, lang, type))
	E_{ty} = (op-app, (op, expr-type, L, type), ℓ)

Table 3: The Built-In Operators of Chiron.

e is a *proper subexpression* of e' if e is a subexpression of e' and $e \neq e'$. Notice that (1) a subexpression is always a proper expression, (2) an improper expression has no subexpressions, (3) a quotation has no proper subexpressions, and (4) if e_1 is a proper subexpression of e_2, then $|e_1| < |e_2|$ and $c(e_1) < c(e_2)$, i.e., the length and the complexity of a proper subexpression of an expression is strictly less than length and the complexity of the expression itself.

Compact Notation	**Official Notation**
$(o :: k_1, \ldots, k_{n+1})$	$(\mathsf{op}, o, k_1, \ldots, k_{n+1})$
$(o :: k_1, \ldots, k_{n+1})(e_1, \ldots, e_n)$	$(\mathsf{op\text{-}app}, (\mathsf{op}, o, k_1, \ldots, k_{n+1}), e_1, \ldots, e_n)$
$(x : \alpha)$	$(\mathsf{var}, x, \alpha)$
$\alpha(a)$	$(\mathsf{type\text{-}app}, \alpha, a)$
$(\Lambda\, x : \alpha\ .\ \beta)$	$(\mathsf{dep\text{-}fun\text{-}type}, (\mathsf{var}, x, \alpha), \beta)$
$f(a)$	$(\mathsf{fun\text{-}app}, f, a)$
$(\lambda\, x : \alpha\ .\ b)$	$(\mathsf{fun\text{-}abs}, (\mathsf{var}, x, \alpha), b)$
$\mathsf{if}(A, b, c)$	(if, A, b, c)
$(\exists\, x : \alpha\ .\ B)$	$(\mathsf{exists}, (\mathsf{var}, x, \alpha), B)$
$(\iota\, x : \alpha\ .\ B)$	$(\mathsf{def\text{-}des}, (\mathsf{var}, x, \alpha), B)$
$(\epsilon\, x : \alpha\ .\ B)$	$(\mathsf{indef\text{-}des}, (\mathsf{var}, x, \alpha), B)$
$\ulcorner e \urcorner$	(quote, e)
$[\![a]\!]_k$	(eval, a, k)
$[\![a]\!]_{\mathrm{ty}}$	$(\mathsf{eval}, a, \mathsf{type})$
$[\![a]\!]_{\mathrm{te}}$	$(\mathsf{eval}, a, (\mathsf{op\text{-}app}, (\mathsf{op}, \mathsf{class}, \mathsf{type})))$
$[\![a]\!]_{\mathrm{fo}}$	$(\mathsf{eval}, a, \mathsf{formula})$

Table 4: Compact Notation

3.2 Compact Notation

We introduce in this subsection a compact notation for proper expressions—which we will use in the rest of the paper whenever it is convenient. The first group of notational definitions in Table 4 defines the compact notation for each of the 13 proper expression categories.

The next group of notational definitions in Table 5 defines additional compact notation for the built-in operators and the universal quantifier.

We will often employ the following abbreviation rules when using the compact notation:

1. A matching pair of parentheses in an expression may be dropped if there is no resulting ambiguity.

2. A variable $(x : \alpha)$ occurring in the body e of $(\star\, x : \alpha\ .\ e)$, where $\star$ is Λ, λ, $\exists$, $\forall$, ι, or ϵ may be written as x if there is no resulting ambiguity.

3. $(\star\, x_1 : \alpha_1 \ldots (\star\, x_n : \alpha_n\ .\ e) \cdots)$, where $\star$ is Λ, λ, $\exists$, or $\forall$, may be written as

$$(\star\, x_1 : \alpha_1, \ldots, x_n : \alpha_n\ .\ e).$$

Compact Notation	**Defining Expression**
V	$(\mathsf{set} :: \mathsf{type})(\,)$
C	$(\mathsf{class} :: \mathsf{type})(\,)$
ℓ	$(\mathsf{op\text{-}names} :: \mathsf{term})(\,)$
L	$(\mathsf{lang} :: \mathsf{type})(\,)$
E_{sy}	$(\mathsf{expr\text{-}sym} :: \mathsf{type})(\,)$
$\mathsf{E}_{\mathrm{on},a}$	$(\mathsf{expr\text{-}op\text{-}name} :: \mathsf{L}, \mathsf{type})(a)$
E_{on}	$(\mathsf{expr\text{-}op\text{-}name} :: \mathsf{L}, \mathsf{type})(\ell)$
E_a	$(\mathsf{expr} :: \mathsf{L}, \mathsf{type})(a)$
E	$(\mathsf{expr} :: \mathsf{L}, \mathsf{type})(\ell)$
$\mathsf{E}_{\mathrm{op},a}$	$(\mathsf{expr\text{-}op} :: \mathsf{L}, \mathsf{type})(a)$
E_{op}	$(\mathsf{expr\text{-}op} :: \mathsf{L}, \mathsf{type})(\ell)$
$\mathsf{E}_{\mathrm{ty},a}$	$(\mathsf{expr\text{-}type} :: \mathsf{L}, \mathsf{type})(a)$
E_{ty}	$(\mathsf{expr\text{-}type} :: \mathsf{L}, \mathsf{type})(\ell)$
$\mathsf{E}_{\mathrm{te},a}$	$(\mathsf{expr\text{-}term} :: \mathsf{L}, \mathsf{type})(a)$
E_{te}	$(\mathsf{expr\text{-}term} :: \mathsf{L}, \mathsf{type})(\ell)$
$\mathsf{E}^b_{\mathrm{te},a}$	$(\mathsf{expr\text{-}term\text{-}type} :: \mathsf{L}, \mathsf{E}_{\mathrm{ty}}, \mathsf{type})(a, b)$
$\mathsf{E}^b_{\mathrm{te}}$	$(\mathsf{expr\text{-}term\text{-}type} :: \mathsf{L}, \mathsf{E}_{\mathrm{ty}}, \mathsf{type})(\ell, b)$
$\mathsf{E}_{\mathrm{fo},a}$	$(\mathsf{expr\text{-}formula} :: \mathsf{L}, \mathsf{type})(a)$
E_{fo}	$(\mathsf{expr\text{-}formula} :: \mathsf{L}, \mathsf{type})(\ell)$
$(a \in b)$	$(\mathsf{in} :: \mathsf{V}, \mathsf{C}, \mathsf{formula})(a, b)$
$(\alpha =_{\mathrm{ty}} \beta)$	$(\mathsf{type\text{-}equal} :: \mathsf{type}, \mathsf{type}, \mathsf{formula})(\alpha, \beta)$
$(a =_{\alpha} b)$	$(\mathsf{term\text{-}equal} :: \mathsf{C}, \mathsf{C}, \mathsf{type}, \mathsf{formula})(a, b, \alpha)$
$(a = b)$	$(a =_{\mathsf{C}} b)$
$(A \equiv B)$	$(\mathsf{formula\text{-}equal} :: \mathsf{formula}, \mathsf{formula}, \mathsf{formula})(A, B)$
$(\neg A)$	$(\mathsf{not} :: \mathsf{formula}, \mathsf{formula})(A)$
$(a \notin b)$	$(\neg(a \in b))$
$(a \neq b)$	$(\neg(a = b))$
$(A \vee B)$	$(\mathsf{or} :: \mathsf{formula}, \mathsf{formula}, \mathsf{formula})(A, B)$
$(\forall\, x : \alpha \,.\, A)$	$(\neg(\exists\, x : \alpha \,.\, (\neg A)))$

Table 5: Additional Compact Notation

Similarly, $(\star\, x_1 : \alpha \ldots (\star\, x_n : \alpha\ .\ e) \cdots)$, where $\star$ is Λ, λ, $\exists$, or $\forall$, may be written as

$$(\star\, x_1, \ldots, x_n : \alpha\ .\ e).$$

4. If we assign a fixed type, say α, to a symbol x to be used as a variable name, then an expression of the form $(\star\, x : \alpha\ .\ e)$, where $\star$ is Λ, λ, $\exists$, $\forall$, ι, or ϵ, may be written as $(\star\, x\ .\ e)$.

5. If $k_1, \ldots, k_{n+1}$ is the preferred signature assigned to an operator name o, then an operator application of the form

   $$(o :: k_1, \ldots, k_{n+1})(e_1, \ldots, e_n)$$

   may be written as $o(e_1, \ldots, e_n)$ and an operator application of the form $(o :: k)(\,)$ may be written as o.

6. $[\![a]\!]_{\mathrm{ty}}$, $[\![a]\!]_{\mathrm{te}}$, $[\![a]\!]_{\alpha}$, and $[\![a]\!]_{\mathrm{fo}}$ may be shortened to $[\![a]\!]$ if a is of type E_{ty}, E_{te}, $\mathsf{E}_{\mathrm{te}}^{\ulcorner\alpha\urcorner}$, and E_{fo}, respectively.

Using the compact notation, expressions can be written in Chiron so that they look very much like expressions written in mathematics textbooks and papers.

3.3 Quasiquotation

Quasiquotation is a parameterized form of quotation in which the parameters serve as holes in a quotation that are filled with the values of expressions. It is a very powerful syntactic device for specifying expressions and defining macros. Quasiquotation was introduced by W. Quine in 1940 in the first version of his book *Mathematical Logic* [21]. It has been extensively employed in the Lisp family of programming languages [1].[1]

We will introduce quasiquotation into Chiron as a notational definition. Unlike quotation, quasiquotation will not be part of the official Chiron syntax. The meaning of a quasiquotation will be an expression that denotes a construction.

The three formation rules below inductively define the notion of a *marked expression* of L. $\mathbf{m\text{-}expr}_L[m]$ asserts that m is a marked expression of L.

[1] In Lisp, the standard symbol for quasiquotation is the backquote (‘) symbol, and thus in Lisp, quasiquotation is usually called *backquote*.

M-Expr-1

$$\frac{s \in \mathcal{S} \cup \mathcal{O}}{\mathbf{m\text{-}expr}_L[s]}$$

M-Expr-2

$$\frac{\mathbf{term}_L[a]}{\mathbf{m\text{-}expr}_L[\lfloor a \rfloor]}$$

M-Expr-3

$$\frac{\mathbf{m\text{-}expr}_L[m_1], \ldots, \mathbf{m\text{-}expr}_L[m_n]}{\mathbf{m\text{-}expr}_L[(m_1, \ldots, m_n)]}$$

where $n \geq 0$.

A marked expression of the form $\lfloor a \rfloor$ is called an *evaluated component*.

Proposition 3.3.1 *Every expression is a marked expression of L.*

A *quasiquotation* of L is a syntactic entity of the form

$(\mathsf{quasiquote}, m)$

where m is a marked expression of L.[2,3] A compact notation for quasiquotation can be easily defined as indicated by the following example: Let $\ulcorner f(\lfloor a \rfloor) \urcorner$ be the compact notation for the quasiquotation

$(\mathsf{quasiquote}, (\mathsf{fun\text{-}app}, f, \lfloor a \rfloor))$.

Here we are using $\ulcorner m \urcorner$ when m is an expression to mean the quotation of m and when m is a marked expression containing evaluated components to mean the quasiquotation of m.

We next define the semantics of quasiquotation. It assumes a knowledge of the semantics of Chiron given in section 4 and the defined-in and ord-pair operators defined in section 5. Let F be the function, mapping marked expressions of L to terms of L, recursively defined by:

1. If m is a symbol $s \in \mathcal{S} \cup \mathcal{O}$, then $F(m) = (\mathsf{quote}, s)$.

[2]The evaluated components in a quasiquotation are sometimes called *antiquotations*.

[3]Quasiquotations correspond to backquotes in Lisp as follows. The symbol quasiquote corresponds to the backquote symbol (`) and an evaluated component $\lfloor a \rfloor$ in a quasiquotation corresponds to $,a$. Thus the quasiquotation $(\mathsf{quasiquote}, (a, b, \lfloor c \rfloor))$ corresponds to the backquote `` `(a b ,c) ``.

2. If m is an evaluated component $\lfloor a \rfloor$, then $F(m) = a$.

3. If m is a marked expression $(m_1, \ldots, m_n)$ where $n \geq 0$, then

$$F(m) = [F(m_1), \ldots, F(m_n)].$$

Note that, as defined in subsection 5.2, $[a_1, \ldots, a_n]$ denotes an n-tuple of sets. For a quasiquotation $q = (\mathsf{quasiquote}, m)$, define $G(q) = F(m)$.

Proposition 3.3.2

1. *If e is an expression, then $G((\mathsf{quasiquote}, e))$ is a term of L such that*

$$\models G((\mathsf{quasiquote}, e)) = (\mathsf{quote}, e).$$

2. *If q is a quasiquotation containing an evaluated component $\lfloor (\mathsf{quote}, e) \rfloor$ and q' is the quasiquotation that results from replacing $\lfloor (\mathsf{quote}, e) \rfloor$ in q with e, then*

$$\models G(q) = G(q').$$

3. *If $\{\lfloor a_1 \rfloor, \ldots, \lfloor a_n \rfloor\}$ is the set of evaluated components occurring in a quasiquotation q, then*

$$\models (a_1 \downarrow \mathsf{E} \wedge \cdots \wedge a_n \downarrow \mathsf{E}) \supset G(q) \downarrow \mathsf{E}.$$

Note that, as defined in subsection 5.1, $a \downarrow \alpha$ asserts that the value of the term a is in the value of type α.

From now on, we will consider a quasiquotation q to be an alternate notation for the term $G(q)$. Many expressions involving syntax can be expressed very succinctly using quasiquotation. See sections 6 and 7 for examples, especially the liar paradox example in subsection 7.7.

4 Semantics

This section presents the official semantics of Chiron which is based on *standard models*. Two alternate semantics based on other kinds of models are given in appendix A.

4.1 The Liar Paradox

Using quotation and evaluation, it is possible to express the *liar paradox* in Chiron. That is, it is possible to construct a term LIAR whose value equals the value of

$$\ulcorner\neg[\![\mathsf{LIAR}]\!]_{\mathrm{fo}}\urcorner.$$

(See subsection 7.7 for details.) LIAR denotes a construction representing a formula that says in effect "I am a formula that is false".

If the naive semantics is employed for quotation and evaluation, a contradiction is immediately obtained:

$$\begin{aligned}[\![\mathsf{LIAR}]\!]_{\mathrm{fo}} &= [\![\ulcorner\neg[\![\mathsf{LIAR}]\!]_{\mathrm{fo}}\urcorner]\!]_{\mathrm{fo}} \\ &= \neg[\![\mathsf{LIAR}]\!]_{\mathrm{fo}}\end{aligned}$$

$[\![\mathsf{LIAR}]\!]_{\mathrm{fo}}$ is thus *ungrounded* in the sense that its value depends on itself. This simple argument is essentially the proof of A. Tarski's 1933 theorem on the undefinability of truth [25, 26, 27, Theorem I]. The theorem says that $[\![x]\!]_{\mathrm{fo}}$ cannot serve as a truth predicate over all formulas.

Any reasonable semantics for Chiron needs a way to block the liar paradox and similar ungrounded expressions. We will briefly describe three approaches.

The first approach is to remove evaluation (eval) from Chiron but keep quotation (quote). This would eliminate ungrounded expressions. However, it would also severely limit Chiron's facility for reasoning about the syntax of expressions. It would be possible using quotation to construct terms that denote constructions, but without evaluation it would not be possible to employ these terms as the expressions represented by the constructions. Some of the power of evaluation could be replaced by introducing functions that map types of constructions to types of values. An example would be a function that maps numerals to natural numbers.

A second approach is to define a semantics with "value gaps" so that expressions like $[\![\mathsf{LIAR}]\!]_{\mathrm{fo}}$ are not assigned any value at all. In his famous paper *Outline of a Theory of Truth* [15], S. Kripke presents a framework for defining semantics with *truth-value gaps* using various evaluation schemes. Kripke's approach can be easily generalized to allow *value gaps* for types and terms as well as for formulas. In appendix A we define two value-gap semantics for Chiron using Kripke's framework with valuation schemes based on the weak Kleene logic [14] and B. van Fraassen's notion of a *supervaluation* [28]. Kripke-style value-gap semantics are very interesting, if not enlightening, but they are exceeding difficult to work with in practice. The

main problem is that there is no mechanism to assert that an expression has no value (see appendix A for details).

The third approach is to consider any evaluation of a term that denotes a construction representing a non-eval-free expression to be undefined. For example, the value of $[\![\mathsf{LIAR}]\!]_{\mathrm{fo}}$ would be F, the undefined value for formulas. The origin of this idea is found in Tarski's famous paper on the concept of truth [25, 26, 27, Theorem III]. It is important to understand that an evaluation of a term a containing the symbol eval can be defined as long as a denotes a construction representing an expression that does not contain eval outside of a quotation. Thus the value of an evaluation like $[\![[\![\ulcorner\ulcorner 17\urcorner\urcorner]\!]_{\mathrm{te}}]\!]_{\mathrm{te}}$ would be the value of 17 because the expression 17 presumably does not contain eval at all. The official semantics for Chiron defined in this section employs this third approach to blocking the liar paradox.

4.2 Prestructures

A *prestructure* of Chiron is a pair $(D, \in)$, where D is a nonempty domain and $\in$ is a membership relation on D, that satisfies the axioms of NBG set theory as given, for example, in [13] or [17].

Let $P = (D_{\mathrm{c}}^P, \in^P)$ be a prestructure of Chiron. A *class* of P is a member of D_{c}^P. A *set* of P is a member x of D_{c}^P such that $x \in^P y$ for some member y of D_{c}^P. That is, a set is a class that is itself a member of a class. A class is *proper* if it is not a set. A *superclass* of P is a collection of classes in D_{c}^P. We consider a class, as a collection of sets, to be a superclass. Let D_{v}^P be the domain of sets of P and D_{s}^P be the domain of superclasses of P. The following inclusions hold: $D_{\mathrm{v}}^P \subset D_{\mathrm{c}}^P \subset D_{\mathrm{s}}^P$.

A *function* of P is a member f of D_{c}^P such that:

1. Each $p \in^P f$ is an ordered pair $\langle x, y\rangle$ where x, y are in D_{v}^P.
2. For all $\langle x_1, y_1\rangle, \langle x_2, y_2\rangle \in^P f$, if $x_1 = x_2$, then $y_1 = y_2$.

Notice that a function of P may be partial, i.e., there may not be an ordered pair $\langle x, y\rangle$ in a function for each x in D_{v}^P.

Let $D_{\mathrm{f}}^P \subset D_{\mathrm{c}}^P$ be the domain of functions of P. For f, x in D_{c}^P, $f(x)$ denotes the unique y in D_{v}^P such that f is in D_{f}^P and $\langle x, y\rangle \in^P f$. ($f(x)$ is undefined if there is no such unique y in D_{v}^P.) For Σ in D_{s}^P and x in D_{c}^P, $\Sigma[x]$ denotes the unique class in D_{c}^P composed of all y in D_{v}^P such that, for some f in both Σ and D_{f}^P, $f(x) = y$. ($\Sigma[x]$ is undefined if there is no such unique class in D_{c}^P.)

Let T^P, F^P, and $\bot^P$ be distinct values not in D_{s}^P. T^P and F^P represent the truth values *true* and *false*, respectively. $\bot^P$ is the *undefined value* that represents values that are undefined.

For $n \geq 0$, an *n-ary operation* of P is a total mapping from $D_1 \times \cdots \times D_n$ to D_{n+1} where D_i is D_{s}^P, $D_{\mathrm{c}}^P \cup \{\bot^P\}$, or $\{\mathrm{T}^P, \mathrm{F}^P\}$ for each i with $1 \leq i \leq n+1$. Let D_{o}^P be the domain of operations of P. We assume that $D_{\mathrm{s}}^P \cup \{\mathrm{T}^P, \mathrm{F}^P, \bot^P\}$ and D_{o}^P are disjoint.

Choose $D_{\mathrm{e},1}^P$ to be any countably infinite subset of D_{v}^P whose members are neither the empty set $\emptyset$ nor ordered pairs. Then let $D_{\mathrm{e},2}^P$ be the subset of D_{v}^P defined inductively as follows:

1. The empty set $\emptyset$ is a member of $D_{\mathrm{e},2}^P$.
2. If $x \in D_{\mathrm{e},1}^P \cup D_{\mathrm{e},2}^P$ and $y \in D_{\mathrm{e},2}^P$, then the ordered pair $\langle x, y \rangle$ of x and y is a member of $D_{\mathrm{e},2}^P$.

Finally, let $D_{\mathrm{e}}^P = D_{\mathrm{e},1}^P \cup D_{\mathrm{e},2}^P$. The members of D_{e}^P are called *constructions*. They are sets with the form of trees whose leaves are members of $D_{\mathrm{e},1}^P$. Their purpose is to represent the syntactic structure of expressions. The symbols in expressions are represented by members of $D_{\mathrm{e},1}^P$. A construction is a *symbol construction*, an *operator name construction*, *operator construction*, *type construction*, *term construction*, or *formula construction* if it represents a symbol, operator name, operator, type, term, or formula, respectively.

Choose G^P to be any bijective mapping from $\mathcal{S} \cup \mathcal{O}$ onto $D_{\mathrm{e},1}^P$. Let H^P be the bijective mapping of $\mathcal{E}_L$ onto D_{e}^P defined recursively by:

1. If $e \in \mathcal{S} \cup \mathcal{O}$, then $H^P(e) = G^P(e)$.
2. If $e = (\,) \in \mathcal{E}_L$, then $H^P(e) = \emptyset$.
3. If $e = (e_1, \ldots, e_n) \in \mathcal{E}_L$ where $n \geq 1$, then

$$H^P(e) = \langle H^P(e_1), H^P((e_2, \ldots, e_n)) \rangle.$$

$H^P(e)$ is called the *construction* of the expression e.

4.3 Structures

A *structure* for L is a tuple

$$S = (D_{\mathrm{v}}, D_{\mathrm{c}}, D_{\mathrm{s}}, D_{\mathrm{f}}, D_{\mathrm{o}}, D_{\mathrm{e}}, \in, \mathrm{T}, \mathrm{F}, \bot, \xi, H, I)$$

where:

1. $P = (D_c, \in)$ is a prestructure of Chiron. $D_{\mathrm{v}} = D_{\mathrm{v}}^P$, $D_{\mathrm{s}} = D_{\mathrm{s}}^P$, $D_{\mathrm{f}} = D_{\mathrm{f}}^P$, $D_{\mathrm{o}} = D_{\mathrm{o}}^P$, $\mathrm{T} = \mathrm{T}^P$, $\mathrm{F} = \mathrm{F}^P$, and $\bot = \bot^P$.

2. $D_{\mathrm{e}} = D_{\mathrm{e}}^P$ where $D_{\mathrm{e},1}^P$ is some chosen countably infinite subset of D_{v}^P whose members are neither the empty set $\emptyset$ nor ordered pairs.

3. ξ is a choice function on D_{s}. Hence, for all nonempty superclasses Σ in D_{s}, $\xi(\Sigma)$ is a class in Σ.

4. $H = H^P$ where G^P is some chosen bijective mapping from $\mathcal{S} \cup \mathcal{O}$ onto $D_{\mathrm{e},1}^P$.

5. For each operator name $o \in \mathcal{O}$ with $\theta(o) = s_1, \ldots, s_{n+1}$, $I(o)$ is an n-ary operation in D_{o} from $D_1 \times \cdots \times D_n$ into D_{n+1} where $D_i = D_{\mathrm{s}}$ if $s_i = \mathsf{type}$, $D_i = D_{\mathrm{c}} \cup \{\bot\}$ if $s_i = \mathsf{term}$, and $D_i = \{\mathrm{T}, \mathrm{F}\}$ if $s_i = \mathsf{formula}$ for each i with $1 \leq i \leq n+1$ such that:

   a. $I(\mathsf{set})(\,) = D_{\mathrm{v}}$, the universal class that contains all sets.

   b. $I(\mathsf{class})(\,) = D_{\mathrm{c}}$, the universal superclass that contains all classes.

   c. $I(\mathsf{op\text{-}names})(\,) = D_{\mathrm{on}}$, the subset of D_{e} whose members represent the operator names in $\mathcal{O}$.

   d. $I(\mathsf{lang})(\,)$ is the set of all subsets of D_{on}.

   e. $I(\mathsf{expr\text{-}sym})(\,) = D_{\mathrm{sy}}$, the subset of D_{e} whose members represent the symbols in $\mathcal{S}$.

   f. If x is a member of $D_{\mathrm{c}} \cup \{\bot\}$,

      $$I(\mathsf{expr\text{-}op\text{-}name})(x)$$

      is x if $x \subseteq D_{\mathrm{on}}$, D_{c} if $x = \bot$, and not D_{c} otherwise.

   g. If x is a member of $D_{\mathrm{c}} \cup \{\bot\}$,

      $$I(\mathsf{expr})(x)$$

      is the subset of D_{e} whose members are constructed from only members of $D_{\mathrm{sy}} \cup x$ if $x \subseteq D_{\mathrm{on}}$, D_{c} if $x = \bot$, and not D_{c} otherwise.

   h. Let D_{op} be the subset of D_{e} whose members represent operators of L. If x is a member of $D_{\mathrm{c}} \cup \{\bot\}$,

      $$I(\mathsf{expr\text{-}op})(x)$$

      is the subset of D_{op} whose members are constructed from only members of $D_{\mathrm{sy}} \cup x$ if $x \subseteq D_{\mathrm{on}}$, D_{c} if $x = \bot$, and not D_{c} otherwise.

i. Let D_{ty} be the subset of D_{e} whose members represent types of L. If x is a member of $D_{\mathrm{c}} \cup \{\bot\}$,

$$I(\mathsf{expr\text{-}type})(x)$$

is the subset of D_{ty} whose members are constructed from only members of $D_{\mathrm{sy}} \cup x$ if $x \subseteq D_{\mathrm{on}}$, D_{c} if $x = \bot$, and not D_{c} otherwise.

j. Let D_{te} be the subset of D_{e} whose members represent terms of L. If x is a member of $D_{\mathrm{c}} \cup \{\bot\}$,

$$I(\mathsf{expr\text{-}term})(x)$$

is the subset of D_{te} whose members are constructed from only members of $D_{\mathrm{sy}} \cup x$ if $x \subseteq D_{\mathrm{on}}$, D_{c} if $x = \bot$, and not D_{c} otherwise.

k. If x, y is a member of $D_{\mathrm{c}} \cup \{\bot\}$, then

$$I(\mathsf{expr\text{-}term\text{-}type})(x, y)$$

is the subset of D_{te} whose members represent terms of the type represented by y and are constructed from only members of $D_{\mathrm{sy}} \cup x$ if $x \subseteq D_{\mathrm{on}}$ and $y \in D_{\mathrm{ty}}$, D_{c} if $x = \bot$ or $y = \bot$, and not D_{c} otherwise.

l. Let D_{fo} be the subset of D_{e} whose members represent formulas of L. If x is a member of $D_{\mathrm{c}} \cup \{\bot\}$,

$$I(\mathsf{expr\text{-}formula})(x)$$

is the subset of D_{fo} whose members represent formulas and are constructed from only members of $D_{\mathrm{sy}} \cup x$ if $x \subseteq D_{\mathrm{on}}$, D_{c} if $x = \bot$, and not D_{c} otherwise.

m. If x and y are members of $D_{\mathrm{c}} \cup \{\bot\}$, then

$$I(\mathsf{in})(x, y)$$

is T if x is a member of y (and hence x is a member of D_{v}) and is F otherwise.

n. If Σ, Σ' are members of D_{s}, then

$$I(\mathsf{type\text{-}equal})(\Sigma, \Sigma')$$

is T if Σ and Σ' are identical and is F otherwise.

o. If x, y are members of $D_{\mathrm{c}} \cup \{\bot\}$ and Σ is a member of D_{s}, then

$$I(\mathsf{term\text{-}equal})(x, y, \Sigma)$$

is T if x, y are identical members of Σ and is F otherwise.

p. If t, t' are members of $\{\text{T}, \text{F}\}$, then

$$I(\mathsf{formula\text{-}equal})(t, t')$$

is T if t and t' are identical and is F otherwise.

q. If t is a member of $\{\text{T}, \text{F}\}$, then

$$I(\mathsf{not})(t)$$

is T if t is F and is F otherwise.

r. If t, t' are members of $\{\text{T}, \text{F}\}$, then

$$I(\mathsf{or})(t, t')$$

is T if at least one of t and t' is T and is F otherwise.

Fix a structure

$$S = (D_{\text{v}}, D_{\text{c}}, D_{\text{s}}, D_{\text{f}}, D_{\text{o}}, D_{\text{e}}, \in, \text{T}, \text{F}, \bot, \xi, H, I)$$

for L. An *assignment* into S is a mapping that assigns a class in D_{c} to each symbol in $\mathcal{S}$. Given an assignment φ into S, a symbol $s \in \mathcal{S}$, and a class $d \in D_{\text{c}}$, let $\varphi[s \mapsto d]$ be the assignment φ' into S such that $\varphi'(s) = d$ and $\varphi'(t) = \varphi(t)$ for all symbols $t \in \mathcal{S}$ different from s. Let $\mathsf{assign}(S)$ be the collection of assignments into S.

4.4 Valuations

A *valuation* for S is a possibly partial mapping V from $\mathcal{E}_L \times \mathsf{assign}(S)$ into $D_{\text{o}} \cup D_{\text{s}} \cup \{\text{T}, \text{F}, \bot\}$ such that, for all $e \in \mathcal{E}_L$ and $\varphi \in \mathsf{assign}(S)$, if $V_\varphi(e)$ is defined, then $V_\varphi(e) \in D_{\text{o}}$ if e is an operator, $V_\varphi(e) \in D_{\text{s}}$ if e is a type, $V_\varphi(e) \in D_{\text{c}} \cup \{\bot\}$ if e is a term, and $V_\varphi(e) \in \{\text{T}, \text{F}\}$ if e is a formula. (We write $V_\varphi(e)$ instead of $V(e, \varphi)$.)

Let the *standard valuation for* S be the valuation V for S defined recursively by the following statements:

1. Let e be improper. Then $V_\varphi(e)$ is undefined.

2. Let $O = (\mathsf{op}, o, k_1, \ldots, k_n, k_{n+1})$ be proper. Then $I(o)$ is an n-ary operation in D_{o} from $D_1 \times \cdots \times D_n$ into D_{n+1}. $V_\varphi(O)$ is the n-ary operation in D_{o} from $D_1 \times \cdots \times D_n$ into D_{n+1} defined as follows. Let $(d_1, \ldots, d_n) \in D_1 \times \cdots \times D_n$ and $d = I(o)(d_1, \ldots, d_n)$. If d_i is in $V_\varphi(k_i)$ or $d_i = \bot$ for all i such that $1 \leq i \leq n$ and $\mathbf{type}_L[k_i]$ and d is in $V_\varphi(k_{n+1})$ or $d = \bot$ when $\mathbf{type}_L[k_{n+1}]$, then $V_\varphi(O)(d_1, \ldots, d_n) = d$. Otherwise, $V_\varphi(O)(d_1, \ldots, d_n)$ is D_{c} if $k_{n+1} = \mathsf{type}$, $\bot$ if $\mathbf{type}_L[k_{n+1}]$, and F if $k_{n+1} = \mathsf{formula}$.

3. Let $e = (\mathsf{op\text{-}app}, O, e_1, \ldots, e_n)$ be proper. Then

$$V_\varphi(e) = V_\varphi(O)(V_\varphi(e_1), \ldots, V_\varphi(e_n)).$$

4. Let $a = (\mathsf{var}, x, \alpha)$ be proper. If $\varphi(x)$ is in $V_\varphi(\alpha)$, then $V_\varphi(a) = \varphi(x)$. Otherwise $V_\varphi(a) = \bot$.

5. Let $\beta = (\mathsf{type\text{-}app}, \alpha, a)$ be proper. If $V_\varphi(a) \neq \bot$ and $V_\varphi(\alpha)[V_\varphi(a)]$ is defined, then $V_\varphi(\beta) = V_\varphi(\alpha)[V_\varphi(a)]$. Otherwise $V_\varphi(\beta) = D_\mathrm{c}$.

6. Let $\gamma = (\mathsf{dep\text{-}fun\text{-}type}, (\mathsf{var}, x, \alpha), \beta)$ be proper. Then $V_\varphi(\gamma)$ is the superclass of all g in D_f such that, for all d in D_v, if $g(d)$ is defined, then d is in $V_\varphi(\alpha)$ and $g(d)$ is in $V_{\varphi[x \mapsto d]}(\beta)$.

7. Let $b = (\mathsf{fun\text{-}app}, f, a)$ be proper. If $V_\varphi(f) \neq \bot$, $V_\varphi(a) \neq \bot$, and $V_\varphi(f)(V_\varphi(a))$ is defined, then $V_\varphi(b) = V_\varphi(f)(V_\varphi(a))$. Otherwise $V_\varphi(b) = \bot$.

8. Let $f = (\mathsf{fun\text{-}abs}, (\mathsf{var}, x, \alpha), b)$ be proper in L. If

$$g = \{\langle d, d' \rangle \mid d \text{ is a set in } V_\varphi(\alpha) \text{ and } d' = V_{\varphi[x \mapsto d]}(b) \text{ is a set}\}$$

   is in D_f, then $V_\varphi(f) = g$. Otherwise $V_\varphi(f) = \bot$.

9. Let $a = (\mathsf{if}, A, b, c)$ be proper. If $V_\varphi(A) = \textsc{t}$, then $V_\varphi(a) = V_\varphi(b)$. Otherwise $V_\varphi(a) = V_\varphi(c)$.

10. Let $A = (\mathsf{exists}, (\mathsf{var}, x, \alpha), B)$ be proper. If there is some d in $V_\varphi(\alpha)$ such that $V_{\varphi[x \mapsto d]}(B) = \textsc{t}$, then $V_\varphi(A) = \textsc{t}$. Otherwise, $V_\varphi(A) = \textsc{f}$.

11. Let $a = (\mathsf{def\text{-}des}, (\mathsf{var}, x, \alpha), B)$ be proper. If there is a unique d in $V_\varphi(\alpha)$ such that $V_{\varphi[x \mapsto d]}(B) = \textsc{t}$, then $V_\varphi(a) = d$. Otherwise, $V_\varphi(a) = \bot$.

12. Let $a = (\mathsf{indef\text{-}des}, (\mathsf{var}, x, \alpha), B)$ be proper. If there is some d in $V_\varphi(\alpha)$ such that $V_{\varphi[x \mapsto d]}(B) = \textsc{t}$, then $V_\varphi(a) = \xi(\Sigma)$ where Σ is the superclass of all d in $V_\varphi(\alpha)$ such that $V_{\varphi[x \mapsto d]}(B) = \textsc{t}$. Otherwise, $V_\varphi(a) = \bot$.

13. Let $a = (\mathsf{quote}, e)$ be proper. $V_\varphi(a) = H(e)$.

14. Let $e = (\mathsf{eval}, a, k)$ be proper. If (1) $V_\varphi(a)$ is in D_ty and $k = \mathsf{type}$, $V_\varphi(a)$ is in D_te and $\mathbf{type}_L[k]$, or $V_\varphi(a)$ is in D_fo and $k = \mathsf{formula}$; (2) $H^{-1}(V_\varphi(a))$ is eval-free; and (3) $V_\varphi(H^{-1}(V_\varphi(a)))$ is in $V_\varphi(k)$ if $\mathbf{type}_L[k]$, then $V_\varphi(e) = V_\varphi(H^{-1}(V_\varphi(a)))$. Otherwise $V_\varphi(e)$ is D_c if $k = \mathsf{type}$, $\bot$ if $\mathbf{type}_L[k]$, and $\textsc{f}$ if $k = \mathsf{formula}$.

Lemma 4.4.1 *The standard valuation for S is well defined.*

Proof We will show that, for all $e \in \mathcal{E}_L$ and $\varphi \in \mathsf{assign}(S)$, $V_\varphi(e)$ is well defined. Our proof is by induction on the complexity of e. There are 14 cases corresponding to the 14 clauses of the definition of a standard valuation.

> Case 1: e is improper. $V_\varphi(e)$ is always undefined in this case.
>
> Cases 2–12: e is a proper expression in L that is not a quotation or evaluation. For each case, $V_\varphi(e)$ depends on well-defined components of S and a collection of values $V_{\varphi'}(e')$ where e' ranges over a set of subexpressions of e and φ' is φ or ranges over an infinite subset of $\mathsf{assign}(S)$. Each such $V_{\varphi'}(e')$ is well defined by the induction hypothesis because e' is a subexpression of e, and hence, $c(e') < c(e)$. Therefore, $V_\varphi(e)$ is well defined.
>
> Case 13: $e = (\mathsf{quote}, e')$ is proper. $V_\varphi(e) = H(e')$ is well defined since H is a well-defined component of S.
>
> Case 14: $e = (\mathsf{eval}, a, k)$ is proper. $V_\varphi(e)$ depends on one, two, or three of the values of $V_\varphi(a)$, $V_\varphi(k)$, and $V_\varphi(H^{-1}(V_\varphi(a)))$. $V_\varphi(a)$ and $V_\varphi(k)$ when $\mathbf{type}_L[k]$ holds are well defined by the induction hypothesis because a and k are subexpressions of e, and hence, $c(a) < c(e)$ and $c(k) < c(e)$. $V_\varphi(H^{-1}(V_\varphi(a)))$ when $H^{-1}(V_\varphi(a)) \in \mathcal{E}_L$ is well defined by the induction hypothesis because $H^{-1}(V_\varphi(a))$ is eval-free and e is not, and hence, $c(H^{-1}(V_\varphi(a))) < c(e)$. Therefore, $V_\varphi(e)$ is well defined.

□

Theorem 4.4.2 *Let V be the standard valuation for S. Then the following statements hold for all $e \in \mathcal{E}_L$ and $\varphi \in \mathsf{assign}(S)$:*

1. *$V_\varphi(e)$ is defined iff e is proper.*
2. *If e is an n-ary operator of L, then $V_\varphi(e)$ is an n-ary operation in D_o.*
3. *If e is a type of L, then $V_\varphi(e)$ is in D_s.*
4. *If e is a term of L of type α, then $V_\varphi(e)$ is in $V_\varphi(\alpha) \cup \{\bot\}$.*
5. *If e is a formula of L, then $V_\varphi(e)$ is in $\{\mathrm{T}, \mathrm{F}\}$.*

Proof By Lemma 4.4.1, the standard valuation for S is well defined. Parts 1, 2, 3, and 5 of the theorem follow immediately from the definitions of a structure and the standard valuation for a structure.

Our proof of part 4 is by induction on the length of the term e.

Case 1: $e = (\mathsf{op\text{-}app}, O, e_1, \ldots, e_n)$ where $O = (\mathsf{op}, o, k_1, \ldots, k_n, k_{n+1})$. Then e is of type k_{n+1}. By the definition of V, $V_\varphi(e)$ is in $V_\varphi(k_{n+1}) \cup \{\bot\}$.

Case 2: $e = (\mathsf{var}, x, \alpha)$. Then e is of type α. By the definition of V, $V_\varphi(e)$ is in $V_\varphi(\alpha) \cup \{\bot\}$.

Case 3: $e = (\mathsf{fun\text{-}app}, f, a)$ where f is of type α. Then e is of type $\alpha(a)$. By the induction hypothesis, $V_\varphi(f)$ is in $V_\varphi(\alpha) \cup \{\bot\}$. By the definition of V, if $V_\varphi(e) = V_\varphi(f)(V_\varphi(a)) \neq \bot$, then $V_\varphi(f) \neq \bot$ and $V_\varphi(a) \neq \bot$, and so $V_\varphi(e)$ is in $V_\varphi(\alpha)[V_\varphi(\alpha)] = V_\varphi(\alpha(a))$. Therefore, $V_\varphi(e)$ is in $V_\varphi(\alpha(a)) \cup \{\bot\}$.

Case 4: $e = (\mathsf{fun\text{-}abs}, (\mathsf{var}, x, \alpha), b)$ where b is of type β. Then e is of type $\gamma = (\Lambda\, x : \alpha\ .\ \beta)$. By the induction hypothesis, $V_{\varphi'}(b)$ is in $V_{\varphi'}(\beta) \cup \{\bot\}$ for all $\varphi' \in \mathsf{assign}(S)$. Suppose $g = V_\varphi(e) \neq \bot$. For all sets d in $V_\varphi(\alpha)$, if $g(d)$ is defined, $g(d) = V_{\varphi[x \mapsto d]}(b)$ is a set in $V_{\varphi[x \mapsto d]}(\beta)$. For all sets d not in $V_\varphi(\alpha)$, $g(d)$ is undefined. Therefore, by the definition of V, $V_\varphi(e)$ is in $V_\varphi(\gamma) \cup \{\bot\}$.

Case 5: $e = (\mathsf{if}, A, b, c)$ where b is of type β and c is of type γ. Then e is of type

$$\delta = \begin{cases} \beta & \text{if } \beta = \gamma \\ \mathsf{C} & \text{otherwise} \end{cases}$$

By the induction hypothesis, $V_\varphi(b)$ is in $V_\varphi(\beta) \cup \{\bot\}$ and $V_\varphi(c)$ is in $V_\varphi(\gamma) \cup \{\bot\}$. $V_\varphi(\beta) \subseteq V_\varphi(\mathsf{C})$ and $V_\varphi(\gamma) \subseteq V_\varphi(\mathsf{C})$. Therefore, by the definition of V, $V_\varphi(e)$ is in $V_\varphi(\delta) \cup \{\bot\}$.

Case 6: $e = (\mathsf{def\text{-}des}, (\mathsf{var}, x, \alpha), B)$. Then e is of type α. By the definition of V, $V_\varphi(e)$ is in $V_\varphi(\alpha) \cup \{\bot\}$.

Case 7: $e = (\mathsf{indef\text{-}des}, (\mathsf{var}, x, \alpha), B)$. Then e is of type α. By the definition of V, $V_\varphi(e)$ is in $V_\varphi(\alpha) \cup \{\bot\}$.

Case 8: $e = (\mathsf{quote}, e)$. Then e is of type E. By the definition of H, I, and V, $V_\varphi(e)$ is in $D_\mathrm{e} = V_\varphi(\mathsf{E})$.

Case 9: $e = (\mathsf{eval}, a, k)$. Then e is of type k. By the definition of V, $V_\varphi(e)$ is in $V_\varphi(k) \cup \{\bot\}$.

□

4.5 Models

A *model for* L is a pair $M = (S, V)$ where S is a structure for L and V is a valuation for S. Let $M = (S, V)$ be a model for L. An expression e is *denoting* [*nondenoting*] in M with respect to an assignment $\varphi \in \mathsf{assign}(S)$ if $V_\varphi(e)$ is defined [undefined]. A denoting term a is *defined* [*undefined*] in M with respect to φ if $V_\varphi(a)$ is in D_c [$V_\varphi(a) = \bot$]. If an expression e is denoting in M with respect to φ, then its *value* in M with respect to φ is $V_\varphi(e)$.

A model $M = (S, V)$ for L is a *standard model* for L if V is the standard valuation for S. The official semantics of Chiron is based on standard models. A formula A is *valid in* M, written $M \models A$, if $V_\varphi(A) = \mathrm{T}$ for all $\varphi \in \mathsf{assign}(S)$. A is *valid*, written $\models A$, if $M \models A$ for all standard models M. A *standard model* of a set Γ of formulas is a standard model M such that $M \models A$ for all $A \in \Gamma$.

Let e be a proper expression and x be a symbol. e is *semantically closed in* M if $V_\varphi(e)$ does not depend on φ, i.e., $V_\varphi(e) = V_{\varphi'}(e)$ for all $\varphi, \varphi' \in \mathsf{assign}(S)$. e is *semantically closed in* M *with respect to* x if $V_\varphi(e)$ does not depend on $\varphi(x)$, i.e., $V_\varphi(e) = V_{\varphi[x \mapsto d]}(e)$ for all $\varphi \in \mathsf{assign}(S)$ and $d \in D_\mathrm{c}$.

4.6 Theories

A *theory* of Chiron is pair $T = (L, \Gamma)$ where L is a language of Chiron and Γ is a set of formulas of L called the *axioms* of T. T is said to be *over* L. A *standard model* of T is a standard model for L that is a standard model of Γ. A formula A is *valid in* T, written $T \models A$, if $M \models A$ for all standard models M of T. The *empty theory over* L is the theory $(L, \emptyset)$.

Let e be a proper expression of L and T be a theory over L. e is *semantically closed in* T if e is semantically closed in every model of T. e is *semantically closed* (without reference to a theory) if it is semantically closed in the empty theory over L. e is *semantically closed in* T *with respect to* x if e is semantically closed in every model of T with respect to x.

Let $T_i = (L_i, \Gamma_i)$ be a theory of Chiron for $i = 1, 2$. T_1 is a *subtheory* of T_2 (and T_2 is an *extension* of T_1), written $T_1 \leq T_2$, if $L_1 \leq L_2$ and $\Gamma_1 \subseteq \Gamma_2$.

Lemma 4.6.1 *Let $T_i = (L_i, \Gamma_i)$ be a theory of Chiron for $i = 1, 2$ such that $T_1 \leq T_2$. Assume $L_1 = L_2$. Then*

$$T_1 \models A \text{ implies } T_2 \models A$$

for all formulas A of L_1.

Proof Assume the hypotheses of the lemma. Let A be a formula of L_1 such that (a) $T_1 \models A$. Let (b) $M = (S, V)$ be a standard model of T_2. We must show that $M \models A$. (b) implies (c) M is a standard model of T_1 since $T_1 \leq T_2$ and $L_1 = L_2$. (a) and (c) imply $M \models A$. □

The following example shows that the assumption in Lemma 4.6.1 that $L_1 = L_2$ is necessary.

Example 4.6.2 Let $L_i = (\mathcal{O}_i, \theta_i)$ be a language of Chiron and $T_i = (L_i, \Gamma_i)$ be a theory of Chiron for $i = 1, 2$ such that $\mathcal{O}_1 = \{o_1, \ldots, o_n\}$, $L_1 < L_2$, and $T_1 \leq T_2$. If A is $\ell = \{\ulcorner o_1 \urcorner, \ldots, \ulcorner o_n \urcorner\}$, then $T_1 \models A$ but $T_2 \not\models A$.

4.7 Notes Concerning the Semantics of Chiron

Fix a standard model $M = (S, V)$ for L.

1. An improper expression never has a value, but its quotation (as well as the quotation of any proper expression) always has a value. The latter is a set called a construction that represents the syntactic structure of the expression as a tree.

2. The value of an undefined type is D_{c} (the universal superclass). The value of an undefined term is the undefined value $\perp$. And the value of an undefined formula is F.

3. Proper expressions—particularly variables and evaluations—within a quotation are not semantically "active". They can only become active if the quotation is within an evaluation.

4. Any types in an operator's signature will normally be semantically closed. This is the case for all the built-in operators of Chiron as well as all the operators defined in sections 5 and 6.

5. Suppose $O = (\mathsf{op}, o, k_1, \ldots, k_n, k_{n+1})$ is an operator. The operation assigned to O by a standard valuation is a "restriction" of the operation $I(o)$. Roughly speaking, the domain and range of $I(o)$ is restricted according to types in the signature $k_1, \ldots, k_n, k_{n+1}$.

6. Dependent function types, function abstractions, existential quantifications, definition descriptions, and indefinite descriptions are *variable binding* expressions. According to the semantics of Chiron, a variable binding $(\star\, x : \alpha\ .\ e)$, where $\star$ is Λ, λ, $\exists$, ι, or ϵ, binds all the "free" variables occurring in e that are similar to $(x : \alpha)$. Variables are bound in the traditional, naive way. It might be possible to use other more sophisticated variable binding mechanisms such as de Bruijn notation [2] or nominal data types [11, 20].

7. The type α of a variable $(\mathsf{var}, x, \alpha)$ restricts the value of a variable in two ways. First, if $(\mathsf{var}, x, \alpha)$ immediately follows Λ, λ, $\exists$, ι, or ϵ in a variable binding expression, then the values assigned to x for the body of the variable binding expression are restricted to the values in the superclass denoted by α. Second, $V_\varphi((\mathsf{var}, x, \alpha)) = \varphi(x)$ iff $\varphi(x)$ is in the superclass denoted by α. ($V_\varphi((\mathsf{var}, x, \alpha)) = \bot$ if $\varphi(x)$ is not in the superclass denoted by α.)

8. In a variable $(\mathsf{var}, x, \alpha)$, the value of α usually does not depend on the value assigned to x. Suppose $d \in V_\varphi(\alpha)$. If $V_\varphi(\alpha)$ does not depend on x, then $V_{\varphi[x \mapsto d]}((\mathsf{var}, x, \alpha)) = d$. If $V_\varphi(\alpha)$ does depend on x, then $V_{\varphi[x \mapsto d]}((\mathsf{var}, x, \alpha))$ might equal $\bot$. Hence, in the latter case, it might be necessary to use $(\mathsf{var}, x, \mathsf{C})$ instead of $(\mathsf{var}, x, \alpha)$ in the body of a variable binding $(\star\, x : \alpha\ .\ e)$ where $\star$ is Λ, λ, $\exists$, ι, or ϵ

9. Symbols and operator names are atomic in Chiron. That is, they cannot be "constructed" from other expressions.

10. ℓ, the set of constructions that represent the operator names of L, represents L itself. The members of the type L represent sublanguages of L. The construction types $\mathsf{E}_{\mathrm{on},a}$, E_a, $\mathsf{E}_{\mathrm{op},a}$, $\mathsf{E}_{\mathrm{ty},a}$, $\mathsf{E}_{\mathrm{te},a}$, $\mathsf{E}^b_{\mathrm{te},a}$, $\mathsf{E}_{\mathrm{fo},a}$, where a is a sublanguage in L, are parameterized versions of the respective types E_{on}, E, E_{op}, E_{ty}, E_{te}, $\mathsf{E}^b_{\mathrm{te}}$, E_{fo}. (The construction type E_{sy} does not depend on the set of operator names.)

11. The notions of a free variable, substitution for a variable, variable capturing, etc. can be formalized in Chiron as defined operators (see subsection 6.1.) As a result, syntactic side conditions can be expressed directly within Chiron formulas. However, these notions are more complicated in Chiron than in traditional predicate logic due to the presence of evaluation. For example, when the value of $(e : \mathsf{E}_{\mathrm{te}})$ equals

the value of $\ulcorner(y : \mathsf{C})\urcorner$ with $x \neq y$, the variable $(y : \mathsf{C})$ is free in

$$\forall\, x : \mathsf{C} \;.\; x = [\![(e : \mathsf{E}_{\mathrm{te}})]\!]_{\mathrm{te}}$$

because this formula is semantically equivalent to

$$\forall\, x : \mathsf{C} \;.\; x = (y : \mathsf{C}).$$

12. Since variables denote classes, they can be called *class variables*. There are no operation, superclass, or truth value variables in Chiron. Thus direct quantification over operations, superclasses, and truth values is not possible. Direct quantification over the undefined value $\bot$ is also not possible. However, indirect quantification over "definable" operations, "definable" superclasses, "definable" members of $D_{\mathrm{c}} \cup \{\bot\}$, or truth values can be done via variables of type E_{op}, E_{ty}, E_{te}, or E_{fo}, respectively. As in standard NBG set theory, only classes are first-class values in Chiron.

13. Since sets and classes are superclasses, a type may denote a set or a proper class. In particular, a type may denote the *empty set*. That is, types in Chiron are allowed to be empty. Empty types result, for example, from a type application $\alpha(a)$ where a denotes a value that is not in the domain of any function in the superclass that α denotes.

14. α is a *subtype* of β in M if $V_\varphi(\alpha) \subseteq V_\varphi(\beta)$ for all $\varphi \in \mathsf{assign}(S)$. For example, E is a subtype of V in every standard model. C is the *universal type* by virtue of denoting D_c, the universal superclass. Every type is a subtype of C in every standard model.

15. An application of a term denoting a function to an undefined term is itself undefined. That is, function application is strict with respect to undefinedness. In contrast, the application of term operators is not necessarily strict with respect to undefinedness.

16. Suppose $f(a)$ is a function application where f is a term of type $(\Lambda\, x : \alpha \;.\; \beta)$. Then a could be any term of any type whatsoever. However, if the value of a is not a set in the value of α, then $f(a)$ is undefined. Similarly, suppose

$$(o :: k_1, \ldots, k_{i-1}, \alpha, k_{i+1}, \ldots, k_{n+1})(e_1, \ldots, e_{i-1}, a, e_{i+1}, \ldots, e_n)$$

is an operator application. Then a could be any term of any type whatsoever. However, if the value of a is a class not in the value of α, then the value of this operator application is D_{c} if $k_{n+1} = \mathsf{type}$, $\bot$ if $\mathbf{type}_L[k_{n+1}]$, and F if $k_{n+1} = \mathsf{formula}$.

17. Since functions are classes that represent mappings from sets to sets, proper classes in the denotations of the types α and β have no effect on the meaning of the dependent function type $(\Lambda\, x : \alpha\ .\ \beta)$. As a consequence,

$$V_\varphi((\Lambda\, x : \mathsf{V}\ .\ \mathsf{V})) = V_\varphi((\Lambda\, x : \mathsf{C}\ .\ \mathsf{C})) = D_{\mathrm{f}},$$

the domain of functions in M, for all $\varphi \in \mathsf{assign}(S)$. For the same reason, if $V_\varphi(a) \neq \bot$, then $V_\varphi(\alpha(a))$ is a class (i.e., not a proper superclass) for all $\varphi \in \mathsf{assign}(S)$.

18. Suppose a built-in operator

$$(\mathsf{op}, \mathsf{lub}, \mathsf{type}, \mathsf{type}, \mathsf{type})$$

is added to Chiron that denotes an operation that, given superclasses Σ_1 and Σ_2, returns a superclass that is the least upper bound of Σ_1 and Σ_2. Then formation rule P-Expr-8 could be modified so that a conditional term (if, A, b, c) is assigned the type

$$(\mathsf{op\text{-}app}, (\mathsf{op}, \mathsf{lub}, \mathsf{type}, \mathsf{type}, \mathsf{type}), \beta, \gamma)$$

where β and γ are the types of b and c, respectively. See appendix B for details.

19. A *Gödel number* [12] is a number that encodes an expression. Analogously, a *Gödel set* is a set that encodes an expression. Employing this terminology, the construction that represents an expression $e \in \mathcal{E}_L$ is the *Gödel set* of e which is denoted by (quote, e). Hence, "Gödel numbering" is built into Chiron.

20. Quotations cannot be expressed as applications of a built-in operator since expressions are not values (i.e., not classes, superclasses, truth values, or operations).

21. The formation rule P-Expr-12 could be sharpened so that the type of (quote, e) is E_{ty} when e is a type, $\mathsf{E}_{\mathrm{te}}^{\ulcorner\alpha\urcorner}$ when e is a term of type α, etc. See appendix B for details.

22. When an evaluation $b = (\mathsf{eval}, a, k)$ is "semantically well-formed", the value of b is, roughly speaking, *the value of the value of* a.

23. An "ungrounded expression" is considered to be an undefined expression. For example, the value of an ungrounded formula like $[\![\mathsf{LIAR}]\!]_{\mathrm{fo}}$ is F.

24. If we restrict evaluations to those of the form (eval, a, k) where k is type, C, or formula, evaluations could be expressed as applications of the following three built-in operators with appropriate definitions:

    a. $(\mathsf{eval\text{-}type} :: \mathsf{E}_{\mathrm{ty}}, \mathsf{type})$.

    b. $(\mathsf{eval\text{-}term} :: \mathsf{E}_{\mathrm{te}}, \mathsf{C})$.

    c. $(\mathsf{eval\text{-}formula} :: \mathsf{E}_{\mathrm{fo}}, \mathsf{formula})$.

25. The operator $(\mathsf{eval\text{-}formula} :: \mathsf{E}_{\mathrm{fo}}, \mathsf{formula})$ can be defined by the following formula schema:

    $$(\mathsf{eval\text{-}formula} :: \mathsf{E}_{\mathrm{fo}}, \mathsf{formula})(a) \equiv [\![a]\!]_{\mathrm{fo}}.$$

    This operator is a truth predicate. It satisfies four of the eight criteria ((a), (c), (f), (h)) for a truth predicate given by H. Leitgeb in [16], and it meets all eight criteria for eval-free formulas.

26. Operators of the form $(o :: k)$ are applied to an empty tuple of arguments. The value of $(o :: k)$ is a 0-ary operation σ such that $\sigma(\,) = v$ for some value v. We will sometimes abuse terminology and say that the value of $(o :: k)$ is v instead of o.

27. A value $x \in D_{\mathrm{e}}$ is a construction that represents an expression of L. We will sometimes abuse terminology and say $x \in D_{\mathrm{e}}$ is an *expression* of L. Similarly, we will sometimes say that a value $x \in D_{\mathrm{e}}$ that represents a symbol, operator name, operator, type, term, or formula of L is a *symbol*, *operator name*, *operator*, *type*, *term*, or *formula* of L, respectively.

28. Let $\mathcal{E}_{\mathrm{op}}$, $\mathcal{E}_{\mathrm{ty}}$, $\mathcal{E}_{\mathrm{te}}$, and $\mathcal{E}_{\mathrm{fo}}$ be sets of expressions of L defined inductively by the following statements:

    a. $\mathcal{E}_{\mathrm{op}}$ is the set of built-in operators named set, class, term-equal, in, not, and or.

b. $\mathcal{E}_{\text{ty}}$ is the set $\{\mathsf{V}, \mathsf{C}\}$ of types.

c. If x is in $\mathcal{S}$ and α is in $\mathcal{E}_{\text{ty}}$, then $(x : \alpha)$ is in $\mathcal{E}_{\text{te}}$.

d. If x is in $\mathcal{S}$, α is in $\mathcal{E}_{\text{ty}}$, a and b are in $\mathcal{E}_{\text{te}}$, and A and B are in $\mathcal{E}_{\text{fo}}$, then $a = b$, $a \in b$, $\neg A$, $A \vee B$, and $\exists\, x : \alpha\ .\ A$ are in $\mathcal{E}_{\text{fo}}$.

Let CNBG be the restriction of Chiron to the operators, types, terms, and formulas in $\mathcal{E}_{\text{op}}$, $\mathcal{E}_{\text{ty}}$, $\mathcal{E}_{\text{te}}$, and $\mathcal{E}_{\text{fo}}$, respectively. CNBG is a version of NBG embedded in any language of Chiron (see Corollary 4.8.3 below).

4.8 Relationship to NBG

We show in this section that there is a faithful semantic interpretation of NBG set theory in Chiron. Loosely speaking, this means Chiron is a conservative extension of NBG. That is, Chiron adds new reasoning machinery to NBG without compromising the underlying semantics of NBG.

Let $L = (\mathcal{O}, \theta)$ be any language of Chiron. NBG is usually formulated as a theory in first-order logic over a language L_{nbg} containing an infinite set $\mathcal{V}$ of variables, a unary predicate symbol V, binary predicates symbols $=$ and $\in$, and some complete set of logical connectives (say $\neg, \vee, \exists$). Assume $\mathcal{V} = \mathcal{S}$, i.e., variables of L_{nbg} are the symbols of Chiron. Let $\mathcal{F}_{\text{nbg}}$ denote the set of formulas of L_{nbg}. A *model* of NBG is a structure $N = (D^N, V^N, =^N, \in^N)$ for L_{nbg} that satisfies the axioms of NBG. An *assignment* into N is a mapping that assigns a member of D^N to each variable $x \in \mathcal{V}$. Let $\mathsf{assign}(N)$ be the collection of assignments into N. The *valuation* for N is a total mapping W^N from $\mathcal{F}_{\text{nbg}} \times \mathsf{assign}(N)$ to the set $\{\text{T}, \text{F}\}$ of truth values. A formula A of L_{nbg} is *valid*, written $\models_{\text{nbg}} A$, if $W^N_\varphi(A) = \text{T}$ for all models N of NBG and all $\varphi \in \mathsf{assign}(N)$.[4]

Φ is a *translation from* NBG *to* L if Φ maps the terms (variables) of L_{nbg} to the terms of L and the formulas of L_{nbg} to formulas of L. Φ is a *(semantic) interpretation of* NBG *in* Chiron if Φ is a translation from NBG to L and, for all formulas A of L_{nbg}, $\models_{\text{nbg}} A$ implies $\models \Phi(A)$. That is, Φ is an interpretation of NBG in Chiron if Φ is a meaning-preserving translation from NBG to L. Φ is a *faithful interpretation of* NBG *in* Chiron if Φ is an interpretation of NBG in Chiron and, for all formulas A of L_{nbg}, $\models \Phi(A)$ implies $\models_{\text{nbg}} A$. That is, Φ is a faithful interpretation of NBG in Chiron if Chiron over L is a conservative extension of the image of NBG under Φ.

Let Φ be the translation from NBG to L recursively defined by:

1. If $x \in \mathcal{V}$, then $\Phi(x) = (x : \mathsf{C})$.

[4] As above for a valuation V, we write $W^N(A, \varphi)$ as $W^N_\varphi(A)$.

2. If $V(x)$ is a formula of L_{nbg}, then $\Phi(V(x)) = (\Phi(x) =_{\mathsf{V}} \Phi(x))$.

3. If $(x = y)$ is a formula of L_{nbg}, then $\Phi((x = y)) = (\Phi(x) = \Phi(y))$.

4. If $(x \in y)$ is a formula of L_{nbg}, then $\Phi((x \in y)) = (\Phi(x) \in \Phi(y))$.

5. If $(\neg A)$ is a formula of L_{nbg}, then $\Phi((\neg A)) = (\neg\Phi(A))$.

6. If $(A \vee B)$ is a formula of L_{nbg}, then $\Phi((A \vee B)) = (\Phi(A) \vee \Phi(B))$.

7. If $(\exists\, x\ .\ A)$ is a formula of L_{nbg}, then $\Phi((\exists\, x\ .\ A)) = (\exists\, x : \mathsf{C}\ .\ \Phi(A))$.

Lemma 4.8.1 *Let $N = (D^N, V^N, =^N, \in^N)$ be a model of* NBG *and $M = (S, V)$ be a standard model for L, where*

$$S = (D_{\mathrm{v}}, D_{\mathrm{c}}, D_{\mathrm{s}}, D_{\mathrm{f}}, D_{\mathrm{o}}, D_{\mathrm{e}}, \in, \mathrm{T}, \mathrm{F}, \bot, \xi, H, I),$$

such that $(D^N, \in^N)$ is identical to the prestructure $(D_{\mathrm{c}}, \in)$.

1. *For all $d \in D^N$, $V^N(d)$ iff d is in D_{v}.*

2. *For all $d, d' \in D^N$, $d =^N d'$ iff $d = d'$.*

3. *For all $d, d' \in D^N$, $d \in^N d'$ iff $d \in d'$.*

4. *$\mathsf{assign}(N) = \mathsf{assign}(M)$.*

5. *For all formulas A of L_{nbg} and $\varphi \in \mathsf{assign}(N)$, $W^N_\varphi(A) = V_\varphi(\Phi(A))$.*

Proof Clauses 1–4 are obvious. Clause 5 is easily proved by induction on the structure of the formulas of L_{nbg} since the logical connectives $\neg, \vee, \exists$ of L_{nbg} have the same meanings as the negation operator, the disjunction operator, and existential quantification, respectively, in Chiron. □

Theorem 4.8.2 *For all formulas A of L_{nbg},*

$$\models_{\mathrm{NBG}} A \quad \textit{iff} \quad \models \Phi(A).$$

That is, Φ is a faithful interpretation of NBG *in Chiron.*

Proof For every model $N = (D^N, V^N, =^N, \in^N)$ of NBG, there is a standard model $M = (S, V)$ for L, where

$$S = (D_{\mathrm{v}}, D_{\mathrm{c}}, D_{\mathrm{s}}, D_{\mathrm{f}}, D_{\mathrm{o}}, D_{\mathrm{e}}, \in, \mathrm{T}, \mathrm{F}, \bot, \xi, H, I),$$

such that $(D^N, \in^N)$ is identical to the prestructure $(D_c, \in)$. Likewise, for every standard model $M = (S, V)$ for L, where

$$S = (D_v, D_c, D_s, D_f, D_o, D_e, \in, \mathrm{T}, \mathrm{F}, \bot, \xi, H, I),$$

there is a model $N = (D^N, V^N, =^N, \in^N)$ of NBG such that the prestructure $(D_c, \in)$ is identical to $(D^N, \in^N)$. The theorem follows from this observation and clause 5 of Lemma 4.8.1. □

In the previous subsection we defined a restriction of Chiron named CNBG that we claimed is a version of NBG embedded in any language of Chiron. This claim is established by the following corollary.

Corollary 4.8.3 Φ *is a faithful interpretation of* NBG *in* CNBG.

5 Operator Definitions

There are two ways of assigning a meaning to an operator. The first is to make the operator one of the built-in operators like $(\mathsf{op}, \mathsf{set}, \mathsf{type})$ ($(\mathsf{set} :: \mathsf{type})$ in compact notation) and then define its meaning as part of the definition of a standard model. The second is to construct one or more formulas that together define the meaning of an operator. We will use this latter approach to define several useful logical, set-theoretic, and syntactic operators in this section and substitution operators in the next section.

Each operator definition consists of an operator

$$O = (o :: k_1, \ldots, k_{n+1})$$

and a set of defining axioms for O. The definition defines o to be an operator name, $\theta(o)$ to be the signature form corresponding to $k_1, \ldots, k_{n+1}$, and $k_1, \ldots, k_{n+1}$ to be the preferred signature of o. Each defining axiom is usually eval-free formula. The defining axioms are presented as individual formulas or as formula schemas. The set of defining axioms presented by a formula schema usually depends on the language that is being considered. An operator definition may optionally include compact syntax for applications of the defined operator.

5.1 Logical Operators

1. **Truth**
   Operator: $(\mathsf{true} :: \mathsf{formula})$

Defining axioms:

$$(\mathsf{true} :: \mathsf{formula})(\,) \equiv \mathsf{C} =_{\mathrm{ty}} \mathsf{C}.$$

Compact notation:

T means $(\mathsf{true} :: \mathsf{formula})(\,)$.

2. **Falsehood**
   Operator: $(\mathsf{false} :: \mathsf{formula})$
   Defining axioms:

$$(\mathsf{false} :: \mathsf{formula})(\,) \equiv \mathsf{V} =_{\mathrm{ty}} \mathsf{C}.$$

   Compact notation:

   F means $(\mathsf{false} :: \mathsf{formula})(\,)$.

3. **Conjunction**
   Operator: $(\mathsf{and} :: \mathsf{formula}, \mathsf{formula}, \mathsf{formula})$
   Defining axioms:

$$(\mathsf{and} :: \mathsf{formula}, \mathsf{formula}, \mathsf{formula})(A^{\mathrm{ef}}, B^{\mathrm{ef}}) \equiv \neg(\neg A^{\mathrm{ef}} \vee \neg B^{\mathrm{ef}}).$$

   Compact notation:

   $(A \wedge B)$ means $(\mathsf{and} :: \mathsf{formula}, \mathsf{formula}, \mathsf{formula})(A, B)$.

   Note: The defining axioms are presented as a formula schema. Recall that metavariables like A^{ef} and B^{ef} denote eval-free formulas.

4. **Implication**
   Operator: $(\mathsf{implies} :: \mathsf{formula}, \mathsf{formula}, \mathsf{formula})$
   Defining axioms:

$$(\mathsf{implies} :: \mathsf{formula}, \mathsf{formula}, \mathsf{formula})(A^{\mathrm{ef}}, B^{\mathrm{ef}}) \equiv \neg A^{\mathrm{ef}} \vee B^{\mathrm{ef}}.$$

   Compact notation:

   $(A \supset B)$ means $(\mathsf{implies} :: \mathsf{formula}, \mathsf{formula}, \mathsf{formula})(A, B)$.

5. **Definedness in a Type**
   Operator: $(\mathsf{defined\text{-}in} :: \mathsf{C}, \mathsf{type}, \mathsf{formula})$
   Defining axioms:

$$(\mathsf{defined\text{-}in} :: \mathsf{C}, \mathsf{type}, \mathsf{formula})(a^{\mathrm{ef}}, \alpha^{\mathrm{ef}}) \equiv (a^{\mathrm{ef}} =_{\alpha^{\mathrm{ef}}} a^{\mathrm{ef}}).$$

   Compact notation:

   $(a \downarrow \alpha)$ means $(\mathsf{defined\text{-}in} :: \mathsf{C}, \mathsf{type}, \mathsf{formula})(a, \alpha)$.
   $(a \uparrow \alpha)$ means $\neg(a \downarrow \alpha)$.
   $(a\downarrow)$ means $(a \downarrow \mathsf{C})$.
   $(a\uparrow)$ means $\neg(a\downarrow)$.

6. **Quasi-Equality**
   Operator: $(\mathsf{quasi\text{-}equal} :: \mathsf{C}, \mathsf{C}, \mathsf{formula})$
   Defining axioms:

$$(\mathsf{quasi\text{-}equal} :: \mathsf{C}, \mathsf{C}, \mathsf{formula})(a^{\mathrm{ef}}, b^{\mathrm{ef}}) \equiv$$
$$(a^{\mathrm{ef}}\downarrow \vee b^{\mathrm{ef}}\downarrow) \supset a^{\mathrm{ef}} = b^{\mathrm{ef}}.$$

   Note: The application of quasi-equal to two undefined terms is true. Hence quasi-equal is not strict with respect to undefinedness. Nearly all the operators defined in this section are strict with respect to undefinedness.

   Compact notation:

   $(a \simeq b)$ means $(\mathsf{quasi\text{-}equal} :: \mathsf{C}, \mathsf{C}, \mathsf{formula})(a, b)$.
   $(a \not\simeq b)$ means $\neg(a \simeq b)$.

7. **Canonical Undefined Term**
   Operator: $(\mathsf{undefined} :: \mathsf{C})$
   Defining axioms:

$$(\mathsf{undefined} :: \mathsf{C})(\,) \simeq (\iota\, x : \mathsf{C}\ .\ x \neq x).$$

   Compact notation:

   $\perp_{\mathsf{C}}$ means $(\mathsf{undefined} :: \mathsf{C})(\,)$.

8. **Canonical Empty Type**
   Operator: $(\mathsf{empty\text{-}type} :: \mathsf{type})$
   Defining axioms:

$$\forall\, x : \mathsf{C} \,.\, x \uparrow (\mathsf{empty\text{-}type} :: \mathsf{type})(\,).$$

   Compact notation:

∇ means $(\mathsf{empty\text{-}type} :: \mathsf{type})(\,)$.

9. **Type Order**
   Operator: $(\mathsf{type\text{-}le} :: \mathsf{type}, \mathsf{type}, \mathsf{formula})$
   Defining axioms:

$$(\neg \mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner \alpha^{\mathrm{ef}} \urcorner) \wedge \neg \mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner \beta^{\mathrm{ef}} \urcorner)) \supset$$
$$(\mathsf{type\text{-}le} :: \mathsf{type}, \mathsf{type}, \mathsf{formula})(\alpha^{\mathrm{ef}}, \beta^{\mathrm{ef}}) \equiv \forall\, x : \alpha^{\mathrm{ef}} \,.\, x \downarrow \beta^{\mathrm{ef}}.$$

   Compact notation:

$(\alpha \ll \beta)$ means $(\mathsf{type\text{-}le} :: \mathsf{type}, \mathsf{type}, \mathsf{formula})(\alpha, \beta)$.

   Note: The formula schema of defining axioms utilizes the free-in operator defined in subsection 6.1.

10. **Conditional Type**
    Operator: $(\mathsf{if\text{-}type} :: \mathsf{formula}, \mathsf{type}, \mathsf{type}, \mathsf{type})$
    Defining axioms:

$$A^{\mathrm{ef}} \supset$$
$$(\mathsf{if\text{-}type} :: \mathsf{formula}, \mathsf{type}, \mathsf{type}, \mathsf{type})(A^{\mathrm{ef}}, \beta^{\mathrm{ef}}, \gamma^{\mathrm{ef}}) =_{\mathrm{ty}} \beta^{\mathrm{ef}}.$$

$$\neg A^{\mathrm{ef}} \supset$$
$$(\mathsf{if\text{-}type} :: \mathsf{formula}, \mathsf{type}, \mathsf{type}, \mathsf{type})(A^{\mathrm{ef}}, \beta^{\mathrm{ef}}, \gamma^{\mathrm{ef}}) =_{\mathrm{ty}} \gamma^{\mathrm{ef}}.$$

    Compact notation:

$\mathsf{if}(A, \beta, \gamma)$ means
$(\mathsf{if\text{-}type} :: \mathsf{formula}, \mathsf{type}, \mathsf{type}, \mathsf{type})(A, \beta, \gamma)$.

11. **Conditional Formula**
Operator: $(\mathsf{if\text{-}formula} :: \mathsf{formula}, \mathsf{formula}, \mathsf{formula}, \mathsf{formula})$
Defining axioms:

$$A^{\mathrm{ef}} \supset (\mathsf{if\text{-}formula} :: \mathsf{formula}, \mathsf{formula}, \mathsf{formula}, \mathsf{formula})(A^{\mathrm{ef}}, B^{\mathrm{ef}}, C^{\mathrm{ef}}) =_{\mathrm{ty}} B^{\mathrm{ef}}.$$

$$\neg A^{\mathrm{ef}} \supset (\mathsf{if\text{-}formula} :: \mathsf{formula}, \mathsf{formula}, \mathsf{formula}, \mathsf{formula})(A^{\mathrm{ef}}, B^{\mathrm{ef}}, C^{\mathrm{ef}}) =_{\mathrm{ty}} C^{\mathrm{ef}}.$$

Compact notation:

$\mathsf{if}(A, B, C)$ means $(\mathsf{if\text{-}formula} :: \mathsf{formula}, \mathsf{formula}, \mathsf{formula}, \mathsf{formula})(A, B, C)$.

12. **Simple Function Type**
Operator: $(\mathsf{sim\text{-}fun\text{-}type} :: \mathsf{type}, \mathsf{type}, \mathsf{type})$
Defining axioms:

$$(\mathsf{sim\text{-}fun\text{-}type} :: \mathsf{type}, \mathsf{type}, \mathsf{type})(\alpha^{\mathrm{ef}}, \beta^{\mathrm{ef}}) =_{\mathrm{ty}} \mathsf{if}(\mathsf{syn\text{-}closed}(\ulcorner \beta^{\mathrm{ef}} \urcorner), (\Lambda\, x : \alpha^{\mathrm{ef}}\ .\ \beta^{\mathrm{ef}}), \mathsf{C}).$$

Compact notation:

$(\alpha \to \beta)$ means $(\mathsf{sim\text{-}fun\text{-}type} :: \mathsf{type}, \mathsf{type}, \mathsf{type})(\alpha, \beta)$.

Note: The formula schema of defining axioms utilizes the syn-closed operator defined in subsection 6.1.

5.2 Set-Theoretic Operators

1. **Empty set**
Operator: $(\mathsf{empty\text{-}set} :: \mathsf{V})$
Defining axioms:

$$(\mathsf{empty\text{-}set} :: \mathsf{V})(\,) \simeq (\iota\, u : \mathsf{V}\ .\ \forall\, v : \mathsf{V}\ .\ v \notin u).$$

Compact notation:

$\emptyset$ means $(\mathsf{empty\text{-}set} :: \mathsf{V})(\,)$.

2. **Universal class**
   Operator: $(\mathsf{universal\text{-}class} :: \mathsf{C})$
   Defining axioms:

$$(\mathsf{univeral\text{-}class} :: \mathsf{C})(\,) \simeq \iota\, x : \mathsf{C}\ .\ \forall\, u : \mathsf{V}\ .\ u \in x.$$

   Compact notation:

   U means $(\mathsf{universal\text{-}set} :: \mathsf{C})(\,)$.

3. **Pair**
   Operator: $(\mathsf{pair} :: \mathsf{V}, \mathsf{V}, \mathsf{V})$
   Defining axioms:

$$\forall\, u, v : \mathsf{V}\ .\ (\mathsf{pair} :: \mathsf{V}, \mathsf{V}, \mathsf{V})(u, v) \simeq$$
$$\iota\, w : \mathsf{V}\ .\ \forall\, w' : \mathsf{V}\ .\ w' \in w \equiv (w' = u \vee w' = v).$$

$$(a^{\mathrm{ef}}{\uparrow} \vee b^{\mathrm{ef}}{\uparrow}) \supset (\mathsf{pair} :: \mathsf{V}, \mathsf{V}, \mathsf{V})(a^{\mathrm{ef}}, b^{\mathrm{ef}}){\uparrow}\ .$$

   Note: The formula schema of defining axioms that comes second asserts that pair is strict with respect to undefinedness. The operators named singleton, triple, etc. are defined in a similar way to pair.

   Compact notation:

   $\{\,\}$ means $\emptyset$.
   $\{a\}$ means $(\mathsf{singleton} :: \mathsf{V}, \mathsf{V})(a)$.
   $\{a, b\}$ means $(\mathsf{pair} :: \mathsf{V}, \mathsf{V}, \mathsf{V})(a, b)$.
   $\{a, b, c\}$ means $(\mathsf{triple} :: \mathsf{V}, \mathsf{V}, \mathsf{V}, \mathsf{V})(a, b, c)$.
   $\vdots$

4. **Ordered Pair**
   Operator: $(\mathsf{ord\text{-}pair} :: \mathsf{V}, \mathsf{V}, \mathsf{V})$
   Defining axioms:

$$\forall\, u, v : \mathsf{V}\ .\ (\mathsf{ord\text{-}pair} :: \mathsf{V}, \mathsf{V}, \mathsf{V})(u, v) \simeq$$
$$\{\{u\}, \{u, v\}\}.$$

$$(a^{\mathrm{ef}}\!\uparrow \vee b^{\mathrm{ef}}\!\uparrow) \supset (\mathsf{ord\text{-}pair} :: \mathsf{V},\mathsf{V},\mathsf{V})(a^{\mathrm{ef}}, b^{\mathrm{ef}})\!\uparrow .$$

Compact notation:

$\langle a, b\rangle$ means $(\mathsf{ord\text{-}pair} :: \mathsf{V},\mathsf{V},\mathsf{V})(a, b)$.
$\langle a_1, \ldots, a_n\rangle$ means $\langle a_1, \langle a_2, \ldots, a_n\rangle\rangle$ for $n \geq 3$.
$[\,]$ means $\emptyset$.
$[a_1, \ldots, a_n]$ means $\langle a_1, [a_2, \ldots, a_n]\rangle$ for $n \geq 1$.

5. **Subclass**
Operator: $(\mathsf{subclass} :: \mathsf{C},\mathsf{C},\mathsf{formula})$
Defining axioms:

$$\forall\, x, y : \mathsf{C} \;.\; (\mathsf{subclass} :: \mathsf{C},\mathsf{C},\mathsf{formula})(x, y) \equiv \forall\, u : \mathsf{V} \;.\; u \in x \supset u \in y.$$

$$(a^{\mathrm{ef}}\!\uparrow \vee b^{\mathrm{ef}}\!\uparrow) \supset (\mathsf{subclass} :: \mathsf{C},\mathsf{C},\mathsf{formula})(a^{\mathrm{ef}}, b^{\mathrm{ef}}) \equiv \mathsf{F}.$$

Compact notation:

$a \subseteq b$ means $(\mathsf{subclass} :: \mathsf{C},\mathsf{C},\mathsf{formula})(a, b)$.
$a \subset b$ means $a \subseteq b \wedge a \neq b$.

6. **Union**
Operator: $(\mathsf{union} :: \mathsf{C},\mathsf{C},\mathsf{C})$
Defining axioms:

$$\forall\, x, y : \mathsf{C} \;.\; (\mathsf{union} :: \mathsf{C},\mathsf{C},\mathsf{C})(x, y) \simeq \iota\, z : \mathsf{C} \;.\; \forall\, u : \mathsf{V} \;.\; u \in z \equiv (u \in x \vee u \in y).$$

$$(a^{\mathrm{ef}}\!\uparrow \vee b^{\mathrm{ef}}\!\uparrow) \supset (\mathsf{union} :: \mathsf{C},\mathsf{C},\mathsf{C})(a^{\mathrm{ef}}, b^{\mathrm{ef}})\!\uparrow .$$

Compact notation:

$a \cup b$ means $(\mathsf{union} :: \mathsf{C},\mathsf{C},\mathsf{C})(a, b)$.

7. **Intersection**
   Operator: $(\mathsf{intersection} :: \mathsf{C}, \mathsf{C}, \mathsf{C})$
   Defining axioms:

$$\forall\, x, y : \mathsf{C}\ .\ (\mathsf{intersection} :: \mathsf{C}, \mathsf{C}, \mathsf{C})(x, y) \simeq \\ \quad \iota\, z : \mathsf{C}\ .\ \forall\, u : \mathsf{V}\ .\ u \in z \equiv (u \in x \wedge u \in y).$$

$$(a^{\mathrm{ef}}{\uparrow} \vee b^{\mathrm{ef}}{\uparrow}) \supset (\mathsf{intersection} :: \mathsf{C}, \mathsf{C}, \mathsf{C})(a^{\mathrm{ef}}, b^{\mathrm{ef}}){\uparrow}\ .$$

   Compact notation:

$a \cap b$ means $(\mathsf{intersection} :: \mathsf{C}, \mathsf{C}, \mathsf{C})(a, b)$.

8. **Complement**
   Operator: $(\mathsf{complement} :: \mathsf{C}, \mathsf{C})$
   Defining axioms:

$$\forall\, x : \mathsf{C}\ .\ (\mathsf{complement} :: \mathsf{C}, \mathsf{C})(x) \simeq \\ \quad \iota\, y : \mathsf{C}\ .\ \forall\, u : \mathsf{V}\ .\ u \in y \equiv u \notin x.$$

$$a^{\mathrm{ef}}{\uparrow} \supset (\mathsf{complement} :: \mathsf{C}, \mathsf{C})(a^{\mathrm{ef}}){\uparrow}\ .$$

   Compact notation:

$\overline{a}$ means $(\mathsf{complement} :: \mathsf{C}, \mathsf{C})(a)$.

9. **Head**
   Operator: $(\mathsf{head} :: \mathsf{V}, \mathsf{V})$
   Defining axioms:

$$\forall\, w : \mathsf{V}\ .\ (\mathsf{head} :: \mathsf{V}, \mathsf{V})(w) \simeq \\ \quad \iota\, u : \mathsf{V}\ .\ \exists\, v : \mathsf{V}\ .\ w = \langle u, v \rangle.$$

$$a^{\mathrm{ef}}{\uparrow} \supset (\mathsf{head} :: \mathsf{V}, , \mathsf{V})(a^{\mathrm{ef}}){\uparrow}\ .$$

   Compact notation:

$\mathsf{hd}(a)$ means $(\mathsf{head} :: \mathsf{V}, \mathsf{V})(a)$.

10. **Tail**
    Operator: $(\mathsf{tail} :: \mathsf{V}, \mathsf{V})$
    Defining axioms:

$$\forall\, w : \mathsf{V} \,.\, (\mathsf{tail} :: \mathsf{V}, \mathsf{V})(w) \simeq \\ \quad \iota\, v : \mathsf{V} \,.\, \exists\, u : \mathsf{V} \,.\, w = \langle u, v \rangle.$$

$$a^{\mathrm{ef}}{\uparrow} \supset (\mathsf{tail} :: \mathsf{V}, , \mathsf{V})(a^{\mathrm{ef}}){\uparrow}\,.$$

    Compact notation:

    $\mathsf{tl}(a)$ means $(\mathsf{tail} :: \mathsf{V}, \mathsf{V})(a)$.

11. **Append**
    Operator: $(\mathsf{append} :: \mathsf{V}, \mathsf{V}, \mathsf{V})$
    Defining axioms:

$$\forall\, x, y : \mathsf{V} \,.\, (\mathsf{append} :: \mathsf{V}, \mathsf{V}, \mathsf{V})(x, y) \simeq \\ \quad \mathsf{if}(x = \emptyset, y, \langle \mathsf{hd}(x), \mathsf{append}(\mathsf{tl}(x), y) \rangle).$$

$$(a^{\mathrm{ef}}{\uparrow} \vee b^{\mathrm{ef}}{\uparrow}) \supset (\mathsf{append} :: \mathsf{V}, \mathsf{V}, \mathsf{V})(a^{\mathrm{ef}}, b^{\mathrm{ef}}){\uparrow}\,.$$

    Compact notation:

    $a \hat{} b$ means $(\mathsf{append} :: \mathsf{V}, \mathsf{V}, \mathsf{V})(a, b)$.

12. **In List**
    Operator: $(\mathsf{in\text{-}list} :: \mathsf{V}, \mathsf{V}, \mathsf{formula})$
    Defining axioms:

$$\forall\, x, y : \mathsf{V} \,.\, (\mathsf{in\text{-}list} :: \mathsf{V}, \mathsf{V}, \mathsf{formula})(x, y) \equiv \\ \quad x = \mathsf{hd}(y) \vee \mathsf{in\text{-}list}(x, \mathsf{tl}(y)).$$

$$(a^{\mathrm{ef}}{\uparrow} \vee b^{\mathrm{ef}}{\uparrow}) \supset (\mathsf{in\text{-}list} :: \mathsf{V}, \mathsf{V}, \mathsf{formula})(a^{\mathrm{ef}}, b^{\mathrm{ef}}) \equiv \mathsf{F}.$$

    Compact notation:

    $a \in_{\mathrm{li}} b$ means $(\mathsf{in\text{-}list} :: \mathsf{V}, \mathsf{V}, \mathsf{formula})(a, b)$.

13. **List Type**
Operator: $(\mathsf{list\text{-}type} :: \mathsf{type}, \mathsf{type})$
Defining axioms:

$$\neg\mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner \alpha^{\mathrm{ef}} \urcorner) \supset$$
$$\forall\, x : \mathsf{C} \;.\; x \downarrow (\mathsf{list\text{-}type} :: \mathsf{type}, \mathsf{type})(\alpha^{\mathrm{ef}}) \equiv$$
$$x = [\,] \vee \exists\, y : \alpha^{\mathrm{ef}}, z : \mathsf{list\text{-}type}(\alpha^{\mathrm{ef}}) \;.\; x = \langle y, z \rangle.$$

14. **Type Union**
Operator: $(\mathsf{type\text{-}union} :: \mathsf{type}, \mathsf{type}, \mathsf{type})$
Defining axioms:

$$(\neg\mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner \alpha^{\mathrm{ef}} \urcorner) \wedge \neg\mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner \beta^{\mathrm{ef}} \urcorner)) \supset$$
$$\forall\, x : \mathsf{C} \;.\; x \downarrow (\mathsf{type\text{-}union} :: \mathsf{type}, \mathsf{type}, \mathsf{type})(\alpha^{\mathrm{ef}}, \beta^{\mathrm{ef}}) \equiv$$
$$x \downarrow \alpha^{\mathrm{ef}} \vee x \downarrow \beta^{\mathrm{ef}}.$$

Compact notation:

$(\alpha \cup \beta)$ means $(\mathsf{type\text{-}union} :: \mathsf{type}, \mathsf{type}, \mathsf{type})(\alpha, \beta)$.

15. **Type Intersection**
Operator: $(\mathsf{type\text{-}intersection} :: \mathsf{type}, \mathsf{type}, \mathsf{type})$
Defining axioms:

$$(\neg\mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner \alpha^{\mathrm{ef}} \urcorner) \wedge \neg\mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner \beta^{\mathrm{ef}} \urcorner)) \supset$$
$$\forall\, x : \mathsf{C} \;.\; x \downarrow (\mathsf{type\text{-}intersection} :: \mathsf{type}, \mathsf{type}, \mathsf{type})(\alpha^{\mathrm{ef}}, \beta^{\mathrm{ef}}) \equiv$$
$$x \downarrow \alpha^{\mathrm{ef}} \wedge x \downarrow \beta^{\mathrm{ef}}.$$

Compact notation:

$(\alpha \cap \beta)$ means $(\mathsf{type\text{-}intersection} :: \mathsf{type}, \mathsf{type}, \mathsf{type})(\alpha, \beta)$.

16. **Type Complement**
Operator: $(\mathsf{type\text{-}complement} :: \mathsf{type}, \mathsf{type})$
Defining axioms:

$$\neg\mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner \alpha^{\mathrm{ef}} \urcorner) \supset$$
$$\forall\, x : \mathsf{C} \;.\; x \downarrow (\mathsf{type\text{-}complement} :: \mathsf{type}, \mathsf{type})(\alpha^{\mathrm{ef}}) \equiv$$
$$x \uparrow \alpha^{\mathrm{ef}}.$$

Compact notation:

$\overline{\alpha}$ means $(\mathsf{type\text{-}complement} :: \mathsf{type}, \mathsf{type})(\alpha)$.

17. **Type Product**
Operator: $(\mathsf{type\text{-}prod} :: \mathsf{type}, \mathsf{type}, \mathsf{type})$
Defining axioms:

$$(\neg\mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner \alpha^{\mathrm{ef}} \urcorner) \wedge \neg\mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner \beta^{\mathrm{ef}} \urcorner)) \supset$$
$$\forall\, x : \mathsf{C}\ .\ x \downarrow (\mathsf{type\text{-}prod} :: \mathsf{type}, \mathsf{type}, \mathsf{type})(\alpha^{\mathrm{ef}}, \beta^{\mathrm{ef}}) \equiv$$
$$\mathsf{hd}(x) \downarrow \alpha^{\mathrm{ef}} \wedge \mathsf{tl}(x) \downarrow \beta^{\mathrm{ef}}.$$

Compact notation:

$(\alpha \times \beta)$ means $(\mathsf{type\text{-}prod} :: \mathsf{type}, \mathsf{type}, \mathsf{type})(\alpha, \beta)$.

18. **Type Sum**
Operator: $(\mathsf{type\text{-}sum} :: \mathsf{type}, \mathsf{type}, \mathsf{type})$
Defining axioms:

$$(\neg\mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner \alpha^{\mathrm{ef}} \urcorner) \wedge \neg\mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner \beta^{\mathrm{ef}} \urcorner)) \supset$$
$$\forall\, x : \mathsf{C}\ .\ x \downarrow (\mathsf{type\text{-}sum} :: \mathsf{type}, \mathsf{type}, \mathsf{type})(\alpha^{\mathrm{ef}}, \beta^{\mathrm{ef}}) \equiv$$
$$(\mathsf{hd}(x) \downarrow \alpha^{\mathrm{ef}} \wedge \mathsf{tl}(x) = \emptyset) \vee (\mathsf{hd}(x) \downarrow \beta^{\mathrm{ef}} \wedge \mathsf{tl}(x) = \{\emptyset\}).$$

Compact notation:

$(\alpha + \beta)$ means $(\mathsf{type\text{-}sum} :: \mathsf{type}, \mathsf{type}, \mathsf{type})(\alpha, \beta)$.

19. **Binary Relation**
Operator: $(\mathsf{bin\text{-}rel} :: \mathsf{C}, \mathsf{formula})$
Defining axioms:

$$\forall\, x : \mathsf{C}\ .\ (\mathsf{bin\text{-}rel} :: \mathsf{C}, \mathsf{formula})(x) \equiv$$
$$\forall\, w : \mathsf{V}\ .\ w \in x \supset \exists\, u, v : \mathsf{V}\ .\ w = \langle u, v \rangle.$$

$$a^{\mathrm{ef}}\!\uparrow \supset (\mathsf{bin\text{-}rel} :: \mathsf{C}, \mathsf{formula})(a^{\mathrm{ef}}) \equiv \mathsf{F}.$$

20. **Univocal**
Operator: $(\mathsf{univocal} :: \mathsf{C}, \mathsf{formula})$
Defining axioms:

$$\forall\, x : \mathsf{C} \;.\; (\mathsf{univocal} :: \mathsf{C}, \mathsf{formula})(x) \equiv$$
$$\forall\, u, v, v' : \mathsf{V} \;.\; (\langle u, v\rangle \in x \wedge \langle u, v'\rangle \in x) \supset v = v'.$$

$$a^{\mathrm{ef}}{\uparrow} \supset (\mathsf{univocal} :: \mathsf{C}, \mathsf{formula})(a^{\mathrm{ef}}) \equiv \mathsf{F}.$$

21. **Function**
Operator: $(\mathsf{fun} :: \mathsf{C}, \mathsf{formula})$
Defining axioms:

$$\forall\, x : \mathsf{C} \;.\; (\mathsf{fun} :: \mathsf{C}, \mathsf{formula})(x) \equiv$$
$$\mathsf{bin\text{-}rel}(x) \wedge \mathsf{univocal}(x).$$

$$a^{\mathrm{ef}}{\uparrow} \supset (\mathsf{fun} :: \mathsf{C}, \mathsf{formula})(a^{\mathrm{ef}}) \equiv \mathsf{F}.$$

22. **Domain of a Class**
Operator: $(\mathsf{dom} :: \mathsf{C}, \mathsf{C})$
Defining axioms:

$$\forall\, x : \mathsf{C} \;.\; (\mathsf{dom} :: \mathsf{C}, \mathsf{C})(x) \simeq$$
$$\iota\, y : \mathsf{C} \;.\; \forall\, u : \mathsf{V} \;.\; u \in y \equiv (\exists\, v : \mathsf{V} \;.\; \langle u, v\rangle \in x).$$

$$a^{\mathrm{ef}}{\uparrow} \supset (\mathsf{dom} :: \mathsf{C}, \mathsf{C})(a^{\mathrm{ef}}){\uparrow}\,.$$

23. **Range of a Class**
Operator: $(\mathsf{ran} :: \mathsf{C}, \mathsf{C})$
Defining axioms:

$$\forall\, x : \mathsf{C} \;.\; (\mathsf{ran} :: \mathsf{C}, \mathsf{C})(x) \simeq$$
$$\iota\, y : \mathsf{C} \;.\; \forall\, u : \mathsf{V} \;.\; u \in y \equiv (\exists\, v : \mathsf{V} \;.\; \langle v, u\rangle \in x).$$

$$a^{\mathrm{ef}}{\uparrow} \supset (\mathsf{ran} :: \mathsf{C}, \mathsf{C})(a^{\mathrm{ef}}){\uparrow}\,.$$

24. **Total Function on a Class**
Operator: $(\mathsf{total} :: \mathsf{C}, \mathsf{C}, \mathsf{formula})$
Defining axioms:

$$\forall\, f, x : \mathsf{C}\ .\ (\mathsf{total} :: \mathsf{C}, \mathsf{C}, \mathsf{formula})(f, x) \equiv \mathsf{fun}(f) \wedge \mathsf{dom}(f) = x.$$

$$(a^{\mathrm{ef}}\!\uparrow \wedge\, b^{\mathrm{ef}}\!\uparrow) \supset (\mathsf{total} :: \mathsf{C}, \mathsf{C}, \mathsf{formula})(a^{\mathrm{ef}}, b^{\mathrm{ef}}) \equiv \mathsf{F}.$$

25. **Surjective Function on a Class**
Operator: $(\mathsf{surjective} :: \mathsf{C}, \mathsf{C}, \mathsf{formula})$
Defining axioms:

$$\forall\, f, y : \mathsf{C}\ .\ (\mathsf{surjective} :: \mathsf{C}, \mathsf{C}, \mathsf{formula})(f, y) \equiv \mathsf{fun}(f) \wedge \mathsf{ran}(f) = y.$$

$$(a^{\mathrm{ef}}\!\uparrow \wedge\, b^{\mathrm{ef}}\!\uparrow) \supset (\mathsf{surjective} :: \mathsf{C}, \mathsf{C}, \mathsf{formula})(a^{\mathrm{ef}}, b^{\mathrm{ef}}) \equiv \mathsf{F}.$$

26. **Injective Function on a Class**
Operator: $(\mathsf{injective} :: \mathsf{C}, \mathsf{C}, \mathsf{formula})$
Defining axioms:

$$\forall\, f, x : \mathsf{C}\ .\ (\mathsf{injective} :: \mathsf{C}, \mathsf{C}, \mathsf{formula})(f, x) \equiv \mathsf{fun}(f) \wedge \forall u, v : \mathsf{V}\ .\ (u \in x \wedge v \in x \wedge f(u) = f(v)) \supset u = v.$$

$$(a^{\mathrm{ef}}\!\uparrow \wedge\, b^{\mathrm{ef}}\!\uparrow) \supset (\mathsf{injective} :: \mathsf{C}, \mathsf{C}, \mathsf{formula})(a^{\mathrm{ef}}, b^{\mathrm{ef}}) \equiv \mathsf{F}.$$

27. **Bijective Function from a Class to a Class**
Operator: $(\mathsf{bijective} :: \mathsf{C}, \mathsf{C}, \mathsf{C}, \mathsf{formula})$
Defining axioms:

$$\forall\, f, x, y : \mathsf{C}\ .\ (\mathsf{bijective} :: \mathsf{C}, \mathsf{C}, \mathsf{C}, \mathsf{formula})(f, x, y) \equiv \mathsf{fun}(f) \wedge \mathsf{total}(f, x) \wedge \mathsf{surjective}(f, y) \wedge \mathsf{injective}(f, x).$$

$$(a^{\mathrm{ef}}\!\uparrow \wedge\, b^{\mathrm{ef}}\!\uparrow \wedge\, c^{\mathrm{ef}}\!\uparrow) \supset (\mathsf{bijective} :: \mathsf{C}, \mathsf{C}, \mathsf{C}, \mathsf{formula})(a^{\mathrm{ef}}, b^{\mathrm{ef}}, c^{\mathrm{ef}}) \equiv \mathsf{F}.$$

28. **Infinite Class**
Operator: $(\mathsf{infinite} :: \mathsf{C}, \mathsf{formula})$
Defining axioms:

$$\forall\, x : \mathsf{C} \;.\; (\mathsf{infinite} :: \mathsf{C}, \mathsf{formula})(x) \equiv$$
$$\exists\, f, y : \mathsf{C} \;.\; y \subset x \wedge \mathsf{bijective}(f, x, y).$$

$$a^{\mathrm{ef}}{\uparrow} \supset (\mathsf{infinite} :: \mathsf{C}, \mathsf{formula})(a^{\mathrm{ef}}) \equiv \mathsf{F}.$$

29. **Countably Infinite Class**
Operator: $(\mathsf{countably\text{-}infinite} :: \mathsf{C}, \mathsf{formula})$
Defining axioms:

$$\forall\, x : \mathsf{C} \;.\; (\mathsf{countably\text{-}infinite} :: \mathsf{C}, \mathsf{formula})(x) \equiv$$
$$\mathsf{infinite}(x) \wedge (\forall\, y : \mathsf{C} \;.\; \mathsf{infinite}(y) \supset$$
$$(\exists y' : \mathsf{C} \;.\; y' \subseteq y \supset \mathsf{bijective}(f, x, y'))).$$

$$a^{\mathrm{ef}}{\uparrow} \supset (\mathsf{countably\text{-}infinite} :: \mathsf{C}, \mathsf{formula})(a^{\mathrm{ef}}) \equiv \mathsf{F}.$$

30. **Sum Class**
Operator: $(\mathsf{sum} :: \mathsf{C}, \mathsf{C})$
Defining axioms:

$$\forall\, x : \mathsf{C} \;.\; (\mathsf{sum} :: \mathsf{C}, \mathsf{C})(x) \simeq$$
$$\iota\, y : \mathsf{C} \;.\; \forall\, u : \mathsf{V} \;.\; u \in y \equiv (\exists\, v : \mathsf{V} \;.\; u \in v \wedge v \in x).$$

$$a^{\mathrm{ef}}{\uparrow} \supset (\mathsf{sum} :: \mathsf{C}, \mathsf{C})(a^{\mathrm{ef}}){\uparrow}\,.$$

31. **Power Class**
Operator: $(\mathsf{power} :: \mathsf{C}, \mathsf{C})$
Defining axioms:

$$\forall\, x : \mathsf{C} \;.\; (\mathsf{power} :: \mathsf{C}, \mathsf{C})(x) \simeq$$
$$\iota\, y : \mathsf{C} \;.\; \forall\, u : \mathsf{V} \;.\; u \in y \equiv u \subseteq x.$$

$$a^{\mathrm{ef}}{\uparrow} \supset (\mathsf{power} :: \mathsf{C}, \mathsf{C})(a^{\mathrm{ef}}){\uparrow}\,.$$

32. **Type is Class Checker**
Operator: $(\mathsf{type\text{-}is\text{-}class} :: \mathsf{type}, \mathsf{formula})$
Defining axioms:

$$(\neg \mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner \alpha^{\mathrm{ef}} \urcorner) \wedge \neg \mathsf{free\text{-}in}(\ulcorner y \urcorner, \ulcorner \alpha^{\mathrm{ef}} \urcorner)) \supset (\mathsf{type\text{-}is\text{-}class} :: \mathsf{type}, \mathsf{formula})(\alpha^{\mathrm{ef}}) \equiv \exists\, x : \mathsf{C} \,.\, \forall\, y : \mathsf{C} \,.\, y \downarrow \alpha^{\mathrm{ef}} \equiv y \in x.$$

33. **Type is Set Checker**
Operator: $(\mathsf{type\text{-}is\text{-}set} :: \mathsf{type}, \mathsf{formula})$
Defining axioms:

$$(\neg \mathsf{free\text{-}in}(\ulcorner u \urcorner, \ulcorner \alpha^{\mathrm{ef}} \urcorner) \wedge \neg \mathsf{free\text{-}in}(\ulcorner y \urcorner, \ulcorner \alpha^{\mathrm{ef}} \urcorner)) \supset (\mathsf{type\text{-}is\text{-}set} :: \mathsf{type}, \mathsf{formula})(\alpha^{\mathrm{ef}}) \equiv \exists\, u : \mathsf{V} \,.\, \forall\, y : \mathsf{C} \,.\, y \downarrow \alpha^{\mathrm{ef}} \equiv y \in u.$$

34. **Type to Term**
Operator: $(\mathsf{type\text{-}to\text{-}term} :: \mathsf{type}, \mathsf{C})$
Defining axioms:

$$(\neg \mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner \alpha^{\mathrm{ef}} \urcorner) \wedge \neg \mathsf{free\text{-}in}(\ulcorner y \urcorner, \ulcorner \alpha^{\mathrm{ef}} \urcorner)) \supset (\mathsf{type\text{-}to\text{-}term} :: \mathsf{type}, \mathsf{C})(\alpha^{\mathrm{ef}}) \simeq \iota\, x : \mathsf{C} \,.\, \forall\, y : \mathsf{C} \,.\, y \downarrow \alpha^{\mathrm{ef}} \equiv y \in x.$$

Compact notation:

$\mathsf{term}(\alpha)$ means $(\mathsf{type\text{-}to\text{-}term} :: \mathsf{type}, \mathsf{C})(\alpha)$.

35. **Term to Type**
Operator: $(\mathsf{term\text{-}to\text{-}type} :: \mathsf{C}, \mathsf{type})$
Defining axioms:

$$\forall\, x : \mathsf{C} \,.\, \forall\, y : \mathsf{C} \,.\, y \downarrow (\mathsf{term\text{-}to\text{-}type} :: \mathsf{C}, \mathsf{type})(x) \equiv y \in x.$$

$$a^{\mathrm{ef}} \uparrow \supset (\mathsf{term\text{-}to\text{-}type} :: \mathsf{C}, \mathsf{type})(a^{\mathrm{ef}}) =_{\mathrm{ty}} \mathsf{C}.$$

Compact notation:

$\mathsf{type}(a)$ means $(\mathsf{term\text{-}to\text{-}term} :: \mathsf{C}, \mathsf{type})(a)$.

36. **Power Type**
Operator: $(\mathsf{power\text{-}type} :: \mathsf{type}, \mathsf{type})$
Defining axioms:

$$(\mathsf{power\text{-}type} :: \mathsf{type}, \mathsf{type})(\alpha^{\mathrm{ef}}) =_{\mathrm{ty}} \mathsf{type}(\mathsf{power}(\mathsf{term}(\alpha^{\mathrm{ef}}))).$$

5.3 Syntactic Operators

1. **Proper Expression Checker**
Operator: $(\mathsf{is\text{-}p\text{-}expr} :: \mathsf{E}, \mathsf{formula})$
Defining axioms:

$$\forall\, e : \mathsf{E} \,.\, (\mathsf{is\text{-}p\text{-}expr} :: \mathsf{E}, \mathsf{formula})(e) \equiv$$
$$e \downarrow \mathsf{E}_{\mathrm{op}} \vee e \downarrow \mathsf{E}_{\mathrm{ty}} \vee e \downarrow \mathsf{E}_{\mathrm{te}} \vee e \downarrow \mathsf{E}_{\mathrm{fo}}.$$

$$a^{\mathrm{ef}}\!\uparrow \supset (\mathsf{is\text{-}p\text{-}expr} :: \mathsf{E}, \mathsf{formula})(a^{\mathrm{ef}}) \equiv \mathsf{F}.$$

Note: Checkers for the different sorts of proper expressions are defined in a similar way: is-op, is-type, is-term, is-term-of-type, and is-formula.

2. **First Component Selector for a Proper Expression**
Operator: $(\mathsf{1st\text{-}comp} :: \mathsf{E}, \mathsf{E})$
Defining axioms:

$$\forall\, e : \mathsf{E} \,.\, (\mathsf{1st\text{-}comp} :: \mathsf{E}, \mathsf{E})(e) \simeq$$
$$\mathsf{if}(\mathsf{is\text{-}p\text{-}expr}(e), \mathsf{hd}(\mathsf{tl}(e)), \bot_{\mathsf{C}}).$$

$$a^{\mathrm{ef}}\!\uparrow \supset (\mathsf{1st\text{-}comp} :: \mathsf{E}, \mathsf{E})(a^{\mathrm{ef}})\!\uparrow .$$

Note: The second, third, fourth, ... component selectors for proper expressions are defined in a similar way: 2nd-comp, 3rd-comp, 4th-comp, ...

3. **Operator Application Checker**
Operator: $(\mathsf{is\text{-}op\text{-}app} :: \mathsf{E}, \mathsf{formula})$
Defining axioms:

$$\forall\, e : \mathsf{E} \,.\, (\mathsf{is\text{-}op\text{-}app} :: \mathsf{E}, \mathsf{formula})(e) \equiv$$
$$\mathsf{is\text{-}p\text{-}expr}(e) \wedge \mathsf{hd}(e) = \ulcorner \mathsf{op\text{-}app} \urcorner.$$

$$a^{\mathrm{ef}}\!\uparrow \supset (\mathsf{is\text{-}op\text{-}app} :: \mathsf{E}, \mathsf{formula})(a^{\mathrm{ef}}) \equiv \mathsf{F}.$$

Note: Checkers for the other remaining 11 proper expression categories are defined in a similar way: is-var, is-type-app, is-dep-fun-type, is-fun-app, is-fun-abs, is-if, is-exists, is-def-des, is-indef-des, is-quote, is-eval.

4. **Disjunction Checker**
   Operator: $(\mathsf{is\text{-}or} :: \mathsf{E}, \mathsf{formula})$
   Defining axioms:

$$\forall\, e : \mathsf{E}\ .\ (\mathsf{is\text{-}or} :: \mathsf{E}, \mathsf{formula})(e) \equiv$$
$$\quad \mathsf{is\text{-}op\text{-}app}(e) \wedge \mathsf{1st\text{-}comp}(\mathsf{1st\text{-}comp}(e)) = \ulcorner \mathsf{or} \urcorner.$$

$$a^{\mathrm{ef}}\!\uparrow \supset (\mathsf{is\text{-}or} :: \mathsf{E}, \mathsf{formula})(a^{\mathrm{ef}}) \equiv \mathsf{F}.$$

   Note: Checkers for other kinds of operator applications are defined in a similar way: is-in, is-type-equal, is-union, etc.

5. **First Argument Selector for an Operator Application**
   Operator: $(\mathsf{1st\text{-}arg} :: \mathsf{E}, \mathsf{E})$
   Defining axioms:

$$\forall\, e : \mathsf{E}\ .\ (\mathsf{1st\text{-}arg} :: \mathsf{E}, \mathsf{E})(e) \simeq$$
$$\quad \mathsf{if}(\mathsf{is\text{-}op\text{-}app}(e), \mathsf{2nd\text{-}comp}(e), \bot_{\mathsf{C}}).$$

$$a^{\mathrm{ef}}\!\uparrow \supset (\mathsf{1st\text{-}arg} :: \mathsf{E}, \mathsf{E})(a^{\mathrm{ef}})\!\uparrow .$$

   Note: The second, third, fourth, ... argument selectors for operator application are defined in a similar way: 2nd-arg, 3rd-arg, 4th-arg, ...

6. **Variable Binder Checker**
   Operator: $(\mathsf{is\text{-}binder} :: \mathsf{E}, \mathsf{formula})$
   Defining axioms:

$$\forall\, e : \mathsf{E}\ .\ (\mathsf{is\text{-}binder} :: \mathsf{E}, \mathsf{formula})(e) \equiv$$
$$\quad \mathsf{is\text{-}dep\text{-}fun\text{-}type}(e) \vee \mathsf{is\text{-}fun\text{-}abs}(e) \vee \mathsf{is\text{-}exists}(e) \vee$$
$$\quad \mathsf{is\text{-}def\text{-}des}(e) \vee \mathsf{is\text{-}indef\text{-}des}(e).$$

$$a^{\mathrm{ef}}\!\uparrow \supset (\mathsf{is\text{-}binder} :: \mathsf{E}, \mathsf{formula})(a^{\mathrm{ef}}) \equiv \mathsf{F}.$$

7. **Variable Selector for Variable Binder**
   Operator: $(\mathsf{binder\text{-}var} :: \mathsf{E}, \mathsf{E})$
   Defining axioms:

$$\forall\, e : \mathsf{E}\ .\ (\mathsf{binder\text{-}var} :: \mathsf{E}, \mathsf{E})(e) \simeq \mathsf{if}(\mathsf{is\text{-}binder}(e), \mathsf{1st\text{-}comp}(e), \bot_{\mathsf{C}}).$$

$$a^{\mathrm{ef}}{\uparrow} \supset (\mathsf{binder\text{-}var} :: \mathsf{E}, \mathsf{E})(a^{\mathrm{ef}}){\uparrow}\ .$$

   Note: Selectors for a binder name and a binder body are defined in a similar way: binder-name and binder-body.

8. **Function Redex Checker**
   Operator: $(\mathsf{is\text{-}fun\text{-}redex} :: \mathsf{E}, \mathsf{formula})$
   Defining axioms:

$$\forall\, e : \mathsf{E}\ .\ (\mathsf{is\text{-}fun\text{-}redex} :: \mathsf{E}, \mathsf{formula})(e) \equiv \mathsf{is\text{-}fun\text{-}app}(e) \wedge \mathsf{is\text{-}fun\text{-}abs}(\mathsf{1st\text{-}comp}(e)).$$

$$a^{\mathrm{ef}}{\uparrow} \supset (\mathsf{is\text{-}fun\text{-}redex} :: \mathsf{E}, \mathsf{formula})(a^{\mathrm{ef}}) \equiv \mathsf{F}.$$

   Note: A dependent function type redex checker named is-dep-fun-type-redex is defined in a similar way.

9. **Redex Checker**
   Operator: $(\mathsf{is\text{-}redex} :: \mathsf{E}, \mathsf{formula})$
   Defining axioms:

$$\forall\, e : \mathsf{E}\ .\ (\mathsf{is\text{-}redex} :: \mathsf{E}, \mathsf{formula})(e) \equiv \mathsf{is\text{-}dep\text{-}fun\text{-}type\text{-}redex}(e) \vee \mathsf{is\text{-}fun\text{-}redex}(e).$$

$$a^{\mathrm{ef}}{\uparrow} \supset (\mathsf{is\text{-}redex} :: \mathsf{E}, \mathsf{formula})(a^{\mathrm{ef}}) \equiv \mathsf{F}.$$

10. **Variable Selector for a Redex**
    Operator: $(\mathsf{redex\text{-}var} :: \mathsf{E}, \mathsf{E})$
    Defining axioms:

$$\forall\, e : \mathsf{E}\ .\ (\mathsf{redex\text{-}var} :: \mathsf{E}, \mathsf{E})(e) \simeq \mathsf{if}(\mathsf{is\text{-}redex}(e), \mathsf{binder\text{-}var}(\mathsf{1st\text{-}comp}(e)), \bot_{\mathsf{C}}).$$

$$a^{\mathrm{ef}}\!\uparrow \supset (\mathsf{redex\text{-}var} :: \mathsf{E}, \mathsf{E})(a^{\mathrm{ef}})\!\uparrow .$$

Note: Body and argument selectors for a redex named redex-body and redex-arg are defined in a similar way.

11. **Variable Similarity**
Operator: $(\mathsf{var\text{-}sim} :: \mathsf{E}, \mathsf{E}, \mathsf{formula})$
Defining axioms:

$$\forall\, e_1, e_2 : \mathsf{E} \;.\; (\mathsf{var\text{-}sim} :: \mathsf{E}, \mathsf{E}, \mathsf{formula})(e_1, e_2) \equiv$$
$$\quad \mathsf{is\text{-}var}(e_1) \wedge \mathsf{is\text{-}var}(e_2) \wedge \mathsf{1st\text{-}comp}(e_1) = \mathsf{1st\text{-}comp}(e_2).$$
$$(a^{\mathrm{ef}}\!\uparrow \wedge\, b^{\mathrm{ef}}\!\uparrow) \supset (\mathsf{var\text{-}sim} :: \mathsf{E}, \mathsf{E}, \mathsf{formula})(a^{\mathrm{ef}}, b^{\mathrm{ef}}) \equiv \mathsf{F}.$$

Compact notation:

$e_1 \sim e_2$ means $(\mathsf{var\text{-}sim} :: \mathsf{E}, \mathsf{E}, \mathsf{formula})(e_1, e_2)$.
$e_1 \not\sim e_2$ means $\neg(e_1 \sim e_2)$.

12. **Eval-Free Checker**
Operator: $(\mathsf{is\text{-}eval\text{-}free} :: \mathsf{E}, \mathsf{formula})$
Defining axioms:

$$\forall\, e : \mathsf{E} \;.\; (\mathsf{is\text{-}eval\text{-}free} :: \mathsf{E}, \mathsf{formula})(e) \equiv$$
$$\quad \mathsf{is\text{-}quote}(e) \vee$$
$$\quad (e \downarrow \mathsf{E}_{\mathrm{sy}} \wedge e \neq \ulcorner \mathsf{eval} \urcorner) \vee$$
$$\quad e = [\,] \vee$$
$$\quad (\mathsf{is\text{-}eval\text{-}free}(\mathsf{hd}(e)) \wedge \mathsf{is\text{-}eval\text{-}free}(\mathsf{tl}(e))).$$
$$a^{\mathrm{ef}}\!\uparrow \supset (\mathsf{is\text{-}eval\text{-}free} :: \mathsf{E}, \mathsf{formula})(a^{\mathrm{ef}}) \equiv \mathsf{F}.$$

13. **Coercion to a Type Construction**
Operator: $(\mathsf{coerce\text{-}to\text{-}type} :: \mathsf{E}, \mathsf{E}_{\mathrm{ty}})$
Defining axioms:

$$\forall\, e_1 : \mathsf{E} \;.\; (\mathsf{coerce\text{-}to\text{-}type} :: \mathsf{E}, \mathsf{E}_{\mathrm{ty}})(e_1) \equiv$$
$$\quad \iota\, e_2 : \mathsf{E}_{\mathrm{ty}} \;.\; e_1 =_{\mathrm{ty}} e_2.$$
$$a^{\mathrm{ef}}\!\uparrow \supset (\mathsf{coerce\text{-}to\text{-}type} :: \mathsf{E}, \mathsf{E}_{\mathrm{ty}})(a^{\mathrm{ef}})\!\uparrow .$$

Note: The operators coerce-to-term and coerce-to-formula are defined in a similar way.

14. **Nominal Type of a Term**
Operator: $(\mathsf{nominal\text{-}type} :: \mathsf{E}_{\mathrm{te}}, \mathsf{E}_{\mathrm{ty}})$
Defining axioms:

$$\forall\, e_1 : \mathsf{E}_{\mathrm{te}} \,.\, (\mathsf{nominal\text{-}type} :: \mathsf{E}_{\mathrm{te}}, \mathsf{E}_{\mathrm{ty}})(e_1) \equiv \iota\, e_2 : \mathsf{E}_{\mathrm{ty}} \,.\, e_1 \downarrow \mathsf{E}_{\mathrm{te}}^{e_2}.$$

$$a^{\mathrm{ef}}\!\uparrow \supset (\mathsf{nominal\text{-}type} :: \mathsf{E}_{\mathrm{te}}, \mathsf{E}_{\mathrm{ty}})(a^{\mathrm{ef}})\!\uparrow .$$

15. **Operator Name Strictness Checker**
Operator: $(\mathsf{is\text{-}strict\text{-}op\text{-}name} :: \mathsf{E}_{\mathrm{on}}, \mathsf{formula})$
Defining axioms:

$$\begin{array}{l}\forall\, o : \mathsf{E}_{\mathrm{on}} \,.\, (\mathsf{is\text{-}strict\text{-}op\text{-}name} :: \mathsf{E}_{\mathrm{on}}, \mathsf{formula})(o) \equiv \\ \quad \forall\, e : \mathsf{E} \,. \\ \qquad (\mathsf{is\text{-}op\text{-}app}(e) \wedge \\ \qquad\ o = \mathsf{1st\text{-}comp}(\mathsf{1st\text{-}comp}(e)) \wedge \\ \qquad\ \ulcorner \bot_{\mathcal{C}} \urcorner \in_{\mathrm{li}} \mathsf{tl}(\mathsf{tl}(e))) \\ \qquad\quad \supset \\ \qquad ((\mathsf{is\text{-}type}(e) \supset [\![e]\!]_{\mathrm{ty}} = \mathsf{C}) \wedge \\ \qquad\ (\mathsf{is\text{-}term}(e) \supset [\![e]\!]_{\mathrm{te}}\!\uparrow) \wedge \\ \qquad\ (\mathsf{is\text{-}formula}(e) \supset [\![e]\!]_{\mathrm{fo}} = \mathsf{F})).\end{array}$$

$$a^{\mathrm{ef}}\!\uparrow \supset (\mathsf{is\text{-}strict\text{-}op\text{-}name} :: \mathsf{E}_{\mathrm{on}}, \mathsf{formula})(a^{\mathrm{ef}}) \equiv \mathsf{F}.$$

Compact notation:

$\mathsf{strict}(o)$ means $(\mathsf{is\text{-}strict\text{-}op\text{-}name} :: \mathsf{E}_{\mathrm{on}}, \mathsf{formula})(o)$.

Remark 5.3.1 All the defining axioms given in this section are eval-free.

5.4 Another Notational Definition

Using some of the operators defined in this section, we give a notational definition for evaluation relativized to a language:

Relativized Evaluation

(eval, a, k, b) means

$$\mathsf{if}(a{\downarrow}\,(\mathsf{E}_{\mathrm{ty},b} \cup \mathsf{E}_{\mathrm{te},b} \cup \mathsf{E}_{\mathrm{fo},b}), [\![a]\!]_k, [\![\bot_{\mathsf{C}}]\!]_k).$$

Compact notation:

$[\![a]\!]_{k,b}$ means (eval, a, k, b).

Notice that $[\![a]\!]_{k,\ell}$ and $[\![a]\!]_k$ are logically equivalent.

6 Substitution

In this section we will define the operators needed for substitution of a term for the free occurrences of a variable and prove that they have their intended meanings. Substitution on eval-free expressions is very similar to substitution on first-order formulas, but substitution on non-eval-free expressions is tricky. There are two reasons for this.

We will illustrate the first reason with an example. Consider the formula $a = [\![(e : \mathsf{E})]\!]$. The variable $(e : \mathsf{E})$ is certainly free in the formula. However, if the value of the variable is $\ulcorner(y : \mathsf{C})\urcorner$, then the formula is equivalent to $a = (y : \mathsf{C})$, and so the $(y : \mathsf{C})$ can also be said to be free in the formula. This example shows that some free occurrences of a variable in an expression may not be syntactically visible. This significantly complicates substitution.

The second reason has to do with the evaluation of a substitution. A substitution maps a quoted expression $\ulcorner e\urcorner$ to a new quoted expression $\ulcorner e'\urcorner$. The latter will usually be the argument to an evaluation $[\![\ulcorner e'\urcorner]\!]_k$ so that the value of e' can be expressed. If e' is not eval-free, then the evaluation will be undefined and, as a result, substitution will not work as desired on non-eval-free expressions. To avoid this problem, evaluations must be "cleansed" from the result of a substitution.

6.1 Substitution Operators

We define now the operators related to the substitution of a term for the free occurrences of a variable. Each operator will be defined only for quotations. As a result, the defining axioms will be an infinite set organized into a finite number of formula schemas.

1. **Good Evaluation Arguments**
   Operator: $(\mathsf{gea} :: \mathsf{E}, \mathsf{E}, \mathsf{formula})$
   Defining axioms:

   $\neg(\mathsf{gea} :: \mathsf{E}, \mathsf{E}, \mathsf{formula})(\ulcorner e_1 \urcorner, \ulcorner e_2 \urcorner)$

   where e_1 is a non-eval-free expression, e_1 is a type and e_2 is not type, e_1 is a term and e_2 is not a type, or e_1 is a formula and e_2 is not $\mathsf{formula}$.

   $(\mathsf{gea} :: \mathsf{E}, \mathsf{E}, \mathsf{formula})(\ulcorner \alpha^{\mathrm{ef}} \urcorner, \ulcorner \mathsf{type} \urcorner)$.

   $(\mathsf{gea} :: \mathsf{E}, \mathsf{E}, \mathsf{formula})(\ulcorner a^{\mathrm{ef}} \urcorner, \ulcorner \alpha \urcorner)$.

   $(\mathsf{gea} :: \mathsf{E}, \mathsf{E}, \mathsf{formula})(\ulcorner A^{\mathrm{ef}} \urcorner, \ulcorner \mathsf{formula} \urcorner)$.

2. **Free Variable Occurrence in an Expression**
   Operator: $(\mathsf{free\text{-}in} :: \mathsf{E}, \mathsf{E}, \mathsf{formula})$
   Defining axioms:

   $\neg(\mathsf{free\text{-}in} :: \mathsf{E}, \mathsf{E}, \mathsf{formula})(\ulcorner e_1 \urcorner, \ulcorner e_2 \urcorner)$

   where e_1 is not a symbol or e_2 is an improper expression.

   $(\mathsf{free\text{-}in} :: \mathsf{E}, \mathsf{E}, \mathsf{formula})(\ulcorner x \urcorner, \ulcorner (o :: k_1, \ldots, k_{n+1}) \urcorner) \equiv$
   $\mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner k_1 \urcorner) \vee \cdots \vee \mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner k_{n+1} \urcorner)$

   where $(o :: k_1, \ldots, k_{n+1})$ is proper and $n \geq 0$.

   $(\mathsf{free\text{-}in} :: \mathsf{E}, \mathsf{E}, \mathsf{formula})(\ulcorner x \urcorner, \ulcorner O(e_1, \ldots, e_n) \urcorner) \equiv$
   $\mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner O \urcorner) \vee$
   $\mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner e_1 \urcorner) \vee \cdots \vee \mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner e_n \urcorner)$

   where $O(e_1, \ldots, e_n)$ is proper and $n \geq 0$.

   $(\mathsf{free\text{-}in} :: \mathsf{E}, \mathsf{E}, \mathsf{formula})(\ulcorner x \urcorner, \ulcorner (x : \alpha) \urcorner)$.

   $(\mathsf{free\text{-}in} :: \mathsf{E}, \mathsf{E}, \mathsf{formula})(\ulcorner x \urcorner, \ulcorner (y : \alpha) \urcorner) \equiv$
   $\mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner \alpha \urcorner)$

   where $x \neq y$.

$(\mathsf{free\text{-}in} :: \mathsf{E}, \mathsf{E}, \mathsf{formula})(\ulcorner x \urcorner, \ulcorner (\star\, x : \alpha\ .\ e) \urcorner) \equiv$
$\quad \mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner \alpha \urcorner)$

$\quad$ where $(\star\, x : \alpha\ .\ e)$ is proper and $\star$ is Λ, λ, $\exists$, ι, or ϵ.

$(\mathsf{free\text{-}in} :: \mathsf{E}, \mathsf{E}, \mathsf{formula})(\ulcorner x \urcorner, \ulcorner (\star\, y : \alpha\ .\ e) \urcorner) \equiv$
$\quad \mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner \alpha \urcorner) \vee \mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner e \urcorner)$

$\quad$ where $x \neq y$, $(\star\, x : \alpha\ .\ e)$ is proper, and $\star$ is Λ, λ, $\exists$, ι, or ϵ.

$(\mathsf{free\text{-}in} :: \mathsf{E}, \mathsf{E}, \mathsf{formula})(\ulcorner x \urcorner, \ulcorner \alpha(a) \urcorner) \equiv$
$\quad \mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner \alpha \urcorner) \vee \mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner a \urcorner)$.

$(\mathsf{free\text{-}in} :: \mathsf{E}, \mathsf{E}, \mathsf{formula})(\ulcorner x \urcorner, \ulcorner f(a) \urcorner) \equiv$
$\quad \mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner f \urcorner) \vee \mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner a \urcorner)$.

$(\mathsf{free\text{-}in} :: \mathsf{E}, \mathsf{E}, \mathsf{formula})(\ulcorner x \urcorner, \ulcorner \mathsf{if}(A, b, c) \urcorner) \equiv$
$\quad \mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner A \urcorner) \vee \mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner b \urcorner) \vee \mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner c \urcorner)$.

$\neg(\mathsf{free\text{-}in} :: \mathsf{E}, \mathsf{E}, \mathsf{formula})(\ulcorner x \urcorner, \ulcorner\ulcorner e \urcorner\urcorner)$

$\quad$ where e is any expression.

$(\mathsf{free\text{-}in} :: \mathsf{E}, \mathsf{E}, \mathsf{formula})(\ulcorner x \urcorner, \ulcorner [\![a]\!]_k \urcorner) \equiv$
$\quad \mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner a \urcorner) \vee$
$\quad \mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner k \urcorner) \vee$
$\quad \mathsf{free\text{-}in}(\ulcorner x \urcorner, a)$.

$(a^{\mathrm{ef}}\!\uparrow \vee\, b^{\mathrm{ef}}\!\uparrow) \supset \neg(\mathsf{free\text{-}in} :: \mathsf{E}, \mathsf{E}, \mathsf{formula})(a^{\mathrm{ef}}, b^{\mathrm{ef}})$.

Note: For an evaluation (eval, a, k), a variable may be free in the term a, in the kind k when k is a type, and in the expression that the evaluation represents.

3. **Syntactically Closed Proper Expression**
Operator: $(\mathsf{syn\text{-}closed} :: \mathsf{E}, \mathsf{formula})$
Defining axioms:

$\forall\, e : \mathsf{E}\ .\ (\mathsf{syn\text{-}closed} :: \mathsf{E}, \mathsf{formula})(e) \equiv$
$\quad \mathsf{is\text{-}p\text{-}expr}(e) \wedge \forall\, e' : \mathsf{E}_{\mathrm{sy}}\ .\ \neg\mathsf{free\text{-}in}(e', e)$.

4. **Cleanse Eval Symbols from an Expression**
   Operator: $(\mathsf{cleanse} :: \mathsf{E}, \mathsf{E})$
   Defining axioms:

$$(\mathsf{cleanse} :: \mathsf{E}, \mathsf{E})(\ulcorner e \urcorner) = \ulcorner e \urcorner$$

where e is an improper expression.

$$(\mathsf{cleanse} :: \mathsf{E}, \mathsf{E})(\ulcorner (o :: k_1, \ldots, k_{n+1}) \urcorner) = \ulcorner (o :: \lfloor \widehat{k}_1 \rfloor, \ldots, \lfloor \widehat{k}_{n+1} \rfloor) \urcorner$$

where $(o :: k_1, \ldots, k_{n+1})$ is proper, $n \geq 0$, and $\widehat{k}_i = \mathsf{cleanse}(\ulcorner k_i \urcorner)$ for all i with $1 \leq i \leq n+1$.

$$(\mathsf{cleanse} :: \mathsf{E}, \mathsf{E})(\ulcorner O(e_1, \ldots, e_n) \urcorner) = \ulcorner \lfloor \widehat{O} \rfloor (\lfloor \widehat{e}_1 \rfloor, \ldots, \lfloor \widehat{e}_n \rfloor) \urcorner$$

where $O(e_1, \ldots, e_n)$ is proper, $n \geq 0$,
$\widehat{O} = \mathsf{cleanse}(\ulcorner O \urcorner)$, and
$\widehat{e}_i = \mathsf{cleanse}(\ulcorner e_i \urcorner)$ for all i with $1 \leq i \leq n$.

$$(\mathsf{cleanse} :: \mathsf{E}, \mathsf{E})(\ulcorner (x : \alpha) \urcorner) = \ulcorner (x : \lfloor \widehat{\alpha} \rfloor) \urcorner$$

where $\widehat{\alpha} = \mathsf{cleanse}(\ulcorner \alpha \urcorner)$.

$$(\mathsf{cleanse} :: \mathsf{E}, \mathsf{E})(\ulcorner (\star x : \alpha \,.\, e) \urcorner) = \ulcorner (\star x : \lfloor \widehat{\alpha} \rfloor \,.\, \lfloor \widehat{e} \rfloor) \urcorner$$

where $(\star x : \alpha \,.\, e)$ is proper; $\star$ is Λ, λ, $\exists$, ι, or ϵ;
$\widehat{\alpha} = \mathsf{cleanse}(\ulcorner \alpha \urcorner)$; and $\widehat{e} = \mathsf{cleanse}(\ulcorner e \urcorner)$.

$$(\mathsf{cleanse} :: \mathsf{E}, \mathsf{E})(\ulcorner \alpha(b) \urcorner) = \ulcorner \lfloor \widehat{\alpha} \rfloor (\lfloor \widehat{b} \rfloor) \urcorner$$

where $\widehat{\alpha} = \mathsf{cleanse}(\ulcorner \alpha \urcorner)$ and $\widehat{b} = \mathsf{cleanse}(\ulcorner b \urcorner)$.

$(\mathsf{cleanse} :: \mathsf{E}, \mathsf{E})(\ulcorner f(b) \urcorner) =$
$\quad \ulcorner \lfloor \widehat{f} \rfloor (\lfloor \widehat{b} \rfloor) \urcorner$
$\quad$ where $\widehat{f} = \mathsf{cleanse}(\ulcorner f \urcorner)$ and $\widehat{b} = \mathsf{cleanse}(\ulcorner b \urcorner)$.

$(\mathsf{cleanse} :: \mathsf{E}, \mathsf{E})(\ulcorner \mathsf{if}(A, b, c) \urcorner) =$
$\quad \ulcorner \mathsf{if}(\lfloor \widehat{A} \rfloor, \lfloor \widehat{b} \rfloor, \lfloor \widehat{c} \rfloor) \urcorner$
$\quad$ where $\widehat{A} = \mathsf{cleanse}(\ulcorner A \urcorner)$,
$\quad \widehat{b} = \mathsf{cleanse}(\ulcorner b \urcorner)$, and
$\quad \widehat{c} = \mathsf{cleanse}(\ulcorner c \urcorner)$.

$(\mathsf{cleanse} :: \mathsf{E}, \mathsf{E})(\ulcorner \ulcorner e \urcorner \urcorner) =$
$\quad \ulcorner \ulcorner e \urcorner \urcorner$
$\quad$ where e is any expression.

$(\mathsf{cleanse} :: \mathsf{E}, \mathsf{E})(\ulcorner [\![a]\!]_{\mathrm{ty}} \urcorner) =$
$\quad \mathsf{if}(\mathsf{syn\text{-}closed}(\widehat{a}), w, \bot_{\mathsf{C}})$
$\quad$ where $\widehat{a} = \mathsf{cleanse}(\ulcorner a \urcorner)$,
$\quad \widehat{a}' = \mathsf{coerce\text{-}to\text{-}type}([\![\widehat{a}]\!]_{\mathrm{te}})$, and
$\quad w = \ulcorner \mathsf{if}(\mathsf{gea}(a, \ulcorner \mathsf{type} \urcorner), \lfloor \widehat{a}' \rfloor, \mathsf{C}) \urcorner$.

$(\mathsf{cleanse} :: \mathsf{E}, \mathsf{E})(\ulcorner [\![a]\!]_{\alpha} \urcorner) =$
$\quad \mathsf{if}(\mathsf{syn\text{-}closed}(\widehat{a}), w, \bot_{\mathsf{C}})$
$\quad$ where $\widehat{a} = \mathsf{cleanse}(\ulcorner a \urcorner)$,
$\quad \widehat{a}' = \mathsf{coerce\text{-}to\text{-}term}([\![\widehat{a}]\!]_{\mathrm{te}})$,
$\quad \widehat{\alpha} = \mathsf{cleanse}(\ulcorner \alpha \urcorner)$, and
$\quad w = \ulcorner \mathsf{if}(\mathsf{gea}(a, \ulcorner \alpha \urcorner) \wedge \lfloor \widehat{a}' \rfloor \downarrow \lfloor \widehat{\alpha} \rfloor, \lfloor \widehat{a}' \rfloor, \bot_{\mathsf{C}}) \urcorner$.

$(\mathsf{cleanse} :: \mathsf{E}, \mathsf{E})(\ulcorner [\![a]\!]_{\mathrm{fo}} \urcorner) =$
$\quad \mathsf{if}(\mathsf{syn\text{-}closed}(\widehat{a}), w, \bot_{\mathsf{C}})$
$\quad$ where $\widehat{a} = \mathsf{cleanse}(\ulcorner a \urcorner)$,
$\quad \widehat{a}' = \mathsf{coerce\text{-}to\text{-}formula}([\![\widehat{a}]\!]_{\mathrm{te}})$, and
$\quad w = \ulcorner \mathsf{if}(\mathsf{gea}(a, \ulcorner \mathsf{formula} \urcorner), \lfloor \widehat{a}' \rfloor, \mathsf{F}) \urcorner$.

$a^{\mathrm{ef}}\!\uparrow \supset (\mathsf{cleanse} :: \mathsf{E}, \mathsf{E})(a^{\mathrm{ef}})\!\uparrow$.

5. **Substitution for a Variable in an Expression**
   Operator: $(\mathsf{sub} :: \mathsf{E}, \mathsf{E}, \mathsf{E}, \mathsf{E})$
   Defining axioms:

$$(\mathsf{sub} :: \mathsf{E}, \mathsf{E}, \mathsf{E}, \mathsf{E})(\ulcorner e_1 \urcorner, \ulcorner e_2 \urcorner, \ulcorner e_3 \urcorner) = \ulcorner e_3 \urcorner$$

where e_1 is not a term, e_2 is not a symbol, or e_3 is an improper expression.

$$(\mathsf{sub} :: \mathsf{E}, \mathsf{E}, \mathsf{E}, \mathsf{E})(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner (o :: k_1, \ldots, k_{n+1}) \urcorner) = \ulcorner (o :: \lfloor \widehat{k}_1 \rfloor, \ldots, \lfloor \widehat{k}_{n+1} \rfloor) \urcorner$$

where $(o :: k_1, \ldots, k_{n+1})$ is proper, $n \geq 0$, and $\widehat{k}_i = \mathsf{sub}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner k_i \urcorner)$ for all i with $1 \leq i \leq n+1$.

$$(\mathsf{sub} :: \mathsf{E}, \mathsf{E}, \mathsf{E}, \mathsf{E})(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner O(e_1, \ldots, e_n) \urcorner) = \ulcorner \lfloor \widehat{O} \rfloor (\lfloor \widehat{e}_1 \rfloor, \ldots, \lfloor \widehat{e}_n \rfloor) \urcorner$$

where $O(e_1, \ldots, e_n)$ is proper, $n \geq 0$,
$\widehat{O} = \mathsf{sub}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner O \urcorner)$, and
$\widehat{e}_i = \mathsf{sub}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner e_i \urcorner)$ for all i with $1 \leq i \leq n$.

$$(\mathsf{sub} :: \mathsf{E}, \mathsf{E}, \mathsf{E}, \mathsf{E})(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner (x : \alpha) \urcorner) = \ulcorner \mathsf{if}(a \downarrow \lfloor \widehat{\alpha} \rfloor, \lfloor \widehat{a} \rfloor, \bot_{\mathsf{C}}) \urcorner$$

where $\widehat{\alpha} = \mathsf{sub}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner \alpha \urcorner)$ and $\widehat{a} = \mathsf{cleanse}(\ulcorner a \urcorner)$.

$$(\mathsf{sub} :: \mathsf{E}, \mathsf{E}, \mathsf{E}, \mathsf{E})(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner (y : \alpha) \urcorner) = \ulcorner (y : \lfloor \widehat{\alpha} \rfloor) \urcorner$$

where $x \neq y$ and $\widehat{\alpha} = \mathsf{sub}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner \alpha \urcorner)$.

$$(\mathsf{sub} :: \mathsf{E}, \mathsf{E}, \mathsf{E}, \mathsf{E})(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner (\star x : \alpha \,.\, e) \urcorner) = \ulcorner (\star x : \lfloor \widehat{\alpha} \rfloor \,.\, \lfloor \widehat{e} \rfloor) \urcorner$$

where $(\star x : \alpha \,.\, e)$ is proper; $\star$ is Λ, λ, $\exists$, ι, or ϵ;
$\widehat{\alpha} = \mathsf{sub}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner \alpha \urcorner)$; and $\widehat{e} = \mathsf{cleanse}(\ulcorner e \urcorner)$.

$(\mathsf{sub} :: \mathsf{E}, \mathsf{E}, \mathsf{E}, \mathsf{E})(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner (\star y : \alpha \ . \ e) \urcorner) =$
$\ulcorner (\star y : \lfloor \widehat{\alpha} \rfloor \ . \ \lfloor \widehat{e} \rfloor) \urcorner$

where $x \neq y$; $(\star y : \alpha \ . \ e)$ is proper; $\star$ is Λ, λ, $\exists$, ι, or ϵ; $\widehat{\alpha} = \mathsf{sub}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner \alpha \urcorner)$; and $\widehat{e} = \mathsf{sub}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner e \urcorner)$.

$(\mathsf{sub} :: \mathsf{E}, \mathsf{E}, \mathsf{E}, \mathsf{E})(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner \alpha(b) \urcorner) =$
$\ulcorner \lfloor \widehat{\alpha} \rfloor (\lfloor \widehat{b} \rfloor) \urcorner$

where $\widehat{\alpha} = \mathsf{sub}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner \alpha \urcorner)$ and $\widehat{b} = \mathsf{sub}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner b \urcorner)$.

$(\mathsf{sub} :: \mathsf{E}, \mathsf{E}, \mathsf{E}, \mathsf{E})(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner f(b) \urcorner) =$
$\ulcorner \lfloor \widehat{f} \rfloor (\lfloor \widehat{b} \rfloor) \urcorner$

where $\widehat{f} = \mathsf{sub}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner f \urcorner)$ and $\widehat{b} = \mathsf{sub}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner b \urcorner)$.

$(\mathsf{sub} :: \mathsf{E}, \mathsf{E}, \mathsf{E}, \mathsf{E})(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner \mathsf{if}(A, b, c) \urcorner) =$
$\ulcorner \mathsf{if}(\lfloor \widehat{A} \rfloor, \lfloor \widehat{b} \rfloor, \lfloor \widehat{c} \rfloor) \urcorner$

where $\widehat{A} = \mathsf{sub}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner A \urcorner)$, $\widehat{b} = \mathsf{sub}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner b \urcorner)$, and $\widehat{c} = \mathsf{sub}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner c \urcorner)$.

$(\mathsf{sub} :: \mathsf{E}, \mathsf{E}, \mathsf{E}, \mathsf{E})(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner\ulcorner e \urcorner\urcorner) =$
$\ulcorner\ulcorner e \urcorner\urcorner$

where e is any expression.

$(\mathsf{sub} :: \mathsf{E}, \mathsf{E}, \mathsf{E}, \mathsf{E})(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner [\![b]\!]_{\mathrm{ty}} \urcorner) =$
$\mathsf{if}(\mathsf{syn\text{-}closed}(\widehat{b}), w, \bot_{\mathsf{C}})$

where $\widehat{b} = \mathsf{sub}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner b \urcorner)$,
$\widehat{b'} = \mathsf{coerce\text{-}to\text{-}type}(\mathsf{sub}(\ulcorner a \urcorner, \ulcorner x \urcorner, [\![\widehat{b}]\!]_{\mathrm{te}}))$,
$w = \ulcorner \mathsf{if}(\mathsf{gea}(\lfloor \widehat{b} \rfloor, \ulcorner \mathsf{type} \urcorner), \lfloor \widehat{b'} \rfloor, \mathsf{C}) \urcorner$.

$(\mathsf{sub} :: \mathsf{E}, \mathsf{E}, \mathsf{E}, \mathsf{E})(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner [\![b]\!]_{\alpha} \urcorner) =$
$\mathsf{if}(\mathsf{syn\text{-}closed}(\widehat{b}), w, \bot_{\mathsf{C}})$

where $\widehat{b} = \mathsf{sub}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner b \urcorner)$,
$\widehat{b'} = \mathsf{coerce\text{-}to\text{-}term}(\mathsf{sub}(\ulcorner a \urcorner, \ulcorner x \urcorner, [\![\widehat{b}]\!]_{\mathrm{te}}))$,
$\widehat{\alpha} = \mathsf{sub}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner \alpha \urcorner)$, and
$w = \ulcorner \mathsf{if}(\mathsf{gea}(\lfloor \widehat{b} \rfloor, \ulcorner \alpha \urcorner) \wedge \lfloor \widehat{b'} \rfloor \downarrow \lfloor \widehat{\alpha} \rfloor, \lfloor \widehat{b'} \rfloor, \bot_{\mathsf{C}}) \urcorner$.

$(\mathsf{sub} :: \mathsf{E}, \mathsf{E}, \mathsf{E}, \mathsf{E})(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner [\![b]\!]_{\mathrm{fo}} \urcorner) =$
$\quad \mathsf{if}(\mathsf{syn\text{-}closed}(\widehat{b}), w, \bot_{\mathsf{C}})$
$\quad$ where $\widehat{b} = \mathsf{sub}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner b \urcorner)$,
$\quad \widehat{b}' = \mathsf{coerce\text{-}to\text{-}formula}(\mathsf{sub}(\ulcorner a \urcorner, \ulcorner x \urcorner, [\![\widehat{b}]\!]_{\mathrm{te}}))$,
$\quad w = \ulcorner \mathsf{if}(\mathsf{gea}(\lfloor \widehat{b} \rfloor, \ulcorner \mathsf{formula} \urcorner), \lfloor \widehat{b}' \rfloor, \mathsf{F}) \urcorner$.

$(a^{\mathrm{ef}}\!\uparrow \vee b^{\mathrm{ef}}\!\uparrow \vee c^{\mathrm{ef}}\!\uparrow) \supset (\mathsf{sub} :: \mathsf{E}, \mathsf{E}, \mathsf{E}, \mathsf{E})(a^{\mathrm{ef}}, b^{\mathrm{ef}}, c^{\mathrm{ef}})\!\uparrow .$

Note: For an evaluation (eval, a, k), a substitution for a variable is performed on a to obtain a' and on k when k is a type to obtain k' and then on the expression represented by the new evaluation resulting from these substitutions.

6. **Free for a Variable in an Expression**
Operator: $(\mathsf{free\text{-}for} :: \mathsf{E}, \mathsf{E}, \mathsf{E}, \mathsf{formula})$
Defining axioms:

$(\mathsf{free\text{-}for} :: \mathsf{E}, \mathsf{E}, \mathsf{E}, \mathsf{formula})(\ulcorner e_1 \urcorner, \ulcorner e_2 \urcorner, \ulcorner e_3 \urcorner)$

where e_1 is not a term, e_2 is not a symbol, or e_3 is an improper expression.

$(\mathsf{free\text{-}for} :: \mathsf{E}, \mathsf{E}, \mathsf{E}, \mathsf{formula})(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner (o :: k_1, \ldots, k_{n+1}) \urcorner) \equiv$
$\quad \mathsf{free\text{-}for}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner k_1 \urcorner) \wedge \cdots \wedge \mathsf{free\text{-}for}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner k_{n+1} \urcorner)$

where $(o :: k_1, \ldots, k_{n+1})$ is proper and $n \geq 0$.

$(\mathsf{free\text{-}for} :: \mathsf{E}, \mathsf{E}, \mathsf{E}, \mathsf{formula})(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner O(e_1, \ldots, e_n) \urcorner) \equiv$
$\quad \mathsf{free\text{-}for}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner O \urcorner) \wedge$
$\quad \mathsf{free\text{-}for}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner e_1 \urcorner) \wedge \cdots \wedge \mathsf{free\text{-}for}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner e_n \urcorner)$

where $O(e_1, \ldots, e_n)$ is proper and $n \geq 0$.

$(\mathsf{free\text{-}for} :: \mathsf{E}, \mathsf{E}, \mathsf{E}, \mathsf{formula})(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner (y : \alpha) \urcorner) \equiv$
$\quad \mathsf{free\text{-}for}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner \alpha \urcorner)$.

$(\mathsf{free\text{-}for} :: \mathsf{E}, \mathsf{E}, \mathsf{E}, \mathsf{formula})(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner (\star\, x : \alpha\ .\ e) \urcorner) \equiv$
$\quad \mathsf{free\text{-}for}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner \alpha \urcorner)$

$\quad$ where $(\star\, x : \alpha\ .\ e)$ is proper and $\star$ is Λ, λ, $\exists$, ι, or ϵ.

$(\mathsf{free\text{-}for} :: \mathsf{E}, \mathsf{E}, \mathsf{E}, \mathsf{formula})(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner \star\, y : \alpha\ .\ e \urcorner) \equiv$
$\quad \mathsf{free\text{-}for}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner \alpha \urcorner) \wedge$
$\quad (\neg \mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner e \urcorner) \vee \neg \mathsf{free\text{-}in}(\ulcorner y \urcorner, \ulcorner a \urcorner)) \wedge$
$\quad \mathsf{free\text{-}for}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner e \urcorner)$

$\quad$ where $x \neq y$, $(\star\, y : \alpha\ .\ e)$ is proper, and $\star$ is Λ, λ, $\exists$, ι, or ϵ.

$(\mathsf{free\text{-}for} :: \mathsf{E}, \mathsf{E}, \mathsf{E}, \mathsf{formula})(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner \alpha(b) \urcorner) \equiv$
$\quad \mathsf{free\text{-}for}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner \alpha \urcorner) \wedge \mathsf{free\text{-}for}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner b \urcorner)$.

$(\mathsf{free\text{-}for} :: \mathsf{E}, \mathsf{E}, \mathsf{E}, \mathsf{formula})(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner f(b) \urcorner) \equiv$
$\quad \mathsf{free\text{-}for}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner f \urcorner) \wedge \mathsf{free\text{-}for}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner b \urcorner)$.

$(\mathsf{free\text{-}for} :: \mathsf{E}, \mathsf{E}, \mathsf{E}, \mathsf{formula})(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner \mathsf{if}(A, b, c) \urcorner) \equiv$
$\quad \mathsf{free\text{-}for}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner A \urcorner) \wedge$
$\quad \mathsf{free\text{-}for}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner b \urcorner) \wedge$
$\quad \mathsf{free\text{-}for}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner c \urcorner)$.

$(\mathsf{free\text{-}for} :: \mathsf{E}, \mathsf{E}, \mathsf{E}, \mathsf{formula})(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner\ulcorner e \urcorner\urcorner)$

$\quad$ where e is any expression.

$(\mathsf{free\text{-}for} :: \mathsf{E}, \mathsf{E}, \mathsf{E}, \mathsf{formula})(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner [\![b]\!]_k \urcorner) \equiv$
$\quad \mathsf{free\text{-}for}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner b \urcorner) \wedge$
$\quad \mathsf{free\text{-}for}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner k \urcorner) \wedge$
$\quad \mathsf{free\text{-}for}(\ulcorner a \urcorner, \ulcorner x \urcorner, [\![\widehat{b}]\!]_{\mathrm{te}})$.

$\quad$ where $\widehat{b} = \mathsf{sub}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner b \urcorner)$.

$(a^{\mathrm{ef}}\!\uparrow \vee b^{\mathrm{ef}}\!\uparrow \vee c^{\mathrm{ef}}\!\uparrow) \supset$
$\quad \neg(\mathsf{free\text{-}for} :: \mathsf{E}, \mathsf{E}, \mathsf{E}, \mathsf{formula})(a^{\mathrm{ef}}, b^{\mathrm{ef}}, c^{\mathrm{ef}})$.

Remark 6.1.1 All the defining axioms given in this subsection are eval-free except for those involving the operators free-in and free-for applied to quoted evaluations.

6.2 Kernel, Normal, and Eval-Free Normal Theories

The *kernel language* of Chiron is the language $L_{\text{ker}} = (\mathcal{O}, \theta)$ where:

1. $o \in \mathcal{O}$ iff o is a built-in operator name of Chiron or a name of an operator defined in section 5 or in the previous subsection.

2. For all $o \in \mathcal{O}$, $\theta(o)$ is the signature form assigned to o.

A language L is *normal* if $L_{\text{ker}} \leq L$.

Let L be a normal language. The *kernel theory over* L, written T^L_{ker}, is the theory $(L, \Gamma^L_{\text{ker}})$ where $A \in \Gamma^L_{\text{ker}}$ iff A is a one of the defining axioms (with respect to the L) for an operator defined in section 5 or the previous section.

Let $T = (L, \Gamma)$ be a theory. T is *normal* if $T^L_{\text{ker}} \leq T$. The *eval-free subtheory* of T, written $\mathsf{eval\text{-}free}(T)$, is the theory $T = (L, \Gamma')$ where $\Gamma' = \{A \in \Gamma \mid A \text{ is eval-free}\}$. T is *eval-free normal* if $T = \mathsf{eval\text{-}free}(T')$ for some normal theory T'.

Let a be a term, x be a symbol, and e be a proper expression of L. We say that x is *free in* e if $\mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner e \urcorner)$ is valid in T^L_{ker}, and a is *free for* x *in* e if $\mathsf{free\text{-}for}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner e \urcorner)$ is valid in T^L_{ker}. Similarly, we say x is *not free in* e if $\neg\mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner e \urcorner)$ is valid in T^L_{ker}, and a is *not free for* x *in* e if $\neg\mathsf{free\text{-}for}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner e \urcorner)$ is valid in T^L_{ker}. We also say e is *syntactically closed* if $\mathsf{syn\text{-}closed}(\ulcorner e \urcorner)$ is valid in T^L_{ker}.

Proposition 6.2.1 *Every quotation is syntactically closed.*

6.3 Evaluation and Quasiquotation Lemmas

The lemmas in this subsection are facts about evaluations that are needed for the substitution lemmas given in the next subsection. Let $=_k$ be $=_{\text{ty}}$ if k is a type, $\simeq$ if k is a type, and $\equiv$ if k is a $\mathsf{formula}$.

Lemma 6.3.1 (Good Evaluation Arguments) *Let $T = (L, \Gamma)$ be a normal theory, α be a type, a and b be terms, and A be a formula of L.*

1. $T \models \mathsf{gea}(\ulcorner \alpha \urcorner, \ulcorner \mathsf{type} \urcorner) \supset [\![\ulcorner \alpha \urcorner]\!]_{\text{ty}} =_{\text{ty}} \alpha$.

2. $T \models \neg\mathsf{gea}(b, \ulcorner \mathsf{type} \urcorner) \supset [\![b]\!]_{\text{ty}} =_{\text{ty}} \mathsf{C}$.

3. $T \models \mathsf{gea}(\ulcorner a \urcorner, \ulcorner \alpha \urcorner) \supset [\![\ulcorner a \urcorner]\!]_{\alpha} \simeq \mathsf{if}(a \downarrow \alpha, a, \bot_{\mathsf{C}})$.

4. $T \models \neg\mathsf{gea}(b, \ulcorner \alpha \urcorner) \supset [\![b]\!]_{\alpha} \simeq \bot_{\mathsf{C}}$.

5. $T \models \mathsf{gea}(\ulcorner A \urcorner, \ulcorner \mathsf{formula} \urcorner) \supset [\![\ulcorner A \urcorner]\!]_{\mathrm{fo}} \equiv A$.

6. $T \models \neg\mathsf{gea}(b, \ulcorner \mathsf{formula} \urcorner) \supset [\![b]\!]_{\mathrm{fo}} \equiv \mathsf{F}$.

Proof Follows from the definition of the standard valuation on evaluations. □

Lemma 6.3.2 (Evaluation of Quasiquotations) *Let M be a standard model of a normal theory $T = (L, \Gamma)$ and a be a term of type E of L that denotes a type, term, or formula of kind $k = \mathsf{type}$, C, or $\mathsf{formula}$, respectively, such that $M \models \mathsf{gea}(a, \ulcorner k \urcorner)$.*

1. *Let $a = \ulcorner (o :: e_1, \ldots, e_{n+1})(\lfloor c_1 \rfloor, \ldots, \lfloor c_n \rfloor) \urcorner$ denote an operator application where:*

   a. *e_i is type, $\mathsf{formula}$, or $\lfloor b_i \rfloor$ for all i with $1 \leq i \leq n+1$.*

   b. *c_i denotes a type, term, or formula of kind $k_i = \mathsf{type}$, C, or $\mathsf{formula}$, respectively, for all i with $1 \leq i \leq n$.*

   Then

   $$M \models [\![a]\!]_k =_k (o :: \overline{e_1}, \ldots, \overline{e_{n+1}})([\![c_1]\!]_{k_i}, \ldots, [\![c_n]\!]_{k_n})$$

   where

   $$\overline{e_i} = \begin{cases} [\![b_i]\!]_{\mathrm{ty}} & \text{if } e_i = \lfloor b_i \rfloor \\ e_i & \text{if } e_i = \mathsf{type} \text{ or } \mathsf{formula}. \end{cases}$$

2. *If $a = \ulcorner (x : \lfloor b \rfloor) \urcorner$ denotes a variable, then*

   $$M \models [\![a]\!]_{\mathrm{te}} \simeq (x : [\![b]\!]_{\mathrm{ty}}).$$

3. *If $a = \ulcorner (\star x : \lfloor b_1 \rfloor \,.\, \lfloor b_2 \rfloor) \urcorner$ denotes a variable binder where $\star$ is Λ, λ, $\exists$, ι, or ϵ and b_2 is semantically closed in M with respect to x, then*

   $$M \models [\![a]\!]_k =_k (\star x : [\![b_1]\!]_{\mathrm{ty}} \,.\, [\![b_2]\!]_{\mathrm{fo}}).$$

4. *If $a = \ulcorner \lfloor b_1 \rfloor (\lfloor b_2 \rfloor) \urcorner$ denotes a type application, then*

   $$M \models [\![a]\!]_{\mathrm{ty}} =_{\mathrm{ty}} [\![b_1]\!]_{\mathrm{ty}}([\![b_2]\!]_{\mathrm{te}}).$$

5. *If* $a = \ulcorner \lfloor b_1 \rfloor (\lfloor b_2 \rfloor) \urcorner$ *denotes a function application, then*

$$M \models [\![a]\!]_{\mathrm{te}} \simeq [\![b_1]\!]_{\mathrm{te}}([\![b_2]\!]_{\mathrm{te}}).$$

6. *If* $a = \ulcorner \mathsf{if}(\lfloor b_1 \rfloor, \lfloor b_2 \rfloor, \lfloor b_3 \rfloor) \urcorner$ *denotes a conditional term, then*

$$M \models [\![a]\!]_{\mathrm{te}} \simeq \mathsf{if}([\![b_1]\!]_{\mathrm{fo}}, [\![b_2]\!]_{\mathrm{te}}, [\![b_3]\!]_{\mathrm{te}}).$$

7. *If* $a = \ulcorner [\![\lfloor b \rfloor]\!]_{\mathrm{ty}} \urcorner$ *denotes a type evaluation, then*

$$M \models [\![a]\!]_{\mathrm{ty}} =_{\mathrm{ty}} [\![[\![b]\!]_{\mathrm{te}}]\!]_{\mathrm{ty}}.$$

8. *If* $a = \ulcorner [\![\lfloor b_1 \rfloor]\!]_{\lfloor b_2 \rfloor} \urcorner$ *denotes a term evaluation, then*

$$M \models [\![a]\!]_{\mathrm{te}} \simeq [\![[\![b_1]\!]_{\mathrm{te}}]\!]_{[\![b_2]\!]_{ty}}.$$

9. *If* $a = \ulcorner [\![\lfloor b \rfloor]\!]_{\mathrm{fo}} \urcorner$ *denotes a formula evaluation, then*

$$M \models [\![a]\!]_{\mathrm{fo}} \equiv [\![[\![b]\!]_{\mathrm{te}}]\!]_{\mathrm{fo}}.$$

Proof Let $M = (S, V)$ be a standard model of T and $\varphi \in \mathsf{assign}(S)$.

Parts 1, 4–9 Similar to part 2.

Part 2 We must show that $V_\varphi([\![a]\!]_{\mathrm{te}}) = V_\varphi((x : [\![b]\!]_{\mathrm{ty}}))$.

$$\begin{aligned}
& V_\varphi([\![a]\!]_{\mathrm{te}}) \\
= \; & V_\varphi([\![\ulcorner (x : \lfloor b \rfloor) \urcorner]\!]_{\mathrm{te}}) \\
= \; & V_\varphi(H^{-1}(V_\varphi(\ulcorner (x : \lfloor b \rfloor) \urcorner))) \\
= \; & V_\varphi(H^{-1}(V_\varphi([\ulcorner \mathsf{var} \urcorner, \ulcorner x \urcorner, b]))) \\
= \; & V_\varphi((H^{-1}(V_\varphi(\ulcorner \mathsf{var} \urcorner)), H^{-1}(V_\varphi(\ulcorner x \urcorner)), H^{-1}(V_\varphi(b)))) \\
= \; & V_\varphi((H^{-1}(H(\mathsf{var})), H^{-1}(H(x)), H^{-1}(V_\varphi(b)))) \\
= \; & V_\varphi((\mathsf{var}, x, [\![b]\!]_{\mathrm{ty}})) \\
= \; & V_\varphi((x : [\![b]\!]_{\mathrm{ty}}))
\end{aligned}$$

The third line is by the definition of V on evaluations and $M \models \mathsf{gea}(a, \ulcorner \mathsf{C} \urcorner)$; the fourth is by the definition of a quasiquotation; the fifth is by the definitions of H and $[a_1, \ldots, a_n]$; the sixth is by the definition of V on quotations; and the seventh is by the definition of V on evaluations and the fact that $M \models \mathsf{gea}(a, \ulcorner \mathsf{C} \urcorner)$ implies $M \models \mathsf{gea}(b, \ulcorner \mathsf{type} \urcorner)$.

Part 3 We must show that $V_\varphi([\![a]\!]_k) = V_\varphi((\star x : [\![b_1]\!]_{\mathrm{ty}} \,.\, [\![b_2]\!]_{\mathrm{fo}}))$ assuming b_2 is semantically closed in M with respect to x.

$$
\begin{array}{ll}
 & V_\varphi([\![a]\!]_k) \\
= & V_\varphi([\![\ulcorner(\star x : \lfloor b_1 \rfloor \,.\, \lfloor b_2 \rfloor)\urcorner]\!]_k) \\
= & V_\varphi(H^{-1}(V_\varphi(\ulcorner(\star x : \lfloor b_1 \rfloor \,.\, \lfloor b_2 \rfloor)\urcorner))) \\
= & V_\varphi(H^{-1}(V_\varphi([\ulcorner\star\urcorner, [\ulcorner\mathsf{var}\urcorner, \ulcorner x\urcorner, b_1], b_2]))) \\
= & V_\varphi((H^{-1}(V_\varphi(\ulcorner\star\urcorner)), \\
 & \quad (H^{-1}(V_\varphi(\ulcorner\mathsf{var}\urcorner)), H^{-1}(V_\varphi(\ulcorner x\urcorner)), H^{-1}(V_\varphi(b_1))), \\
 & \quad H^{-1}(V_\varphi(b_2)))) \\
= & V_\varphi((H^{-1}(H(\star)), \\
 & \quad (H^{-1}(H(\mathsf{var})), H^{-1}(H(x)), H^{-1}(V_\varphi(b_1))), \\
 & \quad H^{-1}(V_\varphi(b_2)))) \\
= & V_\varphi((\star, (\mathsf{var}, x, [\![b_1]\!]_{\mathrm{ty}}), [\![b_2]\!]_{\mathrm{fo}})) \\
= & V_\varphi((\star x : [\![b_1]\!]_{\mathrm{ty}} \,.\, [\![b_2]\!]_{\mathrm{fo}}))
\end{array}
$$

The third line is by the definition of V on evaluations and $M \models \mathsf{gea}(a, \ulcorner k\urcorner)$; the fourth is by the definition of a quasiquotation; the fifth is by the definitions of H and $[a_1, \ldots, a_n]$; the sixth is by the definition of V on quotations; and the seventh is by the definition of V on evaluations, the fact that $M \models \mathsf{gea}(a, \ulcorner k\urcorner)$ implies $M \models \mathsf{gea}(b_1, \ulcorner\mathsf{type}\urcorner)$ and $M \models \mathsf{gea}(b_2, \ulcorner\mathsf{formula}\urcorner)$, and the fact b_2 is semantically closed in M with respect to x. □

Lemma 6.3.3 *Let $M = (S, V)$ be a standard model of a normal theory $T = (L, \Gamma)$ and q be a quasiquotation of L whose set of evaluated components is $\{\lfloor a_1 \rfloor, \ldots, \lfloor a_n \rfloor\}$. If a_i is semantically closed in M for all i with $1 \leq i \leq n$, then q is semantically closed in M.*

Proof Let $\varphi, \varphi' \in \mathsf{assign}(S)$. We must show that $V_\varphi(q) = V_{\varphi'}(q)$. Our proof is by induction on the length of q. If q is a quotation, then q is obviously semantically closed, and so we may assume q is not a quotation. Let $q = \ulcorner(m_1, \ldots, m_k)\urcorner$ where $k \geq 1$.

$$
\begin{array}{ll}
 & V_\varphi(q) \\
= & V_\varphi(\ulcorner(m_1, \ldots, m_k)\urcorner) \\
= & V_\varphi([\ulcorner m_1\urcorner, \ldots, \ulcorner m_k\urcorner]) \\
= & [V_\varphi(\ulcorner m_1\urcorner), \ldots, V_\varphi(\ulcorner m_k\urcorner)] \\
= & [V_{\varphi'}(\ulcorner m_1\urcorner), \ldots, V_{\varphi'}(\ulcorner m_k\urcorner)]
\end{array}
$$

$$
\begin{aligned}
&= V_{\varphi'}([\ulcorner m_1 \urcorner, \ldots, \ulcorner m_k \urcorner]) \\
&= V_{\varphi'}(\ulcorner (m_1, \ldots, m_k) \urcorner) \\
&= V_{\varphi'}(q)
\end{aligned}
$$

The third line is by the definition of a quasiquotation; the fourth is by the definition of $[a_1, \ldots, a_n]$; and the fifth is by the following argument. If $\ulcorner m_i \urcorner$ is a quotation, then $\ulcorner m_i \urcorner$ is semantically closed, and so $V_\varphi(\ulcorner m_i \urcorner) = V_{\varphi'}(\ulcorner m_i \urcorner)$. If m_i is an evaluated component $\lfloor a_j \rfloor$, then $V_\varphi(\ulcorner m_i \urcorner) = V_\varphi(a_j) = V_{\varphi'}(a_j) = V_{\varphi'}(\ulcorner m_i \urcorner)$ since a_j is semantically closed in T by hypothesis. If $\ulcorner m_i \urcorner$ is not a quotation and m_i is not an evaluated component, then $V_\varphi(\ulcorner m_i \urcorner) = V_{\varphi'}(\ulcorner m_i \urcorner)$ by the induction hypothesis. □

6.4 Substitution Lemmas

The next several lemmas show that the operators specified in the previous subsection have their intended meanings. Let $k[e]$ be type if e is a type, C if e is a term, and $\mathsf{formula}$ if e is a formula.

Lemma 6.4.1 (Eval-Free) *Let L be a normal language and a be an eval-free term, x be a symbol, and e be an eval-free proper expression of L. Let $T = \mathsf{eval\text{-}free}(T^L_{\mathrm{ker}})$.*

1. *Either $T \models \mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner e \urcorner)$ or $T \models \neg\mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner e \urcorner)$. (That is, either x is free in e or x is not free in e.)*
2. *Either $T \models \mathsf{free\text{-}for}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner e \urcorner)$ or $T \models \neg\mathsf{free\text{-}for}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner e \urcorner)$. That is, either a is free for x in e or a is not free for x in e.*
3. *$T \models \mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner e \urcorner)$ for at most finitely many symbols x.*
4. *$T \models \mathsf{cleanse}(\ulcorner e \urcorner) = \ulcorner e \urcorner$ and thus $T \models \mathsf{cleanse}(\ulcorner e \urcorner)\!\downarrow$.*
5. *$T \models \mathsf{sub}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner e \urcorner) = \ulcorner e' \urcorner$ for some eval-free proper expression e' and thus $T \models \mathsf{sub}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner e \urcorner)\!\downarrow$.*
6. *If $T \models \neg\mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner e \urcorner)$, then $T \models \mathsf{sub}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner e \urcorner) = \ulcorner e \urcorner$.*

Proof

Parts 1–2, 4–5 Follows immediately by induction on the length of e.

Part 3 Follows from the fact that x is free in e iff e contains a subexpression of the form $(x : \alpha)$.

Part 6 Let $S(\ulcorner e' \urcorner)$ mean $\mathsf{sub}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner e' \urcorner)$. Assume

$$T \models \neg \mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner e \urcorner) \quad [\text{designated } H(\ulcorner e \urcorner)].$$

We must show that

$$T \models S(\ulcorner e \urcorner) = \ulcorner e \urcorner \quad [\text{designated } C(\ulcorner e \urcorner)].$$

Our proof is by induction on the length of e. There are 10 cases corresponding to the 10 formula schemas used to define $S(\ulcorner e \urcorner)$ when e is eval-free proper expression.

Case 1. $e = (o :: k_1, \ldots, k_{n+1})$. $H(\ulcorner e \urcorner)$ implies $H(\ulcorner k_i \urcorner)$ for all i with $1 \leq i \leq n+1$ and $\mathbf{type}_L[k_i]$. From these hypotheses, $C(\ulcorner k_i \urcorner)$ follows for all i with $1 \leq i \leq n+1$ and $\mathbf{type}_L[k_i]$ from the induction hypothesis. $C(\ulcorner k_i \urcorner)$ also holds for all i with $1 \leq i \leq n+1$ and $k_i = \mathsf{type}$ or $\mathsf{formula}$. From these conclusions, $C(\ulcorner e \urcorner)$ follows by the definition of sub.

Cases 2, 4, 6–9. Similar to case 1.

case 3. $e = (x : \alpha)$. The hypothesis $H(\ulcorner e \urcorner)$ is false in this case.

case 5. $e = (\star x : \alpha \ . \ e')$ where $\star$ is Λ, λ, $\exists$, ι, or ϵ. $H(\ulcorner e \urcorner)$ implies $H(\ulcorner \alpha \urcorner)$. From this hypothesis, $C(\ulcorner \alpha \urcorner)$ follows from the induction hypothesis. Then $C(\ulcorner e \urcorner)$ follows from $C(\ulcorner \alpha \urcorner)$ by part 4 of this lemma and the definition of sub.

case 10. $e = \ulcorner e' \urcorner$. $C(\ulcorner e \urcorner)$ is always true in this case by the definition of sub.

□

Remark 6.4.2 Lemma 6.4.1, with T^L_{ker} used in place of $\mathsf{eval\text{-}free}(T^L_{\mathrm{ker}})$, does not hold for all non-eval-free expressions. For instance, the example in subsection 7.2 exhibits a non-eval free expression for which part 1 of this lemma does not hold.

A *universal closure* of a formula A is a formula

$$\forall\, x_1, \ldots, x_n : \mathsf{C} \ . \ A$$

where $n \geq 0$ such that x is free in A iff $x \in \{x_1, \ldots, x_n\}$ and x is not free in A iff $x \notin \{x_1, \ldots, x_n\}$.

Lemma 6.4.3 *Every universal closure is syntactically closed.*

Proof By the definitions of free-in, syntactically closed, and universal closure. □

Remark 6.4.4 By virtue of parts 1 and 3 of Lemma 6.4.1, universal closures always exist for eval-free formulas, and by the example in subsection 7.2, may not exist for non-eval-free formulas.

Lemma 6.4.5 (Free Variable) *Let $M = (S, V)$ be a standard model of a normal theory $T = (L, \Gamma)$, x be a symbol, and e be a proper expression of L.*

1. *If $M \models \neg\mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner e \urcorner)$, then $V_\varphi(e) = V_{\varphi[x \mapsto d]}(e)$ for all $\varphi \in \mathsf{assign}(S)$ and $d \in D_\mathrm{c}$. (That is, if $M \models \neg\mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner e \urcorner)$, e is semantically closed in M with respect to x.)*
2. *Let $X = \{x \in \mathcal{S} \mid M \models \neg\mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner e \urcorner)\}$. Then $V_\varphi(e) = V_{\varphi'}(e)$ for all $\varphi, \varphi' \in \mathsf{assign}(S)$ such that $\varphi(x) = \varphi'(x)$ whenever $x \notin X$.*

Proof

Part 1 We will show that, if

$M \models \neg\mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner e \urcorner)$ [designated $H(\ulcorner e \urcorner)$],

then

$V_\varphi(e) = V_{\varphi[x \mapsto d]}(e)$ for all $\varphi \in \mathsf{assign}(S)$ and $d \in D_\mathrm{c}$ [designated $C(\ulcorner e \urcorner)$].

Our proof is by induction on the complexity of e. There are 11 cases corresponding to the 11 formula schemas used to define $\mathsf{free\text{-}in}(\ulcorner e_1 \urcorner, \ulcorner e_2 \urcorner))$ when e_1 is a symbol and e_2 is a proper expression.

case 1: $e = (o :: k_1, \ldots, k_{n+1})$. By the definition of free-in, $H(\ulcorner e \urcorner)$ implies $H(\ulcorner k_i \urcorner)$ for all i with $1 \leq i \leq n+1$ and $\mathbf{type}_L[k_i]$. By the induction hypothesis, this implies $C(\ulcorner k_i \urcorner)$ for all i with $1 \leq i \leq n+1$ and $\mathbf{type}_L[k_i]$. Therefore, $C(\ulcorner e \urcorner)$ holds since $V_\varphi(e)$ is defined in terms of $V_\varphi(k_i)$ for all i with $1 \leq i \leq n+1$ and $\mathbf{type}_L[k_i]$.

case 2: $e = O(e_1, \ldots, e_n)$. By the definition of free-in, $H(\ulcorner e \urcorner)$ implies $H(\ulcorner O \urcorner)$ and $H(\ulcorner e_i \urcorner)$ for all i with $1 \leq i \leq n$. By the induction hypothesis, this implies $C(\ulcorner O \urcorner)$ and $C(\ulcorner e_i \urcorner)$ for all i with $1 \leq i \leq n$. Therefore, $C(\ulcorner e \urcorner)$ holds since $V_\varphi(e)$ is defined in terms of $V_\varphi(O)$ and $V_\varphi(e_i)$ for all i with $1 \leq i \leq n$.

case 3: $e = (x : \alpha)$. By the definition of free-in, $H(\ulcorner e \urcorner)$ is false. Therefore, the lemma is true for this case.

case 4: $e = (y : \alpha)$ where $x \neq y$. By the definition of free-in, $H(\ulcorner e \urcorner)$ implies $H(\ulcorner \alpha \urcorner)$. By the induction hypothesis, this implies $C(\ulcorner \alpha \urcorner)$. Therefore, $C(\ulcorner e \urcorner)$ holds since $V_\varphi(e)$ is defined in terms of $\varphi(y)$ and $V_\varphi(\alpha)$.

case 5: $e = (\star x : \alpha \ . \ e')$ and $\star$ is Λ, λ, $\exists$, ι, or ϵ. By the definition of free-in, $H(\ulcorner e \urcorner)$ implies $H(\ulcorner \alpha \urcorner)$. By the induction hypothesis, this implies $C(\ulcorner \alpha \urcorner)$. It is obvious that

$$V_{\varphi[x \mapsto d']}(e') = V_{\varphi[x \mapsto d][x \mapsto d']}(e')$$

for all $\varphi \in \mathsf{assign}(S)$ and $d, d' \in D_{\mathrm{c}}$. Therefore, $C(\ulcorner e \urcorner)$ holds since $V_\varphi(e)$ is defined in terms of $V_\varphi(\alpha)$ and $V_{\varphi[x \mapsto d']}(e')$ where $d' \in D_{\mathrm{c}}$.

case 6: $e = (\star y : \alpha \ . \ e')$, $x \neq y$, and $\star$ is Λ, λ, $\exists$, ι, or ϵ. By the definition of free-in, $H(\ulcorner e \urcorner)$ implies $H(\ulcorner \alpha \urcorner)$ and $H(\ulcorner e' \urcorner)$. By the induction hypothesis, this implies $C(\ulcorner \alpha \urcorner)$ and $C(\ulcorner e' \urcorner)$. The latter implies

$$V_{\varphi[y \mapsto d']}(e') = V_{\varphi[y \mapsto d'][x \mapsto d]}(e') = V_{\varphi[x \mapsto d][y \mapsto d']}(e')$$

for all $\varphi \in \mathsf{assign}(S)$ and $d, d' \in D_{\mathrm{c}}$. (This is the place in the proof where the full strength of the induction hypothesis is needed.) Therefore, $C(\ulcorner e \urcorner)$ holds since $V_\varphi(e)$ is defined in terms of $V_\varphi(\alpha)$ and $V_{\varphi[y \mapsto d']}(e')$ where $d' \in D_{\mathrm{c}}$.

case 7: $e = \alpha(a)$. Similar to case 4.

Case 8: $e = f(a)$. Similar to case 4.

Case 9: $e = \mathsf{if}(A, b, c)$. Similar to case 4.

Case 10: $e = \ulcorner e' \urcorner$. $C(\ulcorner e \urcorner)$ follows immediately since $V_\varphi(e)$ does not depend on φ.

Case 11: $e = [\![a]\!]_k$. By the definition of free-in, $H(\ulcorner e \urcorner)$ implies (1) $H(\ulcorner a \urcorner)$, (2) $H(\ulcorner k \urcorner)$ when $\mathbf{type}_L[k]$, and (3) $H(a)$. By the induction hypothesis, (1) and (2) imply $C(\ulcorner a \urcorner)$ and $C(\ulcorner k \urcorner)$ when $\mathbf{type}_L[k]$. Suppose $V_\varphi(\mathsf{gea}(a, \ulcorner k \urcorner)) = \mathrm{T}$ for some $\varphi \in \mathsf{assign}(S)$.

Then $V_\varphi(a) = V_\varphi(\ulcorner e' \urcorner)$ for some eval-free expression e' and (3) implies $V_\varphi(\neg\mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner e' \urcorner)) = \mathrm{T}$. By Lemma 6.4.1, this implies $H(\ulcorner e' \urcorner)$, and so by the induction hypothesis, $C(\ulcorner e' \urcorner)$ holds. Therefore, $C(\ulcorner e \urcorner)$ holds since $V_\varphi(e)$ is defined in terms of $V_\varphi(a)$, $V_\varphi(k)$ when $\mathbf{type}_L[k]$, and $V_\varphi(e')$. Now suppose that $V_\varphi(\mathsf{gea}(a, \ulcorner k \urcorner)) = \mathrm{F}$ for all $\varphi \in \mathsf{assign}(S)$. Then, by Lemma 6.3.1, $V_\varphi(e)$ is the undefined value for kind k for all $\varphi \in \mathsf{assign}(S)$. $C(\ulcorner e \urcorner)$ follows immediately since $V_\varphi(e)$ does not depend on φ.

Part 2 Similar to the proof of part 1. □

Lemma 6.4.6 (Syntactically Closed) *Let M be a standard model of a normal theory $T = (L, \Gamma)$ and e be an expression of L. If $M \models \mathsf{syn\text{-}closed}(\ulcorner e \urcorner)$, then e is semantically closed in M.*

Proof Let $M = (S, V)$ be a standard model of T. Assume $M \models \mathsf{syn\text{-}closed}(\ulcorner e \urcorner)$. Then, by the definition of $\mathsf{syn\text{-}closed}$, e is a proper expression and $M \models \neg\mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner e \urcorner)$ for all $x \in \mathcal{S}$. Hence, by part 2 of Lemma 6.4.5, $V_\varphi(e) = V_{\varphi'}(e)$ for all $\varphi, \varphi' \in \mathsf{assign}(S)$. Therefore, e is semantically closed in M. □

The next lemma shows that part 1 of Lemma 6.4.5 is not sufficient to prove the previous lemma.

Lemma 6.4.7 *Let $M = (S, V)$ be a standard model of a normal theory $T = (L, \Gamma)$ and e be a proper expression of L. Suppose $V_\varphi(e) = V_{\varphi[x \mapsto d]}(e)$ for all $\varphi \in \mathsf{assign}(S)$, $x \in \mathcal{S}$, and $d \in D_\mathrm{c}$. Then it is not necessary that e be syntactically closed.*

Proof See the example in subsection 7.3. □

Lemma 6.4.8 (Cleanse) *Let $M = (S, V)$ be a standard model for a normal theory $T = (L, \Gamma)$ and e be an expression of L.*

1. *If e is an operator, type, term, or formula and $V_\varphi(\mathsf{cleanse}(\ulcorner e \urcorner)) \neq \bot$, then*

   $$H^{-1}(V_\varphi(\mathsf{cleanse}(\ulcorner e \urcorner)))$$

   is an operator similar to e, type, term, or formula, respectively, that is eval-free for all $\varphi \in \mathsf{assign}(S)$.

2. $\mathsf{cleanse}(\ulcorner e \urcorner)$ *is semantically closed in* M.

3. *If* e *is an operator, type, term, or formula that contains an evaluation* $[\![e']\!]_k$ *not in a quotation and* $M \models \mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner e' \urcorner)$ *for some* $x \in \mathcal{S}$, *then*

$$M \models \mathsf{cleanse}(\ulcorner e \urcorner)\!\uparrow .$$

4. *If* e *is a type, term, or formula and* $M \models \mathsf{cleanse}(\ulcorner e \urcorner)\!\downarrow$, *then*

$$M \models [\![\mathsf{cleanse}(\ulcorner e \urcorner)]\!]_{k[e]} = e.$$

Proof Let $A(\ulcorner e \urcorner)$ mean $\mathsf{cleanse}(\ulcorner e \urcorner)$.

Part 1 Follows straightforwardly by induction on the complexity of e.

Part 2 Our proof is by induction on the complexity of e. We must show that $A(\ulcorner e \urcorner)$ is semantically closed in M. There are 12 cases corresponding to the 12 formula schemas used to defined $\mathsf{cleanse}(\ulcorner e \urcorner)$ when e is an expression.

> **Case 1**: e is improper. By the definition of $\mathsf{cleanse}$, $A(\ulcorner e \urcorner) = \ulcorner e \urcorner$. The last expression is a quotation and is thus semantically closed in M by Proposition 6.2.1 and Lemma 6.4.6. Therefore, $A(\ulcorner e \urcorner)$ is semantically closed in M.
>
> **Cases 2–8**: $A(\ulcorner e \urcorner)$ is semantically closed in M by the induction hypothesis, the definition of $\mathsf{cleanse}$, and Lemma 6.3.3.
>
> **Case 9**: $e = \ulcorner e' \urcorner$. By the definition of $\mathsf{cleanse}$,
>
> $$A(\ulcorner e \urcorner) = A(\ulcorner\ulcorner e' \urcorner\urcorner) = \ulcorner\ulcorner e' \urcorner\urcorner$$
>
> in M. The last expression is a quotation and is thus semantically closed in M by Proposition 6.2.1 and Lemma 6.4.6. Therefore, $A(\ulcorner e \urcorner)$ is semantically closed in M.
>
> **Case 10**: $e = [\![a]\!]_{\mathrm{ty}}$. By the definition of $\mathsf{cleanse}$,
>
> $$A(\ulcorner e \urcorner) = \mathsf{if}(\mathsf{syn\text{-}closed}(\widehat{a}), w, \bot_{\mathsf{C}})$$
>
> in M where $\widehat{a} = A(\ulcorner a \urcorner)$, $\widehat{a}' = \mathsf{coerce\text{-}to\text{-}type}([\![\widehat{a}]\!]_{\mathrm{te}})$, and $w = \ulcorner \mathsf{if}(\mathsf{gea}(a, \ulcorner \mathsf{type} \urcorner), \lfloor \widehat{a}' \rfloor, \mathsf{C}) \urcorner$. If $M \models A(\ulcorner e \urcorner)\!\uparrow$, then $A(\ulcorner e \urcorner)$ is semantically closed in M. Therefore, it suffices to show that w is semantically

closed in M under the assumption $M \models \mathsf{syn\text{-}closed}(\widehat{a})$. Then $\widehat{a}'$ is semantically closed in M by Lemma 6.4.6. This implies w is semantically closed in M by Lemma 6.3.3.

Case 11: $e = [\![a]\!]_\alpha$. Similar to case 10.

Case 12: $e = [\![a]\!]_{\mathrm{fo}}$. Similar to case 10.

Part 3 Follows immediately from the definition of $\mathsf{cleanse}$.

Part 4 Let e be a type, term, or formula. Assume

$$M \models A(\ulcorner e \urcorner)\!\downarrow \quad [\text{designated } H(\ulcorner e \urcorner)].$$

We must show that

$$M \models [\![A(\ulcorner e \urcorner)]\!]_{k[e]} = e \quad [\text{designated } C(\ulcorner e \urcorner)].$$

Our proof is by induction on the complexity of e. There are 10 cases corresponding to the 10 formula schemas used to defined $\mathsf{cleanse}(\ulcorner e \urcorner)$ when e is a type, term, or formula.

Case 1: $e = O(e_1, \ldots, e_n)$ and $O = (o :: k_1, \ldots, k_{n+1})$. By the definition of $\mathsf{cleanse}$, $H(\ulcorner e \urcorner)$ implies (1) $H(\ulcorner k_i \urcorner)$ for all i with $1 \leq i \leq n+1$ and $\mathbf{type}_L[k_i]$ and (2) $H(\ulcorner e_i \urcorner)$ for all i with $1 \leq i \leq n$. By the induction hypothesis, this implies (1) $C(\ulcorner k_i \urcorner)$ for all i with $1 \leq i \leq n+1$ and $\mathbf{type}_L[k_i]$ and (2) $C(\ulcorner e_i \urcorner)$ for all i with $1 \leq i \leq n$. Let $\varphi \in \mathsf{assign}(S)$. Then

$$\begin{aligned}
& V_\varphi([\![A(\ulcorner e \urcorner)]\!]_{k[e]}) \\
= \; & V_\varphi([\![A(\ulcorner O(e_1, \ldots, e_n) \urcorner)]\!]_{k[e]}) \\
= \; & V_\varphi([\![\ulcorner \lfloor A(\ulcorner O \urcorner) \rfloor (\lfloor A(\ulcorner e_1 \urcorner) \rfloor, \ldots, \lfloor A(\ulcorner e_n \urcorner) \rfloor) \urcorner]\!]_{k[e]}) \\
= \; & V_\varphi(\overline{A(\ulcorner O \urcorner)}([\![A(\ulcorner e_1 \urcorner)]\!]_{k[e_1]}, \ldots, [\![A(\ulcorner e_n \urcorner)]\!]_{k[e_n]})) \\
= \; & V_\varphi(O(e_1, \ldots, e_n)) \\
= \; & V_\varphi(e)
\end{aligned}$$

where

$$\overline{A(\ulcorner O \urcorner)} = (o :: \overline{A(\ulcorner k_1 \urcorner)}, \ldots, \overline{A(\ulcorner k_{n+1} \urcorner)})$$

and

$$\overline{A(\ulcorner k_i \urcorner)} = \begin{cases} [\![A(\ulcorner k_i \urcorner)]\!]_{\mathrm{ty}} & \text{if } \mathbf{type}_L[k_i] \\ k_i & \text{if } k_i = \mathsf{type} \text{ or } \mathsf{formula}. \end{cases}$$

The third line is by the definition of $\mathsf{cleanse}$; the fourth is by Lemma 6.3.2 and part 1 of this lemma; and the fifth holds since (1) O and $\overline{A(\ulcorner O \urcorner)}$ are similar by part 1 of this lemma, (2) $C(\ulcorner k_i \urcorner)$ holds for all i with $1 \leq i \leq n+1$ and $\mathbf{type}_L[k_i]$, and (3) $C(\ulcorner e_i \urcorner)$ holds for all i with $1 \leq i \leq n$. Therefore, $C(\ulcorner e \urcorner)$ holds.

Cases 2, 4–6: Similar to case 1.

Case 3: $e = (\star x : \alpha \,.\, e')$ where $\star$ is Λ, λ, $\exists$, ι, or ϵ. By the definition of $\mathsf{cleanse}$, $H(\ulcorner e \urcorner)$ implies $H(\ulcorner \alpha \urcorner)$ and $H(\ulcorner e' \urcorner)$. By the induction hypothesis, this implies $C(\ulcorner \alpha \urcorner)$ and $C(\ulcorner e' \urcorner)$. Then

$$\begin{aligned}
& V_\varphi([\![A(\ulcorner e \urcorner)]\!]_{k[e]}) \\
= \;& V_\varphi([\![A(\ulcorner (\star x : \alpha \,.\, e')\urcorner)]\!]_{k[e]}) \\
= \;& V_\varphi([\![\ulcorner (\star x : \lfloor A(\ulcorner \alpha \urcorner)\rfloor \,.\, \lfloor A(\ulcorner e' \urcorner)\rfloor)\urcorner]\!]_{k[e]}) \\
= \;& V_\varphi((\star x : [\![A(\ulcorner \alpha \urcorner)]\!]_{\mathrm{ty}} \,.\, [\![A(\ulcorner e' \urcorner)]\!]_{k[e']})) \\
= \;& V_\varphi((\star x : \alpha \,.\, e')) \\
= \;& V_\varphi(e)
\end{aligned}$$

The third line is by the definition of $\mathsf{cleanse}$; the fourth is by Lemma 6.3.2 and parts 1 and 2 of this lemma; and the fifth is by $C(\ulcorner \alpha \urcorner)$ and $C(\ulcorner e' \urcorner)$. Therefore, $C(\ulcorner e \urcorner)$ holds.

Case 7: $e = \ulcorner e' \urcorner$. Let $\varphi \in \mathsf{assign}(S)$. Then

$$\begin{aligned}
& V_\varphi([\![A(\ulcorner e \urcorner)]\!]_{k[e]}) \\
= \;& V_\varphi([\![A(\ulcorner\ulcorner e' \urcorner\urcorner)]\!]_{\mathrm{te}}) \\
= \;& V_\varphi([\![\ulcorner\ulcorner e' \urcorner\urcorner]\!]_{\mathrm{te}}) \\
= \;& V_\varphi(\ulcorner e' \urcorner) \\
= \;& V_\varphi(e).
\end{aligned}$$

The third line is by the definition of $\mathsf{cleanse}$ and the fourth is by Lemma 6.3.1. Therefore, $C(\ulcorner e \urcorner)$ holds.

Case 8: $e = [\![a]\!]_{\mathrm{ty}}$. Let $\varphi \in \mathsf{assign}(S)$. Suppose $V_\varphi(\mathsf{gea}(a, \ulcorner \mathsf{type} \urcorner)) = \mathrm{T}$. By the definition of $\mathsf{cleanse}$, $H(\ulcorner e \urcorner)$ implies $H(\ulcorner a \urcorner)$. By the induction hypothesis, this implies $C(\ulcorner a \urcorner)$. Let

$u = \mathsf{coerce\text{-}to\text{-}type}(\llbracket A(\ulcorner a \urcorner) \rrbracket_{\mathrm{te}})$. Then

$$\begin{aligned}
& V_\varphi(\llbracket A(\ulcorner e \urcorner) \rrbracket_{k[e]}) \\
= \; & V_\varphi(\llbracket A(\ulcorner \llbracket a \rrbracket_{\mathrm{ty}} \urcorner) \rrbracket_{\mathrm{ty}}) \\
= \; & V_\varphi(\llbracket \mathsf{if}(\mathsf{syn\text{-}closed}(A(\ulcorner a \urcorner)), \\
& \quad \ulcorner \mathsf{if}(\mathsf{gea}(a, \ulcorner \mathsf{type} \urcorner), \lfloor u \rfloor, \mathsf{C}) \urcorner, \\
& \quad \bot_{\mathsf{C}}) \rrbracket_{\mathrm{ty}}) \\
= \; & V_\varphi(\llbracket \ulcorner \mathsf{if}(\mathsf{gea}(a, \ulcorner \mathsf{type} \urcorner), \lfloor u \rfloor, \mathsf{C}) \urcorner \rrbracket_{\mathrm{ty}}) \\
= \; & V_\varphi(\mathsf{if}(\mathsf{gea}(a, \ulcorner \mathsf{type} \urcorner), \llbracket u \rrbracket_{\mathrm{ty}}, \mathsf{C})) \\
= \; & V_\varphi(\llbracket \mathsf{coerce\text{-}to\text{-}type}(\llbracket A(\ulcorner a \urcorner) \rrbracket_{\mathrm{te}}) \rrbracket_{\mathrm{ty}}) \\
= \; & V_\varphi(\llbracket \mathsf{coerce\text{-}to\text{-}type}(a) \rrbracket_{\mathrm{ty}}) \\
= \; & V_\varphi(\llbracket a \rrbracket_{\mathrm{ty}}) \\
= \; & V_\varphi(e).
\end{aligned}$$

The third line is by the definition of $\mathsf{cleanse}$; the fourth is by $H(\ulcorner e \urcorner)$; the fifth is by Lemma 6.3.2 and part 1 of this lemma; the sixth by the hypothesis that $V_\varphi(\mathsf{gea}(a, \ulcorner \mathsf{type} \urcorner)) = \textsc{t}$ and the definition of u; the seventh is by $C(\ulcorner a \urcorner)$; and the eighth is by $V_\varphi(\mathsf{gea}(a, \ulcorner \mathsf{type} \urcorner)) = \textsc{t}$. Therefore, $C(\ulcorner e \urcorner)$ holds.

Now suppose $V_\varphi(\mathsf{gea}(a, \ulcorner \mathsf{type} \urcorner)) = \textsc{f}$. By a similar derivation to the one above, $C(\ulcorner e \urcorner)$ holds.

Case 9: $e = \llbracket a \rrbracket_\alpha$. Let $\varphi \in \mathsf{assign}(S)$. Suppose $V_\varphi(\mathsf{gea}(a, \ulcorner \alpha \urcorner) = \textsc{t}$ and $V_\varphi(\llbracket a \rrbracket_{\mathrm{te}} \downarrow \alpha) = \textsc{t}$. By the definition of $\mathsf{cleanse}$, $H(\ulcorner e \urcorner)$ implies $H(\ulcorner \alpha \urcorner)$ and $H(\ulcorner a \urcorner)$. By the induction hypothesis, this implies $C(\ulcorner \alpha \urcorner)$ and $C(\ulcorner a \urcorner)$. Let $u = \mathsf{coerce\text{-}to\text{-}term}(\llbracket A(\ulcorner a \urcorner) \rrbracket_{\mathrm{te}})$. Then

$$\begin{aligned}
& V_\varphi(\llbracket A(\ulcorner e \urcorner) \rrbracket_{k[e]}) \\
= \; & V_\varphi(\llbracket A(\ulcorner \llbracket a \rrbracket_\alpha \urcorner) \rrbracket_{\mathrm{te}}) \\
= \; & V_\varphi(\llbracket \mathsf{if}(\mathsf{syn\text{-}closed}(A(\ulcorner a \urcorner)), \\
& \quad \ulcorner \mathsf{if}(\mathsf{gea}(a, \ulcorner \alpha \urcorner) \wedge \lfloor u \rfloor \downarrow \lfloor A(\ulcorner \alpha \urcorner) \rfloor, \lfloor u \rfloor, \bot_{\mathsf{C}}) \urcorner, \\
& \quad \bot_{\mathsf{C}}) \rrbracket_{\mathrm{te}}) \\
= \; & V_\varphi(\llbracket \ulcorner \mathsf{if}(\mathsf{gea}(a, \ulcorner \alpha \urcorner) \wedge \lfloor u \rfloor \downarrow \lfloor A(\ulcorner \alpha \urcorner) \rfloor, \lfloor u \rfloor, \bot_{\mathsf{C}}) \urcorner \rrbracket_{\mathrm{te}}) \\
= \; & V_\varphi(\mathsf{if}(\mathsf{gea}(a, \ulcorner \alpha \urcorner) \wedge \llbracket u \rrbracket_{\mathrm{te}} \downarrow \llbracket A(\ulcorner \alpha \urcorner) \rrbracket_{\mathrm{ty}}, \llbracket u \rrbracket_{\mathrm{te}}, \bot_{\mathsf{C}})) \\
= \; & V_\varphi(\mathsf{if}(\llbracket \mathsf{coerce\text{-}to\text{-}term}(\llbracket A(\ulcorner a \urcorner) \rrbracket_{\mathrm{te}}) \rrbracket_{\mathrm{te}} \downarrow \llbracket A(\ulcorner \alpha \urcorner) \rrbracket_{\mathrm{ty}}, \\
& \quad \llbracket \mathsf{coerce\text{-}to\text{-}term}(\llbracket A(\ulcorner a \urcorner) \rrbracket_{\mathrm{te}}) \rrbracket_{\mathrm{te}},
\end{aligned}$$

$$
\begin{array}{rl}
& \bot_{\mathsf{C}})) \\
= & V_\varphi(\mathsf{if}(\llbracket \mathsf{coerce\text{-}to\text{-}term}(a) \rrbracket_{\mathrm{te}} \downarrow \alpha, \\
& \quad \llbracket \mathsf{coerce\text{-}to\text{-}term}(a) \rrbracket_{\mathrm{te}}, \\
& \quad \bot_{\mathsf{C}})) \\
= & V_\varphi(\mathsf{if}(\llbracket a \rrbracket_{\mathrm{te}} \downarrow \alpha, \llbracket a \rrbracket_{\mathrm{te}}, \bot_{\mathsf{C}})) \\
= & V_\varphi(\llbracket a \rrbracket_\alpha) \\
= & V_\varphi(e).
\end{array}
$$

The third line is by the definition of $\mathsf{cleanse}$; the fourth is by $H(\ulcorner e \urcorner)$; the fifth is by Lemma 6.3.2 and part 1 of this lemma; the sixth by the hypothesis that $V_\varphi(\mathsf{gea}(a, \ulcorner \mathsf{type} \urcorner)) = \mathrm{T}$ and the definition of u; the seventh is by $C(\ulcorner a \urcorner)$ and $C(\ulcorner \alpha \urcorner)$; the eighth is by $V_\varphi(\mathsf{gea}(a, \ulcorner \mathsf{type} \urcorner)) = \mathrm{T}$; and the ninth is by $V_\varphi(\llbracket a \rrbracket_{\mathrm{te}} \downarrow \alpha) = \mathrm{T}$. Therefore, $C(\ulcorner e \urcorner)$ holds.

Now suppose $V_\varphi(\mathsf{gea}(a, \ulcorner \mathsf{type} \urcorner)) = \mathrm{F}$ or $V_\varphi(\llbracket a \rrbracket_{\mathrm{te}} \downarrow \alpha) = \mathrm{F}$. By a similar derivation to the one above, $C(\ulcorner e \urcorner)$ holds.

Case 10: $e = \llbracket a \rrbracket_{\mathrm{fo}}$. Similar to case 8.

□

Lemma 6.4.9 (Substitution A) *Let $M = (S, V)$ be a standard model for a normal theory $T = (L, \Gamma)$ and a be a term, x be a symbol, and e be an expression of L.*

1. *If e is an operator, type, term, or formula and $V_\varphi(\mathsf{sub}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner e \urcorner)) \neq \bot$, then*

   $$H^{-1}(V_\varphi(\mathsf{sub}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner e \urcorner)))$$

   is an operator similar to e, type, term, or formula, respectively, that is eval-free for all $\varphi \in \mathsf{assign}(S)$.

2. *$\mathsf{sub}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner e \urcorner)$ is semantically closed in M.*

3. *If e is an operator, type, term, or formula that contains an evaluation $\llbracket e' \rrbracket_k$ not in a quotation and $M \models \mathsf{free\text{-}in}(\ulcorner y \urcorner, \mathsf{sub}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner e' \urcorner))$ for some $y \in \mathcal{S}$, then*

   $$M \models \mathsf{sub}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner e \urcorner)\!\uparrow .$$

4. If e is a type, term, or formula, $M \models \mathsf{sub}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner e \urcorner)\downarrow$, and $M \models \neg\mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner e \urcorner)$, then

$$M \models [\![\mathsf{sub}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner e \urcorner)]\!]_{k[e]} = e.$$

Proof Let $S(\ulcorner e \urcorner)$ mean

$$\mathsf{sub}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner e \urcorner)$$

and $E(\ulcorner e \urcorner)$ mean

$$H^{-1}(V_\varphi(S(\ulcorner e \urcorner))).$$

Part 1a Fix $\varphi \in \mathsf{assign}(S)$. Assume e is an operator, type, term, or formula and $V_\varphi(S(\ulcorner e \urcorner)) \neq \bot$. We must show $E(\ulcorner e \urcorner)$ is an operator similar to e, type, term, or formula, respectively. Our proof is by induction on the complexity of e. There are 13 cases corresponding to the 13 formula schemas used to define $\mathsf{sub}(\ulcorner e_1 \urcorner, \ulcorner e_2 \urcorner, \ulcorner e_3 \urcorner)$ when e_1 is a term, e_2 is a symbol, and e_3 is a proper expression.

Case 1: $e = (o :: k_1, \ldots, k_{n+1})$. By the first formula schema of the definition of sub, $V_\varphi(S(\ulcorner \mathsf{type} \urcorner)) = V_\varphi(\ulcorner \mathsf{type} \urcorner)$ and $V_\varphi(S(\ulcorner \mathsf{formula} \urcorner)) = V_\varphi(\ulcorner \mathsf{formula} \urcorner)$, and so $E(\ulcorner \mathsf{type} \urcorner) = \mathsf{type}$ and $E(\ulcorner \mathsf{formula} \urcorner) = \mathsf{formula}$. By the second formula schema of the definition of sub,

$$\begin{aligned}
& E(\ulcorner e \urcorner) \\
= \; & H^{-1}(V_\varphi(S(\ulcorner e \urcorner))) \\
= \; & H^{-1}(V_\varphi(\ulcorner (o :: \lfloor S(\ulcorner k_1 \urcorner) \rfloor, \ldots, \lfloor S(\ulcorner k_{n+1} \urcorner) \rfloor) \urcorner)) \\
= \; & H^{-1}(V_\varphi(\ulcorner (o :: E(\ulcorner k_1 \urcorner), \ldots, E(\ulcorner k_{n+1} \urcorner)) \urcorner)) \\
= \; & (o :: E(\ulcorner k_1 \urcorner), \ldots, E(\ulcorner k_{n+1} \urcorner)).
\end{aligned}$$

By the induction hypothesis, for all i with $1 \leq i \leq n+1$ and $\mathbf{type}_L[k_i]$, $E(\ulcorner k_i \urcorner)$ is a type. Therefore, if e is an operator, then $E(\ulcorner e \urcorner)$ is an operator similar to e.

Case 2: $e = O(e_1, \ldots, e_n)$ and $O = (o :: k_1, \ldots, k_{n+1})$. By the third formula schema of the definition of sub,

$$\begin{aligned}
& E(\ulcorner e \urcorner) \\
= \; & H^{-1}(V_\varphi(S(\ulcorner e \urcorner))) \\
= \; & H^{-1}(V_\varphi(\ulcorner \lfloor S(\ulcorner O \urcorner) \rfloor (\lfloor S(\ulcorner e_1 \urcorner) \rfloor, \ldots, \lfloor S(\ulcorner e_n \urcorner) \rfloor) \urcorner)) \\
= \; & H^{-1}(V_\varphi(\ulcorner E(\ulcorner O \urcorner)(E(\ulcorner e_1 \urcorner), \ldots, E(\ulcorner e_n \urcorner)) \urcorner)) \\
= \; & E(\ulcorner O \urcorner)(E(\ulcorner e_1 \urcorner), \ldots, E(\ulcorner e_n \urcorner)).
\end{aligned}$$

By the induction hypothesis, (1) $E(\ulcorner O \urcorner)$ is an operator similar to O and (2) for all i with $1 \leq i \leq n$, if e_i is a type, term, or formula, then $E(\ulcorner e_i \urcorner)$ is a type, term, or formula, respectively. Therefore, if e is a type, term, or formula, then $E(\ulcorner e \urcorner)$ is a type, term, or a formula, respectively.

Cases 3–13. Similar to case 2. cases 3 and 5 use part 1 of Lemma 6.4.8.

Part 1b Fix $\varphi \in \mathsf{assign}(S)$. Assume e is an operator, type, term, or formula and $V_\varphi(\mathsf{sub}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner e \urcorner)) \neq \bot$. We must show $E(\ulcorner e \urcorner)$ is eval-free. Our proof is by induction on the complexity of e. There are 13 cases as for part 1a.

Cases 1–2, 4, 6–9. Each eval symbol occurring in e that is not in a quotation is removed by the definition of sub. Hence any eval symbol occurring in $E(\ulcorner e \urcorner)$ that is not in a quotation must have been introduced explicitly in $S(\ulcorner e \urcorner)$ or implicitly in $S(\ulcorner e \urcorner)$ via a subcomponent $S(\ulcorner e' \urcorner)$ of $S(\ulcorner e \urcorner)$ where e' is a component of e. In these cases, no eval symbols are explicitly introduced in $S(\ulcorner e \urcorner)$ by the definition of sub, and no eval symbols are implicitly introduced in $S(\ulcorner e \urcorner)$ by the induction hypothesis. Therefore, $E(\ulcorner e \urcorner)$ is eval-free.

Case 3. $e = (x : \alpha)$. $E(\ulcorner e \urcorner)$ is either $H^{-1}(V_\varphi(\mathsf{cleanse}(\ulcorner a \urcorner)))$ or $\bot_{\mathsf{C}}$. Therefore, $E(\ulcorner e \urcorner)$ is eval-free since $H^{-1}(V_\varphi(\mathsf{cleanse}(\ulcorner a \urcorner)))$ is eval-free by part 1 of Lemma 6.4.8 and $\bot_{\mathsf{C}}$ is obviously eval-free.

Case 5. $e = (\star x : \alpha \ . \ e')$ where $\star$ is Λ, λ, $\exists$, ι, or ϵ. $E(\ulcorner \alpha \urcorner)$ is eval-free by the induction hypothesis, and $H^{-1}(V_\varphi(\mathsf{cleanse}(\ulcorner e' \urcorner)))$ is eval-free by part 1 of Lemma 6.4.8. Therefore, $E(\ulcorner e \urcorner)$ is eval-free by the definition of sub.

Case 10. $e = \ulcorner e' \urcorner$. $E(\ulcorner e \urcorner) = e$. Therefore, $E(\ulcorner e \urcorner)$ is eval-free since e is a quotation and hence eval-free.

Cases 11–13. $e = [\![b]\!]_k$. By the induction hypothesis, $E(S(\ulcorner b \urcorner))$ is eval-free, $E(S([\![S(\ulcorner b \urcorner)]\!]_{\mathrm{te}}))$ is eval-free, and $E(S(\ulcorner k \urcorner))$ is eval-free if $\mathbf{type}_L[k]$. Therefore, $E(\ulcorner e \urcorner)$ is eval-free by the definition of sub.

Part 2 Similar to the proof of part 2 of Lemma 6.4.8.

Part 3 Follows immediately from the definition of sub.

Part 4 Let e is a type, term, or formula. Assume

$$M \models S(\ulcorner e \urcorner)\downarrow \quad [\text{designated } H_1(\ulcorner e \urcorner)]$$

and

$$M \models \neg\mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner e \urcorner) \quad [\text{designated } H_2(\ulcorner e \urcorner)].$$

We must show that

$$M \models [\![S(\ulcorner e \urcorner)]\!]_{k[e]} = e \quad [\text{designated } C(\ulcorner e \urcorner)].$$

Our proof is by induction on the complexity of e. There are 12 cases corresponding to the 12 formula schemas used to define $\mathsf{sub}(\ulcorner e_1 \urcorner, \ulcorner e_2 \urcorner, \ulcorner e_3 \urcorner)$ when e_1 is a term, e_2 is a symbol, and e_3 is a type, term, or formula.

Case 1: $e = O(e_1, \ldots, e_n)$ and $O = (o :: k_1, \ldots, k_{n+1})$. By the definitions of sub and free-in, $H_1(\ulcorner e \urcorner)$ and $H_2(\ulcorner e \urcorner)$ imply (1) $H_1(\ulcorner k_i \urcorner)$ and $H_2(\ulcorner k_i \urcorner)$ for all i with $1 \leq i \leq n+1$ and $\mathbf{type}_L[k_i]$ and (2) $H_1(\ulcorner e_i \urcorner)$ and $H_2(\ulcorner e_i \urcorner)$ for all i with $1 \leq i \leq n$. By the induction hypothesis, this implies (1) $C(\ulcorner k_i \urcorner)$ for all i with $1 \leq i \leq n+1$ and $\mathbf{type}_L[k_i]$ and (2) $C(\ulcorner e_i \urcorner)$ for all i with $1 \leq i \leq n$. Let $\varphi \in \mathsf{assign}(S)$. Then

$$\begin{aligned}
& V_\varphi([\![S(\ulcorner e \urcorner)]\!]_{k[e]}) \\
= \; & V_\varphi([\![S(\ulcorner O(e_1, \ldots, e_n) \urcorner)]\!]_{k[e]}) \\
= \; & V_\varphi([\![\ulcorner \lfloor S(\ulcorner O \urcorner) \rfloor (\lfloor S(\ulcorner e_1 \urcorner) \rfloor, \ldots, \lfloor S(\ulcorner e_n \urcorner) \rfloor) \urcorner]\!]_{k[e]}) \\
= \; & V_\varphi(\overline{S(\ulcorner O \urcorner)}([\![S(\ulcorner e_1 \urcorner)]\!]_{k[e_1]}, \ldots, [\![S(\ulcorner e_n \urcorner)]\!]_{k[e_n]})) \\
= \; & V_\varphi(O(e_1, \ldots, e_n)) \\
= \; & V_\varphi(e)
\end{aligned}$$

where

$$\overline{S(\ulcorner O \urcorner)} = (o :: \overline{S(\ulcorner k_1 \urcorner)}, \ldots, \overline{S(\ulcorner k_{n+1} \urcorner)})$$

and

$$\overline{S(\ulcorner k_i \urcorner)} = \begin{cases} [\![S(\ulcorner k_i \urcorner)]\!]_{\mathrm{ty}} & \text{if } \mathbf{type}_L[k_i] \\ k_i & \text{if } k_i = \mathsf{type} \text{ or } \mathsf{formula}. \end{cases}$$

The third line is by the definition of sub; the fourth is by Lemma 6.3.2 and part 1 of this lemma; and the fifth holds since (1) O and $\overline{S(\ulcorner O \urcorner)}$ are similar by part 1 of this lemma, (2) $C(\ulcorner k_i \urcorner)$ holds for all i with $1 \leq i \leq n+1$ and $\mathbf{type}_L[k_i]$, and (3) $C(\ulcorner e_i \urcorner)$ holds for all i with $1 \leq i \leq n$. Therefore, $C(\ulcorner e \urcorner)$ holds.

Case 2: $e = (x : \alpha)$. The hypothesis $H_2(\ulcorner e \urcorner)$ is false in this case.

Case 3: $e = (y : \alpha)$ where $x \neq y$. By the definitions of sub and $\mathsf{free\text{-}in}$, $H_1(\ulcorner e \urcorner)$ and $H_2(\ulcorner e \urcorner)$ imply $H_1(\ulcorner \alpha \urcorner)$ and $H_2(\ulcorner \alpha \urcorner)$. By the induction hypothesis, this implies $C(\ulcorner \alpha \urcorner)$. Let $\varphi \in \mathsf{assign}(S)$. Then

$$\begin{aligned}
& V_\varphi([\![S(\ulcorner e \urcorner)]\!]_{k[e]}) \\
= \; & V_\varphi([\![S(\ulcorner (y : \alpha) \urcorner)]\!]_{\mathrm{te}}) \\
= \; & V_\varphi([\![\ulcorner (y : \lfloor S(\alpha) \rfloor) \urcorner]\!]_{\mathrm{te}}) \\
= \; & V_\varphi((y : [\![S(\ulcorner \alpha \urcorner)]\!]_{\mathrm{ty}})) \\
= \; & V_\varphi((y : \alpha)) \\
= \; & V_\varphi(e).
\end{aligned}$$

The third line is by the definition of sub; the fourth is by Lemma 6.3.2 and part 1 of this lemma; and the fifth is by $C(\ulcorner \alpha \urcorner)$. Therefore, $C(\ulcorner e \urcorner)$ holds.

Case 4: $e = (\star x : \alpha \; . \; e')$ where $\star$ is Λ, λ, $\exists$, ι, or ϵ. By the definitions of sub and $\mathsf{free\text{-}in}$, $H_1(\ulcorner e \urcorner)$ and $H_2(\ulcorner e \urcorner)$ imply $H_1(\ulcorner \alpha \urcorner)$, $H_2(\ulcorner \alpha \urcorner)$, and $M \models \mathsf{cleanse}(\ulcorner e' \urcorner)\!\downarrow$. By the induction hypothesis, $H_1(\ulcorner \alpha \urcorner)$ and $H_2(\ulcorner \alpha \urcorner)$ implies $C(\ulcorner \alpha \urcorner)$. Let $\varphi \in \mathsf{assign}(S)$. Then

$$\begin{aligned}
& V_\varphi([\![S(\ulcorner e \urcorner)]\!]_{k[e]}) \\
= \; & V_\varphi([\![S(\ulcorner (\star x : \alpha \; . \; e') \urcorner)]\!]_{k[e]}) \\
= \; & V_\varphi([\![\ulcorner (\star x : \lfloor S(\ulcorner \alpha \urcorner) \rfloor \; . \; \lfloor \mathsf{cleanse}(\ulcorner e' \urcorner) \rfloor) \urcorner]\!]_{k[e]}) \\
= \; & V_\varphi((\star x : [\![S(\ulcorner \alpha \urcorner)]\!]_{\mathrm{ty}} \; . \; [\![\mathsf{cleanse}(\ulcorner e' \urcorner)]\!]_{k[e']})) \\
= \; & V_\varphi((\star x : \alpha \; . \; e')) \\
= \; & V_\varphi(e)
\end{aligned}$$

The third line is by the definition of sub; the fourth is by Lemma 6.3.2, parts 1 and 2 of this lemma, and parts 1 and 2 of Lemma 6.4.8; and the fifth is by $C(\ulcorner \alpha \urcorner)$, $M \models \mathsf{cleanse}(\ulcorner e' \urcorner)\!\downarrow$, and part 4 of Lemma 6.4.8.

Case 5–8: Similar to case 3.

Case 9. $e = \ulcorner e' \urcorner$. $C(\ulcorner e \urcorner)$ is always true by the definition of sub.

Case 10: $e = [\![b]\!]_{\mathrm{ty}}$. Let $\varphi \in \mathsf{assign}(S)$. Suppose $V_\varphi(\mathsf{gea}(b, \ulcorner \mathsf{type} \urcorner)) = \mathrm{T}$. Then there is an eval-free expression e' such that $V_\varphi(b) = V_\varphi(\ulcorner e' \urcorner)$. By the definitions of sub and $\mathsf{free\text{-}in}$, $H_1(\ulcorner e \urcorner)$ and $H_2(\ulcorner e \urcorner)$ imply

(1) $H_1(\ulcorner b \urcorner)$ and $H_2(\ulcorner b \urcorner)$ and (2) $H_1(\ulcorner e' \urcorner)$ and $H_2(\ulcorner e' \urcorner)$. By the induction hypothesis, this implies (1) $C(\ulcorner b \urcorner)$ and (2) $C(\ulcorner e' \urcorner)$. Let $u = \mathsf{coerce\text{-}to\text{-}type}(S([\![S(\ulcorner b \urcorner)]\!]_{\mathrm{te}}))$. Then

$$
\begin{aligned}
& V_\varphi([\![S(\ulcorner e \urcorner)]\!]_{k[e]}) \\
= \; & V_\varphi([\![S(\ulcorner [\![b]\!]_{\mathrm{ty}} \urcorner)]\!]_{\mathrm{ty}}) \\
= \; & V_\varphi([\![\mathsf{if}(\mathsf{syn\text{-}closed}(S(\ulcorner b \urcorner)), \\
& \quad \ulcorner \mathsf{if}(\mathsf{gea}(\lfloor S(\ulcorner b \urcorner) \rfloor, \ulcorner \mathsf{type} \urcorner), \lfloor u \rfloor, \mathsf{C}) \urcorner, \\
& \quad \bot_{\mathsf{C}})]\!]_{\mathrm{ty}}) \\
= \; & V_\varphi([\![\ulcorner \mathsf{if}(\mathsf{gea}(\lfloor S(\ulcorner b \urcorner) \rfloor, \ulcorner \mathsf{type} \urcorner), \lfloor u \rfloor, \mathsf{C}) \urcorner]\!]_{\mathrm{ty}}) \\
= \; & V_\varphi(\mathsf{if}(\mathsf{gea}([\![S(\ulcorner b \urcorner)]\!]_{\mathrm{te}}, \ulcorner \mathsf{type} \urcorner), [\![u]\!]_{\mathrm{ty}}, \mathsf{C})) \\
= \; & V_\varphi(\mathsf{if}(\mathsf{gea}(b, \ulcorner \mathsf{type} \urcorner), [\![u]\!]_{\mathrm{ty}}, \mathsf{C})) \\
= \; & V_\varphi([\![\mathsf{coerce\text{-}to\text{-}type}(S([\![S(\ulcorner b \urcorner)]\!]_{\mathrm{te}}))]\!]_{\mathrm{ty}}) \\
= \; & V_\varphi([\![\mathsf{coerce\text{-}to\text{-}type}(S(b))]\!]_{\mathrm{ty}}) \\
= \; & V_\varphi([\![S(b)]\!]_{\mathrm{ty}}) \\
= \; & V_\varphi([\![S(\ulcorner e' \urcorner)]\!]_{\mathrm{ty}}) \\
= \; & V_\varphi(e') \\
= \; & V_\varphi([\![\ulcorner e' \urcorner]\!]_{\mathrm{ty}}) \\
= \; & V_\varphi([\![b]\!]_{\mathrm{ty}}) \\
= \; & V_\varphi(e).
\end{aligned}
$$

The third line is by the definition of sub; the fourth is by $H_1(\ulcorner e \urcorner)$; the fifth is by Lemma 6.3.2 and part 1 of this lemma; the sixth is by $C(\ulcorner b \urcorner)$; the seventh is by the hypothesis that $V_\varphi(\mathsf{gea}(b, \ulcorner \mathsf{type} \urcorner)) = \mathrm{T}$ and the definition of u; the eight is by $C(\ulcorner b \urcorner)$; the ninth is by $V_\varphi(\mathsf{gea}(b, \ulcorner \mathsf{type} \urcorner)) = \mathrm{T}$ and part 1 of this lemma; the tenth is by substitution of $\ulcorner e' \urcorner$ for b; the eleventh is by $C(\ulcorner e' \urcorner)$; the twelveth is by the semantics of evaluation; and the thirteenth is by substitution of b for $\ulcorner e' \urcorner$. Therefore, $C(\ulcorner e \urcorner)$ holds.

Now suppose $V_\varphi(\mathsf{gea}(b, \ulcorner \mathsf{type} \urcorner)) = \mathrm{F}$. By a similar derivation to the one above, $C(\ulcorner e \urcorner)$ holds.

Case 11: $e = [\![b]\!]_\alpha$. Let $\varphi \in \mathsf{assign}(S)$. Suppose $V_\varphi(\mathsf{gea}(b, \ulcorner \mathsf{type} \urcorner) = \mathrm{T}$ and $V_\varphi([\![b]\!]_{\mathrm{te}} \downarrow \alpha) = \mathrm{T}$. Then there is an eval-free expression e' such that $V_\varphi(b) = V_\varphi(\ulcorner e' \urcorner)$. By the definitions of sub and $\mathsf{free\text{-}in}$, $H_1(\ulcorner e \urcorner)$ and $H_2(\ulcorner e \urcorner)$ imply (1) $H_1(\ulcorner b \urcorner)$ and $H_2(\ulcorner b \urcorner)$, (2) $H_1(\ulcorner \alpha \urcorner)$

and $H_2(\ulcorner\alpha\urcorner)$, and (3) $H_1(\ulcorner e'\urcorner)$ and $H_2(\ulcorner e'\urcorner)$. By the induction hypothesis, this implies (1) $C(\ulcorner b\urcorner)$, (2) $C(\ulcorner\alpha\urcorner)$, and (3) $C(\ulcorner e'\urcorner)$. Let $u = \mathsf{coerce\text{-}to\text{-}term}(S([\![S(\ulcorner b\urcorner)]\!]_{\mathrm{te}}))$. Then

$$
\begin{aligned}
& V_\varphi([\![S(\ulcorner e\urcorner)]\!]_{k[e]}) \\
= \; & V_\varphi([\![S(\ulcorner [\![b]\!]_\alpha\urcorner)]\!]_{\mathrm{te}}) \\
= \; & V_\varphi([\![\mathsf{if}(\mathsf{syn\text{-}closed}(S(\ulcorner b\urcorner)), \\
& \quad \ulcorner\mathsf{if}(\mathsf{gea}(\lfloor S(\ulcorner b\urcorner)\rfloor, \ulcorner\alpha\urcorner) \wedge \lfloor u\rfloor \downarrow \lfloor S(\ulcorner\alpha\urcorner)\rfloor, \lfloor u\rfloor, \bot_{\mathsf{C}})\urcorner, \\
& \quad \bot_{\mathsf{C}})]\!]_{\mathrm{te}}) \\
= \; & V_\varphi([\![\ulcorner\mathsf{if}(\mathsf{gea}(\lfloor S(\ulcorner b\urcorner)\rfloor, \ulcorner\alpha\urcorner) \wedge \lfloor u\rfloor \downarrow \lfloor S(\ulcorner\alpha\urcorner)\rfloor, \lfloor u\rfloor, \bot_{\mathsf{C}})\urcorner]\!]_{\mathrm{te}}) \\
= \; & V_\varphi(\mathsf{if}(\mathsf{gea}([\![S(\ulcorner b\urcorner)]\!]_{\mathrm{te}}, \ulcorner\alpha\urcorner) \wedge [\![u]\!]_{\mathrm{te}} \downarrow [\![S(\ulcorner\alpha\urcorner)]\!]_{\mathrm{ty}}, [\![u]\!]_{\mathrm{te}}, \bot_{\mathsf{C}})) \\
= \; & V_\varphi(\mathsf{if}(\mathsf{gea}(b, \ulcorner\alpha\urcorner) \wedge [\![u]\!]_{\mathrm{te}} \downarrow \alpha, [\![u]\!]_{\mathrm{te}}, \bot_{\mathsf{C}})) \\
= \; & V_\varphi(\mathsf{if}([\![\mathsf{coerce\text{-}to\text{-}term}(S([\![S(\ulcorner b\urcorner)]\!]_{\mathrm{te}}))]\!]_{\mathrm{te}} \downarrow \alpha, \\
& \quad [\![\mathsf{coerce\text{-}to\text{-}term}(S([\![S(\ulcorner b\urcorner)]\!]_{\mathrm{te}}))]\!]_{\mathrm{te}}, \\
& \quad \bot_{\mathsf{C}})) \\
= \; & V_\varphi(\mathsf{if}([\![\mathsf{coerce\text{-}to\text{-}term}(S(b))]\!]_{\mathrm{te}} \downarrow \alpha, \\
& \quad [\![\mathsf{coerce\text{-}to\text{-}term}(S(b))]\!]_{\mathrm{te}}, \\
& \quad \bot_{\mathsf{C}})) \\
= \; & V_\varphi(\mathsf{if}([\![S(b)]\!]_{\mathrm{te}} \downarrow \alpha, [\![S(b)]\!]_{\mathrm{te}}, \bot_{\mathsf{C}})) \\
= \; & V_\varphi(\mathsf{if}([\![S(\ulcorner e'\urcorner)]\!]_{\mathrm{te}} \downarrow \alpha, [\![S(\ulcorner e'\urcorner)]\!]_{\mathrm{te}}, \bot_{\mathsf{C}})) \\
= \; & V_\varphi(\mathsf{if}(e' \downarrow \alpha, e', \bot_{\mathsf{C}})) \\
= \; & V_\varphi(\mathsf{if}([\![\ulcorner e'\urcorner]\!]_{\mathrm{te}} \downarrow \alpha, [\![\ulcorner e'\urcorner]\!]_{\mathrm{te}}, \bot_{\mathsf{C}})) \\
= \; & V_\varphi(\mathsf{if}([\![b]\!]_{\mathrm{te}} \downarrow \alpha, [\![b]\!]_{\mathrm{te}}, \bot_{\mathsf{C}})) \\
= \; & V_\varphi([\![b]\!]_\alpha) \\
= \; & V_\varphi(e).
\end{aligned}
$$

The third line is by the definition of sub; the fourth is by $H_1(\ulcorner e\urcorner)$; the fifth is by Lemma 6.3.2 and part 1 of this lemma; the sixth is by $C(\ulcorner b\urcorner)$ and $C(\ulcorner\alpha\urcorner)$; the seventh is by the hypothesis that $V_\varphi(\mathsf{gea}(b, \ulcorner\mathsf{type}\urcorner)) = \mathrm{T}$ and the definition of u; the eight is by $C(\ulcorner b\urcorner)$; the ninth is by $V_\varphi(\mathsf{gea}(b, \ulcorner\mathsf{type}\urcorner)) = \mathrm{T}$ and part 1 of this lemma; the tenth is by substitution of $\ulcorner e'\urcorner$ for b; the eleventh is by $C(\ulcorner e'\urcorner)$; the twelveth is by the semantics of evaluation; the thirteenth is by substitution of b for $\ulcorner e'\urcorner$; and the fourteenth in by $V_\varphi([\![b]\!]_{\mathrm{te}} \downarrow \alpha) = \mathrm{T}$. Therefore, $C(\ulcorner e\urcorner)$ holds.

Now suppose $V_\varphi(\mathsf{gea}(b,\ulcorner\mathsf{type}\urcorner)) = \text{F}$ or $V_\varphi([\![b]\!]_{\text{te}} \downarrow \alpha) = \text{F}$ By a similar derivation to the one above, $C(\ulcorner e\urcorner)$ holds.

Case 12: $e = [\![b]\!]_{\text{fo}}$. Similar to case 10.

□

Lemma 6.4.10 (Substitution B) *Let $M = (S, V)$ be a standard model for a normal theory $T = (L, \Gamma)$ and a be a term, x be a symbol, and e be a type, term, or formula of L. If $M \models \mathsf{sub}(\ulcorner a\urcorner, \ulcorner x\urcorner, \ulcorner e\urcorner)\downarrow$ and $M \models \mathsf{free\text{-}for}(\ulcorner a\urcorner, \ulcorner x\urcorner, \ulcorner e\urcorner)$, then*

$$V_\varphi([\![\mathsf{sub}(\ulcorner a\urcorner, \ulcorner x\urcorner, \ulcorner e\urcorner)]\!]_{k[e]}) = V_{\varphi[x\mapsto V_\varphi(a)]}(e)$$

for all $\varphi \in \mathsf{assign}(S)$ such that $V_\varphi(a) \neq \bot$.

Proof Let $S(\ulcorner e\urcorner)$ mean $\mathsf{sub}(\ulcorner a\urcorner, \ulcorner x\urcorner, \ulcorner e\urcorner)$. Assume

$$M \models S(\ulcorner e\urcorner)\downarrow \quad \text{[designated } H_1(\ulcorner e\urcorner)\text{]},$$

and

$$M \models \mathsf{free\text{-}for}(\ulcorner a\urcorner, \ulcorner x\urcorner, \ulcorner e\urcorner) \quad \text{[designated } H_2(\ulcorner e\urcorner)\text{]}.$$

We must show that

$V_\varphi([\![S(\ulcorner e\urcorner)]\!]_{k[e]}) = V_{\varphi[x\mapsto V_\varphi(a)]}(e)$ for all $\varphi \in \mathsf{assign}(S)$ such that $V_\varphi(a) \neq \bot$ [designated $C(\ulcorner e\urcorner)$].

Our proof is by induction on the complexity of e. There are 12 cases corresponding to the 12 formula schemas used to define $\mathsf{sub}(\ulcorner e_1\urcorner, \ulcorner e_2\urcorner, \ulcorner e_3\urcorner))$ when e_1 is a term, e_2 is a symbol, e_3 is a type, term, or formula.

Case 1: $e = O(e_1, \ldots, e_n)$ and $O = (o :: k_1, \ldots, k_{n+1})$. By the definitions of sub and $\mathsf{free\text{-}for}$, $H_1(\ulcorner e\urcorner)$ and $H_2(\ulcorner e\urcorner)$ imply (1) $H_1(\ulcorner k_i\urcorner)$ and $H_2(\ulcorner k_i\urcorner)$ for all i with $1 \leq i \leq n+1$ and $\mathbf{type}_L[k_i]$ and (2) $H_1(\ulcorner e_i\urcorner)$ and $H_2(\ulcorner e_i\urcorner)$ for all i with $1 \leq i \leq n$. By the induction hypothesis, this implies (1) $C(\ulcorner k_i\urcorner)$ for all i with $1 \leq i \leq n+1$ and $\mathbf{type}_L[k_i]$ and (2) $C(\ulcorner e_i\urcorner)$ for all i with $1 \leq i \leq n$. Let $\varphi \in \mathsf{assign}(S)$ such that $V_\varphi(a) \neq \bot$. Then

$$\begin{aligned}
& V_\varphi([\![S(\ulcorner e\urcorner)]\!]_{k[e]}) \\
= \; & V_\varphi([\![S(\ulcorner O(e_1,\ldots,e_n)\urcorner)]\!]_{k[e]}) \\
= \; & V_\varphi([\![\ulcorner \lfloor S(\ulcorner O\urcorner)\rfloor(\lfloor S(\ulcorner e_1\urcorner)\rfloor,\ldots,\lfloor S(\ulcorner e_n\urcorner)\rfloor)\urcorner]\!]_{k[e]}) \\
= \; & V_\varphi(\overline{S(\ulcorner O\urcorner)}([\![S(\ulcorner e_1\urcorner)]\!]_{k[e_1]},\ldots,[\![S(\ulcorner e_n\urcorner)]\!]_{k[e_n]})) \\
= \; & V_{\varphi[x\mapsto V_\varphi(a)]}(O(e_1,\ldots,e_n)) \\
= \; & V_{\varphi[x\mapsto V_\varphi(a)]}(e)
\end{aligned}$$

where

$$\overline{S(\ulcorner O \urcorner)} = (o :: \overline{S(\ulcorner k_1 \urcorner)}, \ldots, \overline{S(\ulcorner k_{n+1} \urcorner)})$$

and

$$\overline{S(\ulcorner k_i \urcorner)} = \begin{cases} [\![S(\ulcorner k_i \urcorner)]\!]_{\mathrm{ty}} & \text{if } \mathbf{type}_L[k_i] \\ k_i & \text{if } k_i = \mathsf{type} \text{ or } \mathsf{formula}. \end{cases}$$

The third line is by the definition of sub; the fourth is by Lemma 6.3.2 and part 1 of Lemma 6.4.9; and the fifth holds since (1) O and $\overline{S(\ulcorner O \urcorner)}$ are similar by part 1 of Lemma 6.4.9, (2) $C(\ulcorner k_i \urcorner)$ holds for all i with $1 \leq i \leq n+1$ and $\mathbf{type}_L[k_i]$, and (3) $C(\ulcorner e_i \urcorner)$ holds for all i with $1 \leq i \leq n$. Therefore, $C(\ulcorner e \urcorner)$ holds.

Case 2: $e = (x : \alpha)$. By the definitions of free-for and free-for, $H_1(\ulcorner e \urcorner)$ and $H_2(\ulcorner e \urcorner)$ imply $H_1(\ulcorner \alpha \urcorner)$, $H_2(\ulcorner \alpha \urcorner)$, and $M \models \mathsf{cleanse}(\ulcorner e' \urcorner)\downarrow$. By the induction hypothesis, this implies $C(\ulcorner \alpha \urcorner)$. Let $\varphi \in \mathsf{assign}(S)$ such that $V_\varphi(a) \neq \bot$. Then

$$\begin{aligned} & V_\varphi([\![S(\ulcorner e \urcorner)]\!]_{k[e]}) \\ = \; & V_\varphi([\![S(\ulcorner (x : \alpha) \urcorner)]\!]_{\mathrm{te}}) \\ = \; & V_\varphi([\![\mathsf{if}(a \downarrow [\![S(\alpha)]\!]_{\mathrm{ty}}, \mathsf{cleanse}(\ulcorner a \urcorner), \ulcorner \bot_{\mathsf{C}} \urcorner)]\!]_{\mathrm{te}}) \\ = \; & V_\varphi(\mathsf{if}(a \downarrow [\![S(\alpha)]\!]_{\mathrm{ty}}, [\![\mathsf{cleanse}(\ulcorner a \urcorner)]\!]_{\mathrm{te}}, \bot_{\mathsf{C}})) \\ = \; & V_\varphi(\mathsf{if}(a \downarrow [\![S(\alpha)]\!]_{\mathrm{ty}}, a, \bot_{\mathsf{C}})) \\ = \; & V_{\varphi[x \mapsto V_\varphi(a)]}(\mathsf{if}((x : \mathsf{C}) \downarrow \alpha, (x : \mathsf{C}), \bot_{\mathsf{C}})) \\ = \; & V_{\varphi[x \mapsto V_\varphi(a)]}((x : \alpha)) \\ = \; & V_{\varphi[x \mapsto V_\varphi(a)]}(e). \end{aligned}$$

The third line is by the definition of sub; the fourth is by Lemma 6.3.2 and part 1 of Lemma 6.4.8; the fifth is by $M \models \mathsf{cleanse}(\ulcorner e' \urcorner)\downarrow$ and part 4 of Lemma 6.4.8; the sixth is by $C(\ulcorner \alpha \urcorner)$; and the seventh is by the definition of the standard valuation on variables. Therefore, $C(\ulcorner e \urcorner)$ holds.

Case 3: $e = (y : \alpha)$ where $x \neq y$. By the definitions of sub and free-for, $H_1(\ulcorner e \urcorner)$ and $H_2(\ulcorner e \urcorner)$ imply $H_1(\ulcorner \alpha \urcorner)$ and $H_2(\ulcorner \alpha \urcorner)$. By the

induction hypothesis, this implies $C(\ulcorner \alpha \urcorner)$. Let $\varphi \in \mathsf{assign}(S)$ such that $V_\varphi(a) \neq \bot$. Then

$$\begin{aligned}
& V_\varphi([\![S(\ulcorner e \urcorner)]\!]_{k[e]}) \\
= \; & V_\varphi([\![S(\ulcorner (y : \alpha) \urcorner)]\!]_{\mathrm{te}}) \\
= \; & V_\varphi([\![\ulcorner (y : \lfloor S(\alpha) \rfloor) \urcorner]\!]_{\mathrm{te}}) \\
= \; & V_\varphi((y : [\![S(\ulcorner \alpha \urcorner)]\!]_{\mathrm{ty}})) \\
= \; & V_{\varphi[x \mapsto V_\varphi(a)]}((y : \alpha)) \\
= \; & V_{\varphi[x \mapsto V_\varphi(a)]}(e).
\end{aligned}$$

The third line is by the definition of sub; the fourth is by Lemma 6.3.2 and part 1 of Lemma 6.4.9; and the fifth is by $C(\ulcorner \alpha \urcorner)$. Therefore, $C(\ulcorner e \urcorner)$ holds.

Case 4: $e = (\star x : \alpha \,.\, e')$ and $\star$ is Λ, λ, $\exists$, ι, or ϵ. By the definitions of sub and $\mathsf{free\text{-}for}$, $H_1(\ulcorner e \urcorner)$ and $H_2(\ulcorner e \urcorner)$ imply $H_1(\ulcorner \alpha \urcorner)$, $H_2(\ulcorner \alpha \urcorner)$, and $M \models \mathsf{cleanse}(\ulcorner e' \urcorner)\downarrow$. By the induction hypothesis, this implies $C(\ulcorner \alpha \urcorner)$. Let $\varphi \in \mathsf{assign}(S)$ such that $V_\varphi(a) \neq \bot$. Then

$$\begin{aligned}
& V_\varphi([\![S(\ulcorner e \urcorner)]\!]_{k[e]}) \\
= \; & V_\varphi([\![S(\ulcorner (\star x : \alpha \,.\, e') \urcorner)]\!]_{k[e]}) \\
= \; & V_\varphi([\![\ulcorner (\star x : \lfloor S(\ulcorner \alpha \urcorner) \rfloor \,.\, \lfloor \mathsf{cleanse}(\ulcorner e' \urcorner) \rfloor) \urcorner]\!]_{k[e]}) \\
= \; & V_\varphi((\star x : [\![S(\ulcorner \alpha \urcorner)]\!]_{\mathrm{ty}} \,.\, [\![\mathsf{cleanse}(\ulcorner e' \urcorner)]\!]_{k[e']})) \\
= \; & V_\varphi((\star x : [\![S(\ulcorner \alpha \urcorner)]\!]_{\mathrm{ty}} \,.\, e')) \\
= \; & V_{\varphi[x \mapsto V_\varphi(a)]}((\star x : \alpha \,.\, e')) \\
= \; & V_{\varphi[x \mapsto V_\varphi(a)]}(e).
\end{aligned}$$

The third line is by the definition of sub; the fourth is by Lemma 6.3.2, part 1 of Lemma 6.4.9, and parts 1 and 2 of Lemma 6.4.8; the fifth is by $M \models \mathsf{cleanse}(\ulcorner e' \urcorner)\downarrow$ and part 4 of Lemma 6.4.8; and the sixth is by $C(\ulcorner \alpha \urcorner)$ and the fact that

$$V_{\varphi[x \mapsto d]}(e') = V_{\varphi[x \mapsto V_\varphi(a)][x \mapsto d]}(e')$$

for all $d \in D_{\mathrm{c}}$. Therefore, $C(\ulcorner e \urcorner)$ holds.

Case 5: $e = (\star y : \alpha \,.\, e')$, $x \neq y$, and $\star$ is Λ, λ, $\exists$, ι, or ϵ. By the definitions of sub and $\mathsf{free\text{-}for}$, $H_1(\ulcorner e \urcorner)$ and $H_2(\ulcorner e \urcorner)$ imply (1) $H_1(\ulcorner \alpha \urcorner)$

and $H_2(\ulcorner\alpha\urcorner)$, (2) either $(*)$ $M \models \neg\mathsf{free\text{-}in}(\ulcorner x\urcorner, \ulcorner e'\urcorner)$ or $(**)$ $M \models \neg\mathsf{free\text{-}in}(\ulcorner y\urcorner, \ulcorner a\urcorner)$, and (3) $H_1(\ulcorner e'\urcorner)$ and $H_2(\ulcorner e'\urcorner)$. By the induction hypothesis, this implies $C(\ulcorner\alpha\urcorner)$ and $C(\ulcorner e'\urcorner)$. Let $\varphi \in \mathsf{assign}(S)$ such that $V_\varphi(a) \neq \bot$. Then

$$\begin{aligned}
& V_\varphi([\![S(\ulcorner e\urcorner)]\!]_{k[e]}) \\
= \; & V_\varphi([\![S(\ulcorner(\star\, y : \alpha \,.\, e')\urcorner)]\!]_{k[e]}) \\
= \; & V_\varphi([\![\ulcorner(\star\, y : \lfloor S(\ulcorner\alpha\urcorner)\rfloor \,.\, \lfloor S(\ulcorner e'\urcorner)\rfloor\urcorner]\!]_{k[e]})) \\
= \; & V_\varphi((\star\, y : [\![S(\ulcorner\alpha\urcorner)]\!]_{\mathrm{ty}} \,.\, [\![S(\ulcorner e'\urcorner)]\!]_{k[e']})) \\
= \; & V_{\varphi[x\mapsto V_\varphi(a)]}((\star\, y : \alpha \,.\, e')) \\
= \; & V_{\varphi[x\mapsto V_\varphi(a)]}(e).
\end{aligned}$$

The third line is by the definition of sub; the fourth is by Lemma 6.3.2 and parts 1 and 2 of Lemma 6.4.9; and the fifth is by $C(\ulcorner\alpha\urcorner)$ and separate arguments for the two cases $(*)$ and $(**)$. In case $(*)$,

$$\begin{aligned}
& V_{\varphi[y\mapsto d]}([\![S(\ulcorner e'\urcorner)]\!]_{k[e]}) \\
= \; & V_{\varphi[y\mapsto d]}(e') \\
= \; & V_{\varphi[y\mapsto d][x\mapsto V_\varphi(a)]}(e') \\
= \; & V_{\varphi[x\mapsto V_\varphi(a)][y\mapsto d]}(e')
\end{aligned}$$

for all $d \in D_{\mathrm{c}}$. The second line is by $(*)$, $H_1(\ulcorner e'\urcorner)$, and part 4 of Lemma 6.4.9, and the third is by $(*)$ and part 1 of Lemma 6.4.5. In case $(**)$,

$$\begin{aligned}
& V_{\varphi[y\mapsto d]}([\![S(\ulcorner e'\urcorner)]\!]_{k[e]}) \\
= \; & V_{\varphi[y\mapsto d][x\mapsto V_{\varphi[y\mapsto d]}(a)]}(e') \\
= \; & V_{\varphi[y\mapsto d][x\mapsto V_\varphi(a)]}(e') \\
= \; & V_{\varphi[x\mapsto V_\varphi(a)][y\mapsto d]}(e')
\end{aligned}$$

for all $d \in D_{\mathrm{c}}$. The second line is by $C(\ulcorner e'\urcorner)$, and the third is by $(**)$ and part 1 of Lemma 6.4.5. Therefore, $C(\ulcorner e\urcorner)$ holds.

Case 6: $e = \alpha(a)$. Similar to case 3.

Case 7: $e = f(a)$. Similar to case 3.

Case 8: $e = \mathsf{if}(A, b, c)$. Similar to case 3.

Case 9: $e = \ulcorner e' \urcorner$. Let $\varphi \in \mathsf{assign}(S)$ such that $V_\varphi(a) \neq \bot$. Then

$$\begin{aligned}
& V_\varphi([\![S(\ulcorner e \urcorner)]\!]_{k[e]}) \\
= \; & V_\varphi([\![S(\ulcorner \ulcorner e' \urcorner \urcorner)]\!]_{\mathrm{te}}) \\
= \; & V_\varphi([\![\ulcorner \ulcorner e' \urcorner \urcorner]\!]_{\mathrm{te}}) \\
= \; & V_\varphi(\ulcorner e' \urcorner) \\
= \; & V_{\varphi[x \mapsto V_\varphi(a)]}(\ulcorner e' \urcorner) \\
= \; & V_{\varphi[x \mapsto V_\varphi(a)]}(e)
\end{aligned}$$

The third line is by the definition of sub; the fourth is by Lemma 6.3.1; and the fifth is by the fact that $V_\varphi(e)$ does not depend on φ. Therefore, $C(\ulcorner e \urcorner)$ holds.

Case 10: $e = [\![b]\!]_{\mathrm{ty}}$. Let $\varphi \in \mathsf{assign}(S)$ such that $V_\varphi(a) \neq \bot$. Suppose $V_\varphi(\mathsf{gea}(\ulcorner H^{-1}(V_{\varphi[x \mapsto V_\varphi(a)]}(b)) \urcorner, \ulcorner \mathsf{type} \urcorner)) = \mathrm{T}$. By the definitions of sub and $\mathsf{free\text{-}for}$, $H_1(\ulcorner e \urcorner)$ and $H_2(\ulcorner e \urcorner)$ imply (1) $H_1(\ulcorner b \urcorner)$ and $H_2(\ulcorner b \urcorner)$ and (2) $H_1([\![S(\ulcorner b \urcorner)]\!]_{\mathrm{te}})$ and $H_2([\![S(\ulcorner b \urcorner)]\!]_{\mathrm{te}})$. By the induction hypothesis, this implies (1) $C(\ulcorner b \urcorner)$ and (2) $C([\![S(\ulcorner b \urcorner)]\!]_{\mathrm{te}})$. Let $u = \mathsf{coerce\text{-}to\text{-}type}(S([\![S(\ulcorner b \urcorner)]\!]_{\mathrm{te}}))$. Then

$$\begin{aligned}
& V_\varphi([\![S(\ulcorner e \urcorner)]\!]_{k[e]}) \\
= \; & V_\varphi([\![S(\ulcorner [\![b]\!]_{\mathrm{ty}} \urcorner)]\!]_{\mathrm{ty}}) \\
= \; & V_\varphi([\![\mathsf{if}(\mathsf{syn\text{-}closed}(S(\ulcorner b \urcorner)), \\
& \quad \ulcorner \mathsf{if}(\mathsf{gea}(\lfloor S(\ulcorner b \urcorner) \rfloor, \ulcorner \mathsf{type} \urcorner), \lfloor u \rfloor, \mathsf{C}) \urcorner, \\
& \quad \bot_{\mathsf{C}})]\!]_{\mathrm{ty}}) \\
= \; & V_\varphi([\![\ulcorner \mathsf{if}(\mathsf{gea}(\lfloor S(\ulcorner b \urcorner) \rfloor, \ulcorner \mathsf{type} \urcorner), \lfloor u \rfloor, \mathsf{C}) \urcorner]\!]_{\mathrm{ty}}) \\
= \; & V_\varphi(\mathsf{if}(\mathsf{gea}([\![S(\ulcorner b \urcorner)]\!]_{\mathrm{te}}, \ulcorner \mathsf{type} \urcorner), [\![u]\!]_{\mathrm{ty}}, \mathsf{C})) \\
= \; & V_\varphi(\mathsf{if}(\mathsf{gea}(\ulcorner H^{-1}(V_{\varphi[x \mapsto V_\varphi(a)]}(b)) \urcorner, \ulcorner \mathsf{type} \urcorner), [\![u]\!]_{\mathrm{ty}}, \mathsf{C})) \\
= \; & V_\varphi([\![\mathsf{coerce\text{-}to\text{-}type}(S([\![S(\ulcorner b \urcorner)]\!]_{\mathrm{te}}))]\!]_{\mathrm{ty}}) \\
= \; & V_\varphi([\![\mathsf{coerce\text{-}to\text{-}type}(S(\ulcorner H^{-1}(V_{\varphi[x \mapsto V_\varphi(a)]}(b)) \urcorner))]\!]_{\mathrm{ty}}) \\
= \; & V_\varphi([\![S(\ulcorner H^{-1}(V_{\varphi[x \mapsto V_\varphi(a)]}(b)) \urcorner)]\!]_{\mathrm{ty}}) \\
= \; & V_{\varphi[x \mapsto V_\varphi(a)]}(H^{-1}(V_{\varphi[x \mapsto V_\varphi(a)]}(b))) \\
= \; & V_{\varphi[x \mapsto V_\varphi(a)]}([\![\ulcorner H^{-1}(V_{\varphi[x \mapsto V_\varphi(a)]}(b)) \urcorner]\!]_{\mathrm{ty}}) \\
= \; & V_{\varphi[x \mapsto V_\varphi(a)]}([\![b]\!]_{\mathrm{ty}}) \\
= \; & V_{\varphi[x \mapsto V_\varphi(a)]}(e).
\end{aligned}$$

The third line is by the definition of sub; the fourth is by $H_1(\ulcorner e \urcorner)$; the fifth is by Lemma 6.3.2 and part 1 of this lemma; the sixth is by $C(\ulcorner b \urcorner)$; the seventh is by the hypothesis that $V_\varphi(\mathsf{gea}(\ulcorner H^{-1}(V_{\varphi[x \mapsto V_\varphi(a)]}(b)) \urcorner, \ulcorner \mathsf{type} \urcorner)) = \mathrm{T}$ and the definition of u; the eight is by $C(\ulcorner b \urcorner)$; the ninth is by $V_\varphi(\mathsf{gea}(\ulcorner H^{-1}(V_{\varphi[x \mapsto V_\varphi(a)]}(b)) \urcorner, \ulcorner \mathsf{type} \urcorner)) = \mathrm{T}$ and part 1 of this lemma; the tenth is by $C([\![S(\ulcorner b \urcorner)]\!]_{\mathrm{te}})$ since

$$V_\varphi([\![S(\ulcorner b \urcorner)]\!]_{\mathrm{te}}) = V_{\varphi[x \mapsto V_\varphi(a)]}(b) = V_\varphi(\ulcorner H^{-1}(V_{\varphi[x \mapsto V_\varphi(a)]}(b)) \urcorner)$$

by $C(\ulcorner b \urcorner)$; the eleventh is by the semantics of evaluation; and the twelveth is by

$$V_{\varphi[x \mapsto V_\varphi(a)]}([\![\ulcorner H^{-1}(V_{\varphi[x \mapsto V_\varphi(a)]}(b)) \urcorner]\!]_{\mathrm{ty}}) = V_{\varphi[x \mapsto V_\varphi(a)]}(b).$$

Therefore, $C(\ulcorner e \urcorner)$ holds.

Now suppose $V_\varphi(\mathsf{gea}(\ulcorner H^{-1}(V_{\varphi[x \mapsto V_\varphi(a)]}(b)) \urcorner, \ulcorner \mathsf{type} \urcorner)) = \mathrm{F}$. By a similar derivation to the one above, $C(\ulcorner e \urcorner)$ holds.

Case 11: $e = [\![b]\!]_\alpha$. Let $\varphi \in \mathsf{assign}(S)$ such that $V_\varphi(a) \neq \bot$. Suppose $V_\varphi(\mathsf{gea}(\ulcorner H^{-1}(V_{\varphi[x \mapsto V_\varphi(a)]}(b)) \urcorner, \ulcorner \mathsf{type} \urcorner)) = \mathrm{T}$ and $V_\varphi([\![b]\!]_{\mathrm{te}} \downarrow \alpha) = \mathrm{T}$. By the definitions of sub and $\mathsf{free\text{-}for}$, $H_1(\ulcorner e \urcorner)$ and $H_2(\ulcorner e \urcorner)$ imply (1) $H_1(\ulcorner b \urcorner)$ and $H_2(\ulcorner b \urcorner)$, (2) $H_1(\ulcorner \alpha \urcorner)$ and $H_2(\ulcorner \alpha \urcorner)$, and (3) $H_1(\ulcorner e' \urcorner)$ and $H_2(\ulcorner e' \urcorner)$. By the induction hypothesis, this implies (1) $C(\ulcorner b \urcorner)$, (2) $C(\ulcorner \alpha \urcorner)$, and (3) $C([\![S(\ulcorner b \urcorner)]\!]_{\mathrm{te}})$. Let $u = \mathsf{coerce\text{-}to\text{-}term}(S([\![S(\ulcorner b \urcorner)]\!]_{\mathrm{te}}))$. Then

$$\begin{aligned}
& V_\varphi([\![S(\ulcorner e \urcorner)]\!]_{k[e]}) \\
= \; & V_\varphi([\![S(\ulcorner [\![b]\!]_\alpha \urcorner)]\!]_{\mathrm{te}}) \\
= \; & V_\varphi([\![\mathsf{if}(\mathsf{syn\text{-}closed}(S(\ulcorner b \urcorner)), \\
& \quad \ulcorner \mathsf{if}(\mathsf{gea}(\lfloor S(\ulcorner b \urcorner) \rfloor, \ulcorner \alpha \urcorner) \wedge \lfloor u \rfloor \downarrow \lfloor S(\ulcorner \alpha \urcorner) \rfloor, \lfloor u \rfloor, \bot_{\mathsf{C}}) \urcorner, \\
& \quad \bot_{\mathsf{C}})]\!]_{\mathrm{te}}) \\
= \; & V_\varphi([\![\ulcorner \mathsf{if}(\mathsf{gea}(\lfloor S(\ulcorner b \urcorner) \rfloor, \ulcorner \alpha \urcorner) \wedge \lfloor u \rfloor \downarrow \lfloor S(\ulcorner \alpha \urcorner) \rfloor, \lfloor u \rfloor, \bot_{\mathsf{C}}) \urcorner]\!]_{\mathrm{te}}) \\
= \; & V_\varphi(\mathsf{if}(\mathsf{gea}([\![S(\ulcorner b \urcorner)]\!]_{\mathrm{te}}, \ulcorner \alpha \urcorner) \wedge [\![u]\!]_{\mathrm{te}} \downarrow [\![S(\ulcorner \alpha \urcorner)]\!]_{\mathrm{ty}}, [\![u]\!]_{\mathrm{te}}, \bot_{\mathsf{C}})) \\
= \; & V_\varphi(\mathsf{if}(\mathsf{gea}(\ulcorner H^{-1}(V_{\varphi[x \mapsto V_\varphi(a)]}(b)) \urcorner, \ulcorner \alpha \urcorner) \wedge \\
& \quad [\![u]\!]_{\mathrm{te}} \downarrow [\![S(\ulcorner \alpha \urcorner)]\!]_{\mathrm{ty}}, \\
& \quad [\![u]\!]_{\mathrm{te}}, \\
& \quad \bot_{\mathsf{C}}))
\end{aligned}$$

$$
\begin{aligned}
&= V_\varphi(\mathsf{if}(\llbracket \mathsf{coerce\text{-}to\text{-}term}(S(\llbracket S(\ulcorner b \urcorner)\rrbracket_{\mathrm{te}}))\rrbracket_{\mathrm{te}} \downarrow \llbracket S(\ulcorner \alpha \urcorner)\rrbracket_{\mathrm{ty}},\\
&\qquad \llbracket \mathsf{coerce\text{-}to\text{-}term}(S(\llbracket S(\ulcorner b \urcorner)\rrbracket_{\mathrm{te}}))\rrbracket_{\mathrm{te}},\\
&\qquad \bot_{\mathsf{C}}))\\
&= V_\varphi(\mathsf{if}(\llbracket \mathsf{coerce\text{-}to\text{-}term}(S(\ulcorner H^{-1}(V_{\varphi[x\mapsto V_\varphi(a)]}(b))\urcorner))\rrbracket_{\mathrm{te}} \downarrow\\
&\qquad \llbracket S(\ulcorner \alpha \urcorner)\rrbracket_{\mathrm{ty}},\\
&\qquad \llbracket \mathsf{coerce\text{-}to\text{-}term}(S(\ulcorner H^{-1}(V_{\varphi[x\mapsto V_\varphi(a)]}(b))\urcorner))\rrbracket_{\mathrm{te}},\\
&\qquad \bot_{\mathsf{C}}))\\
&= V_\varphi(\mathsf{if}(\llbracket S(\ulcorner H^{-1}(V_{\varphi[x\mapsto V_\varphi(a)]}(b))\urcorner)\rrbracket_{\mathrm{te}} \downarrow \llbracket S(\ulcorner \alpha \urcorner)\rrbracket_{\mathrm{ty}},\\
&\qquad \llbracket S(\ulcorner H^{-1}(V_{\varphi[x\mapsto V_\varphi(a)]}(b))\urcorner)\rrbracket_{\mathrm{te}},\\
&\qquad \bot_{\mathsf{C}}))\\
&= V_{\varphi[x\mapsto V_\varphi(a)]}(\mathsf{if}(H^{-1}(V_{\varphi[x\mapsto V_\varphi(a)]}(b)) \downarrow \alpha,\\
&\qquad H^{-1}(V_{\varphi[x\mapsto V_\varphi(a)]}(b)),\\
&\qquad \bot_{\mathsf{C}}))\\
&= V_{\varphi[x\mapsto V_\varphi(a)]}(\mathsf{if}(\llbracket \ulcorner H^{-1}(V_{\varphi[x\mapsto V_\varphi(a)]}(b))\urcorner\rrbracket_{\mathrm{te}} \downarrow \alpha,\\
&\qquad \llbracket \ulcorner H^{-1}(V_{\varphi[x\mapsto V_\varphi(a)]}(b))\urcorner\rrbracket_{\mathrm{te}},\\
&\qquad \bot_{\mathsf{C}}))\\
&= V_{\varphi[x\mapsto V_\varphi(a)]}(\mathsf{if}(\llbracket b\rrbracket_{\mathrm{te}} \downarrow \alpha, \llbracket b\rrbracket_{\mathrm{te}}, \bot_{\mathsf{C}}))\\
&= V_{\varphi[x\mapsto V_\varphi(a)]}(\llbracket b\rrbracket_\alpha)\\
&= V_{\varphi[x\mapsto V_\varphi(a)]}(e).
\end{aligned}
$$

The third line is by the definition of sub; the fourth is by $H_1(\ulcorner e \urcorner)$; the fifth is by Lemma 6.3.2 and part 1 of this lemma; the sixth is by $C(\ulcorner b \urcorner)$; the seventh is by the hypothesis that $V_\varphi(\mathsf{gea}(\ulcorner H^{-1}(V_{\varphi[x\mapsto V_\varphi(a)]}(b))\urcorner, \ulcorner \mathsf{type} \urcorner)) = \mathrm{T}$ and the definition of u; the eight is by $C(\ulcorner b \urcorner)$; the ninth is by $V_\varphi(\mathsf{gea}(\ulcorner H^{-1}(V_{\varphi[x\mapsto V_\varphi(a)]}(b))\urcorner, \ulcorner \mathsf{type} \urcorner)) = \mathrm{T}$ and part 1 of this lemma; the tenth is by $C(\ulcorner \alpha \urcorner)$ and $C(\llbracket S(\ulcorner b \urcorner)\rrbracket_{\mathrm{te}})$ since

$$V_\varphi(\llbracket S(\ulcorner b \urcorner)\rrbracket_{\mathrm{te}}) = V_{\varphi[x\mapsto V_\varphi(a)]}(b) = V_\varphi(\ulcorner H^{-1}(V_{\varphi[x\mapsto V_\varphi(a)]}(b))\urcorner)$$

by $C(\ulcorner b \urcorner)$; the eleventh is by the semantics of evaluation; and the twelveth is by

$$V_{\varphi[x\mapsto V_\varphi(a)]}(\llbracket \ulcorner H^{-1}(V_{\varphi[x\mapsto V_\varphi(a)]}(b))\urcorner\rrbracket_{\mathrm{te}}) = V_{\varphi[x\mapsto V_\varphi(a)]}(b).$$

Therefore, $C(\ulcorner e \urcorner)$ holds.

Now suppose $V_\varphi(\mathsf{gea}(\ulcorner H^{-1}(V_{\varphi[x \mapsto V_\varphi(a)]}(b))\urcorner, \ulcorner \mathsf{type} \urcorner)) = \text{F}$ or $V_\varphi([\![b]\!]_{\text{te}} \downarrow \alpha) = \text{F}$. By a similar derivation to the one above, $C(\ulcorner e \urcorner)$ holds.

Case 12: $e = [\![b]\!]_{\text{fo}}$. Similar to case 10.

□

6.5 More Notational Definitions

Using some of the operators defined in this section and the previous section, we give some notational definitions for additional variable binders. Let L be a normal language.

1. **Class Abstraction**

   $(\mathrm{C}\, u : \alpha \,.\, A)$ means

   $$(\iota\, x : \mathsf{power\text{-}type}(\alpha) \,.\, (\forall\, u : \alpha \,.\, u \in x \equiv A))$$

   where x is some member of $\mathcal{S}$ such that

   $$T^L_{\text{ker}} \models \neg \mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner \alpha \urcorner)$$

   and

   $$T^L_{\text{ker}} \models \neg \mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner A \urcorner).$$

   Compact notation:

   $\{a_1, \ldots, a_n\}$ means $(\mathrm{C}\, u : \mathsf{V} \,.\, u = a_1 \vee \cdots \vee u = a_n)$ for $n \geq 0$.

2. **Dependent Type Product**

   $(\otimes\, x : \alpha \,.\, \beta)$ means

   $$\mathsf{type}((\mathrm{C}\, p : \mathsf{V} \times \mathsf{V} \,.\, \exists\, x : \alpha \,.\, \exists\, y : \beta \,.\, p = \langle x, y \rangle))$$

   where p is some member of $\mathcal{S}$ such that

   $$T^L_{\text{ker}} \models \neg \mathsf{free\text{-}in}(\ulcorner p \urcorner, \ulcorner \alpha \urcorner)$$

   and

   $$T^L_{\text{ker}} \models \neg \mathsf{free\text{-}in}(\ulcorner p \urcorner, \ulcorner \beta \urcorner).$$

3. **Unique Existential Quantification**

   $(\exists !\, x : \alpha \,.\, A)$ means

   $$(\exists\, x : \alpha \,.\, (A \wedge (\forall\, y : \alpha \,.\, [\![\mathsf{sub}(\ulcorner (y : \alpha)\urcorner, \ulcorner x\urcorner, \ulcorner A\urcorner)]\!]_{\mathrm{fo}} \supset y = x)))$$

   where y is some member of $\mathcal{S}$ such that

   $$T^L_{\mathrm{ker}} \models \neg\mathsf{free\text{-}in}(\ulcorner y\urcorner, \ulcorner A\urcorner),$$

   $$T^L_{\mathrm{ker}} \models \mathsf{sub}(\ulcorner (y : \alpha)\urcorner, \ulcorner x\urcorner, \ulcorner A\urcorner)\downarrow,$$

   and

   $$T^L_{\mathrm{ker}} \models \mathsf{free\text{-}for}(\ulcorner (y : \alpha)\urcorner, \ulcorner x\urcorner, \ulcorner A\urcorner).$$

7 Examples

7.1 Law of Excluded Middle

In many traditional logics, e.g., first-order logic, the law of excluded middle (LEM) is expressed as a formula schema

$$A \vee \neg A$$

where A can be any formula. In Chiron LEM can be expressed as a single formula:

$$\forall\, e : \mathsf{E}_{\mathrm{fo}} \,.\, [\![e]\!] \vee \neg [\![e]\!].$$

Using quasiquotation, LEM can alternately be expressed as the following formula:

$$\forall\, e : \mathsf{E}_{\mathrm{fo}} \,.\, \mathsf{is\text{-}eval\text{-}free}(e) \supset [\![\ulcorner \lfloor e \rfloor \vee \neg \lfloor e \rfloor \urcorner]\!]_{\mathrm{fo}}.$$

7.2 Infinite Dependency 1

The value of a proper expression may depend on the values that are assigned to infinitely many symbols. For example, let A is the formula

$$\forall z : \mathsf{E}_{\mathrm{sy}} \,.\, [\![\ulcorner (\lfloor (z : \mathsf{E}_{\mathrm{sy}}) \rfloor : \mathsf{E}_{\mathrm{sy}})\urcorner]\!]_{\mathsf{E}_{\mathrm{sy}}} = \ulcorner z\urcorner.$$

Let $M = (S, V)$ be a standard model of T and $\varphi \in \mathsf{assign}(S)$. Then $V_\varphi(A) = \mathrm{T}$ iff $\varphi(x) = H(z)$ for all $x \in \mathcal{S}$ with $x \neq z$. Moreover, neither $T^L_{\mathrm{ker}} \models \mathsf{free\text{-}in}(\ulcorner x\urcorner, \ulcorner A\urcorner)$ nor $T^L_{\mathrm{ker}} \models \neg\mathsf{free\text{-}in}(\ulcorner x\urcorner, \ulcorner A\urcorner)$ when $x \neq z$. Hence A is not syntactically closed.

A more inclusive definition of free-in could be defined by adding an argument to free-in that represents the local context [18] of its second argument.

7.3 Infinite Dependency 2

Here is another example of a proper expression that depends on the values that are assigned to infinitely many symbols. Let A is the formula

$$\mathsf{infinite}(\mathrm{C}\, e : \mathsf{E}_{\mathrm{sy}} \,.\, [\![\ulcorner(\lfloor e \rfloor : \mathsf{C})\urcorner]\!]_{\mathrm{te}} = \emptyset).$$

Let $M = (S, V)$ be a standard model of T and $\varphi \in \mathsf{assign}(S)$. Then $V_\varphi(A) = \mathrm{T}$ iff $\varphi(x)$ is the empty set for infinitely many $x \in \mathcal{S}$. This implies $V_\varphi(A) = V_{\varphi[x \mapsto d]}(A)$ for all $\varphi \in \mathsf{assign}(S)$, $x \in \mathcal{S}$, and $d \in D_{\mathrm{c}}$. Neither $T^L_{\mathrm{ker}} \models \mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner A \urcorner)$ nor $T^L_{\mathrm{ker}} \models \neg\mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner A \urcorner)$ when $x \in \mathcal{S}$. Hence A is not syntactically closed.

7.4 Conjunction

In subsection 5.1 we defined the operator and for conjunction using an infinite set of syntactically closed eval-free formulas. We can alternately define conjunction as a single formula A:

$$\forall\, p : \mathsf{E}_{\mathrm{fo}} \times \mathsf{E}_{\mathrm{fo}} \,.\, (\mathsf{and} :: \mathsf{formula}, \mathsf{formula}, \mathsf{formula})([\![\mathsf{hd}(p)]\!]_{\mathrm{fo}}, [\![\mathsf{tl}(p)]\!]_{\mathrm{fo}}) \equiv \neg(\neg[\![\mathsf{hd}(p)]\!]_{\mathrm{fo}} \vee \neg[\![\mathsf{tl}(p)]\!]_{\mathrm{fo}}).$$

A is obviously not eval-free, and neither $T^L_{\mathrm{ker}} \models \mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner A \urcorner)$ nor $T^L_{\mathrm{ker}} \models \neg\mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner A \urcorner)$ when $x \neq p$. Hence A is not syntactically closed.

The following is an alternative form for this formula with two universally quantified variables instead of one:

$$\forall\, e_1, e_2 : \mathsf{E}_{\mathrm{fo}} \,.\, (\mathsf{and} :: \mathsf{formula}, \mathsf{formula}, \mathsf{formula})([\![e_1]\!], [\![e_2]\!]) \equiv \neg(\neg[\![e_1]\!] \vee \neg[\![e_2]\!]).$$

This latter form is more natural, but it cannot be instantiated using Lemma 6.4.10 like the former form.

7.5 Modus Ponens

Like the law of excluded middle, other laws of logic are usually expressed as formula schemas. In Chiron these laws can be expressed as single formulas. For example, the following Chiron formula expresses the law of modus ponens:

$$\forall\, p : \mathsf{E}_{\mathrm{fo}} \times \mathsf{E}_{\mathrm{fo}} \,. \\ ([\![\mathsf{hd}(p)]\!]_{\mathrm{fo}} \wedge [\![\mathsf{tl}(p)]\!]_{\mathrm{fo}} \wedge \mathsf{is\text{-}impl}(\mathsf{tl}(p)) \wedge \mathsf{hd}(p) = \mathsf{1st\text{-}arg}(\mathsf{tl}(p))) \supset \\ [\![\mathsf{2nd\text{-}arg}(\mathsf{tl}(p))]\!]_{\mathrm{fo}}.$$

Deduction and computation rules are naturally represented as *transformers* [10], algorithms that map expressions to expressions. Transformers can be directly formalized in Chiron. For example, the following function abstraction, which maps a pair of formulas to a formula, formalizes the modus ponens rule of inference:

$$\lambda\, p : \mathsf{E}_{\mathrm{fo}} \times \mathsf{E}_{\mathrm{fo}} \,.\; \mathsf{if}(\mathsf{is\text{-}impl}(\mathsf{tl}(p)) \wedge \mathsf{hd}(p) = \mathsf{1st\text{-}arg}(\mathsf{tl}(p)), \mathsf{2nd\text{-}arg}(\mathsf{tl}(p)), \bot_{\mathsf{C}}).$$

Let $(\mathsf{modus\text{-}ponens} :: (\mathsf{E}_{\mathrm{fo}} \times \mathsf{E}_{\mathrm{fo}}) \to \mathsf{E}_{\mathrm{fo}})(\,)$ be a constant that is defined to be this function abstraction, and let $\mathsf{mp}(a)$ mean

$$(\mathsf{modus\text{-}ponens} :: (\mathsf{E}_{\mathrm{fo}} \times \mathsf{E}_{\mathrm{fo}}) \to \mathsf{E}_{\mathrm{fo}})(\,)(a).$$

Then the following formula is an alternate expression of the law of modus ponens:

$$\forall\, p : \mathsf{E}_{\mathrm{fo}} \times \mathsf{E}_{\mathrm{fo}} \,.\; (\llbracket \mathsf{hd}(p) \rrbracket_{\mathrm{fo}} \wedge \llbracket \mathsf{tl}(p) \rrbracket_{\mathrm{fo}} \wedge \mathsf{mp}(p)\!\downarrow) \supset \llbracket \mathsf{mp}(p) \rrbracket_{\mathrm{fo}}.$$

7.6 Beta Reduction

There are two laws of beta reduction in Chiron, one for the application of a dependent function type and one for the application of a function abstraction. Without quotation and evaluation, the latter beta reduction law would be informally expressed as the formula schema

$$(\lambda\, x : \alpha \,.\, b)(a) \simeq b[x \mapsto a]$$

provided:

1. $a \downarrow (\alpha \cap \mathsf{V})$.
2. $b[x \mapsto a] \downarrow \mathsf{V}$.
3. a is free for $(x : \alpha)$ in b.

Notice that this schema includes four schema variables (x, α, b, a), a substitution instruction, two semantic side conditions (conditions (1) and (2)), and a syntactic side condition (condition (3)). Moreover, if either of the two semantic side conditions is false, $(\lambda\, x : \alpha \,.\, b)(a)$ is undefined.

Using constructions, quotation, and evaluation, both laws of beta reduction can be formalized in Chiron as rules of inference in which the substitution instructions and syntactic side conditions are expressed in Chiron. For example, the law of beta reduction for function abstractions would be:

From

1. $\mathsf{is\text{-}fun\text{-}redex}(a)$,
2. $\mathsf{sub}(\mathsf{redex\text{-}arg}(a), \mathsf{1st\text{-}comp}(\mathsf{redex\text{-}var}(a)), \mathsf{redex\text{-}body}(a))\!\downarrow$,
3. $\mathsf{free\text{-}for}(\mathsf{redex\text{-}arg}(a), \mathsf{1st\text{-}comp}(\mathsf{redex\text{-}var}(a)), \mathsf{redex\text{-}body}(a))$

infer

$$\begin{aligned}&\mathsf{if}([\![\mathsf{redex\text{-}arg}(a)]\!]_{\mathrm{te}}\downarrow([\![\mathsf{2nd\text{-}comp}(\mathsf{redex\text{-}var}(a))]\!]_{\mathrm{ty}}\cap\mathsf{V})\wedge\\&\quad[\![\mathsf{sub}(\mathsf{redex\text{-}arg}(a),\mathsf{1st\text{-}comp}(\mathsf{redex\text{-}var}(a)),\mathsf{redex\text{-}body}(a))]\!]_{\mathrm{te}}\downarrow\mathsf{V},\\&[\![a]\!]_{\mathrm{te}}\simeq[\![\mathsf{sub}(\mathsf{redex\text{-}arg}(a),\mathsf{1st\text{-}comp}(\mathsf{redex\text{-}var}(a)),\mathsf{redex\text{-}body}(a))]\!]_{\mathrm{te}},\\&[\![a]\!]_{\mathrm{te}}\uparrow)\end{aligned}$$

The beta reduction law for function abstractions is applied to an application b of a function abstraction by letting a be the quotation $\ulcorner b\urcorner$.

7.7 Liar Paradox

We will formalize in Chiron the liar paradox mentioned in subsection 4.1. Assume that nat is a type and $0, 1, 2, \ldots$ denote terms such that nat denotes an infinite set $\{0, 1, 2, \ldots\}$. Assume also that num is a type that denotes the set $\{\ulcorner 0\urcorner, \ulcorner 1\urcorner, \ulcorner 2\urcorner, \ldots\}$. And finally assume that enum is a term that denotes a function which is a bijection from $\{\ulcorner 0\urcorner, \ulcorner 1\urcorner, \ulcorner 2\urcorner, \ldots\}$ to the set F of all terms of type $(\Lambda\, e : \mathsf{E}\ .\ \mathsf{E})$—which is the same as $(\mathsf{E} \to \mathsf{E})$—that are definable by a syntactically closed function abstraction of the form $(\lambda\, e : \mathsf{E}\ .\ b)$.

The following function abstraction denotes some $f \in F$:

$$\begin{aligned}&\lambda\, e : \mathsf{E}\ .\\&\quad[\ulcorner\mathsf{op\text{-}app}\urcorner,\\&\quad\ [\ulcorner\mathsf{op}\urcorner,\ulcorner\mathsf{not}\urcorner,\ulcorner\mathsf{formula}\urcorner,\ulcorner\mathsf{formula}\urcorner],\\&\quad\ [\ulcorner\mathsf{eval}\urcorner,\\&\quad\quad[\ulcorner\mathsf{fun\text{-}app}\urcorner,[\ulcorner\mathsf{fun\text{-}app}\urcorner,\ulcorner\mathsf{enum}\urcorner,e],e],\\&\quad\quad\ulcorner\mathsf{formula}\urcorner]].\end{aligned}$$

Using quasiquotation, f can be expressed much more succinctly as

$$\lambda\, e : \mathsf{E}\ .\ \ulcorner\neg[\![\mathsf{enum}(\lfloor e\rfloor)(\lfloor e\rfloor)]\!]_{\mathrm{fo}}\urcorner.$$

For some i of type nat, $\mathsf{enum}(\ulcorner i\urcorner) = f$. Then

$$\begin{aligned}\mathsf{enum}(\ulcorner i\urcorner)(\ulcorner i\urcorner) &= f(\ulcorner i\urcorner)\\&= \ulcorner\neg[\![\mathsf{enum}(\ulcorner i\urcorner)(\ulcorner i\urcorner)]\!]_{\mathrm{fo}}\urcorner.\end{aligned}$$

Therefore, if LIAR is the term $\mathsf{enum}(\ulcorner i \urcorner)(\ulcorner i \urcorner)$, then

$$\mathsf{LIAR} = \ulcorner \neg [\![\mathsf{LIAR}]\!]_{\mathrm{fo}} \urcorner.$$

8 A Proof System

We present in this section a proof system for Chiron called $\mathbf{C}_L$ (parameterized by a normal language L). We will state and prove a soundness theorem and a completeness theorem for $\mathbf{C}_L$. $\mathbf{C}_L$ is intended to be neither a practical nor implemented proof system for Chiron. Its purpose is to serve as (1) a system test of Chiron's definition and (2) a reference system for other, possibly implemented, proof systems for Chiron.

8.1 Definitions

Fix a normal language L of Chiron for the rest of this section. $\mathbf{C}_L$ is a Hilbert-style proof system for Chiron consisting of 10 rules of inference and an infinite set of axioms. The *rules of inference* are given below in subsection 8.2. The *axioms* are the instances of the 68 axiom schemas given below in subsection 8.3. The set of axioms depends on the language L.

Let $T = (L, \Gamma)$ be a theory of Chiron and A be a formula. A *proof* of A from T in $\mathbf{C}_L$ is a finite sequence of formulas of L such that A is the last member of the sequence and every member of the sequence is an axiom of $\mathbf{C}_L$, a member of Γ, or is inferred from previous members of the sequence by one of the rules of inference of $\mathbf{C}_L$. Let $T \vdash A$ mean there is a proof of A from T in $\mathbf{C}_L$. T is *consistent* if there is some formula A of L such that $T \vdash A$ does not hold.

Let $\mathcal{T}$ be a set of theories over L and $\mathcal{F}$ be a set of formulas of L. $\mathbf{C}_L$ is *sound with respect to $\mathcal{T}$ and $\mathcal{F}$* if, for every $T \in \mathcal{T}$ and formula $A \in \mathcal{F}$,

$$T \vdash A \text{ implies } T \models A.$$

$\mathbf{C}_L$ is *complete with respect to $\mathcal{T}$ and $\mathcal{F}$* if, for every $T \in \mathcal{T}$ and formula $A \in \mathcal{F}$,

$$T \models A \text{ implies } T \vdash A.$$

After presenting the rules of inference and axioms of $\mathbf{C}_L$, we will prove that $\mathbf{C}_L$ is sound with respect to the set of all normal theories over L and the set of all formulas of L and complete with respect to the set of all eval-free normal theories over L and the set of all eval-free formulas of L.

8.2 Rules of Inference

$\mathbf{C}_L$ has the 10 rules of inference stated below. The first two rules of inference are often employed in Hilbert-style proof systems for first-order logic. The last eight would normally be expressed as axiom schemas in a Hilbert-style proof systems for a traditional logic. They are expressed as rules of inference in $\mathbf{C}_L$ because the syntactic side conditions, which are expressed directly in Chiron, must be hypotheses in order for the rules to a preserve validity in a standard model of a normal theory.

Rule 1 (Modus Ponens)
From

1. A,
2. $A \supset B$

infer

B.

Rule 2 (Universal Generalization)
From

A

infer

$\forall\, x : \alpha \,.\, A$.

Rule 3 (Universal Quantifier Shifting)
From

$\neg\mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner A \urcorner)$

infer

$(\forall\, x : \alpha \,.\, (A \vee B)) \supset (A \vee (\forall\, x : \alpha \,.\, B))$.

Rule 4 (Universal Instantiation)
From

1. $\mathsf{sub}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner A \urcorner)\downarrow$,
2. $\mathsf{free\text{-}for}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner A \urcorner)$

infer

$$((\forall\, x : \alpha\ .\ A) \wedge a \downarrow \alpha) \supset [\![\mathsf{sub}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner A \urcorner)]\!]_{\mathrm{fo}}.$$

Rule 5 (Definite Description)
From

1. $\mathsf{sub}(\ulcorner (\iota\, x : \alpha\ .\ A) \urcorner, \ulcorner x \urcorner, \ulcorner A \urcorner)\downarrow$,
2. $\mathsf{free\text{-}for}(\ulcorner (\iota\, x : \alpha\ .\ A) \urcorner, \ulcorner x \urcorner, \ulcorner A \urcorner)$

infer

$$(\exists !\, x : \alpha\ .\ A) \supset [\![\mathsf{sub}(\ulcorner (\iota\, x : \alpha\ .\ A) \urcorner, \ulcorner x \urcorner, \ulcorner A \urcorner)]\!]_{\mathrm{fo}}.$$

Rule 6 (Indefinite Description)
From

1. $\mathsf{sub}(\ulcorner (\epsilon\, x : \alpha\ .\ A) \urcorner, \ulcorner x \urcorner, \ulcorner A \urcorner)\downarrow$,
2. $\mathsf{free\text{-}for}(\ulcorner (\epsilon\, x : \alpha\ .\ A) \urcorner, \ulcorner x \urcorner, \ulcorner A \urcorner)$

infer

$$(\exists\, x : \alpha\ .\ A) \supset [\![\mathsf{sub}(\ulcorner (\epsilon\, x : \alpha\ .\ A) \urcorner, \ulcorner x \urcorner, \ulcorner A \urcorner)]\!]_{\mathrm{fo}}.$$

The function machinery of Chiron—type application, dependent function types, function application, and function abstraction—is specified by the following four rules of inference.

Rule 7 (Type Application)
From

1. $\neg\mathsf{free\text{-}in}(\ulcorner y \urcorner, \ulcorner \alpha \urcorner)$,
2. $\neg\mathsf{free\text{-}in}(\ulcorner y \urcorner, \ulcorner a \urcorner)$,
3. $\neg\mathsf{free\text{-}in}(\ulcorner f \urcorner, \ulcorner \alpha \urcorner)$,
4. $\neg\mathsf{free\text{-}in}(\ulcorner f \urcorner, \ulcorner a \urcorner)$

infer

$$\mathsf{if}(a\downarrow,\ \forall\, y : \mathsf{C}\ .\ y \downarrow \alpha(a) \equiv (\exists\, f : \alpha\ .\ \mathsf{fun}(f) \wedge \langle a, y \rangle \in f),\ \alpha(a) =_{\mathrm{ty}} \mathsf{C}).$$

Rule 8 (Dependent Function Types)
From

1. $\neg\mathsf{free\text{-}in}(\ulcorner f \urcorner, \ulcorner \alpha \urcorner)$,
2. $\neg\mathsf{free\text{-}in}(\ulcorner f \urcorner, \ulcorner \beta \urcorner)$,
3. $\neg\mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner \alpha \urcorner)$,
4. $\neg\mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner \beta \urcorner)$

infer

$$\forall\, f : \mathsf{C}\ .\ f \downarrow (\Lambda\, x : \alpha\ .\ \beta) \equiv \\ (\mathsf{fun}(f) \wedge (\forall\, x : V\ .\ f(x)\downarrow \supset (x \downarrow \alpha \wedge f(x) \downarrow \beta))).$$

Rule 9 (Function Application)
From

1. $\neg\mathsf{free\text{-}in}(\ulcorner y \urcorner, \ulcorner \alpha \urcorner)$,
2. $\neg\mathsf{free\text{-}in}(\ulcorner y \urcorner, \ulcorner a \urcorner)$,
3. $\neg\mathsf{free\text{-}in}(\ulcorner y \urcorner, \ulcorner f \urcorner)$

infer

$$f(a) \simeq (\iota\, y : \alpha(a)\ .\ \mathsf{fun}(f) \wedge \langle a, y \rangle \in f).$$

where f is of type α.

Rule 10 (Function Abstraction)
From

1. $\neg\mathsf{free\text{-}in}(\ulcorner f \urcorner, \ulcorner \alpha \urcorner)$,
2. $\neg\mathsf{free\text{-}in}(\ulcorner f \urcorner, \ulcorner \beta \urcorner)$,
3. $\neg\mathsf{free\text{-}in}(\ulcorner f \urcorner, \ulcorner b \urcorner)$,
4. $\neg\mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner \alpha \urcorner)$,
5. $\neg\mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner b \urcorner)$,
6. $\neg\mathsf{free\text{-}in}(\ulcorner y \urcorner, \ulcorner \beta \urcorner)$,
7. $\neg\mathsf{free\text{-}in}(\ulcorner y \urcorner, \ulcorner b \urcorner)$

infer

$$(\lambda\, x : \alpha\ .\ b) \simeq \\ \quad (\iota\, f : (\Lambda\, x : \alpha\ .\ \beta)\ . \\ \quad\quad (\forall\, x : \alpha,\, y : \beta\ .\ f(x) = y \equiv (x \downarrow \mathsf{V} \wedge b \downarrow \mathsf{V} \wedge y = b))).$$

where b is of type β.

8.3 Axiom Schemas

The axioms of $\mathbf{C}_L$ are presented by 68 axiom schemas, organized into 13 groups. Each axiom schema is specified by a formula schema. An instance of an axiom schema of $\mathbf{C}_L$ is any formula of L that is obtained by replacing the schema variables in the schema with appropriate expressions.

The first group of axiom schemas define conjunction and implication in terms of negation and disjunction in the usual way and give a complete set of axioms for propositional logic in terms of disjunction and implication.

Axiom Schemas 1 (Propositional Logic)

1. $(A \wedge B) \equiv \neg(\neg A \vee \neg B)$.
2. $(A \supset B) \equiv (\neg A \vee B)$.
3. $(A \vee A) \supset A$.
4. $A \supset (B \vee A)$.
5. $(A \supset B) \supset ((C \vee A) \supset (B \vee C)))$.

The next group of axiom schemas specify that the three equalities type-equal, quasi-equal, and formula-equal are equivalence relations as well as congruences with respect to formulas.

Axiom Schemas 2 (Equality)

1. $\alpha =_{\mathrm{ty}} \alpha$.
2. $\alpha =_{\mathrm{ty}} \beta \subseteq C \equiv D$

   where D is the result of replacing one occurrence of α in C by an occurrence of β, provided that the occurrence of α in C is not within a quotation.
3. $a \simeq a$.

4. $a \simeq b \subseteq C \equiv D$

   where D is the result of replacing one occurrence of a in C by an occurrence of b, provided that the occurrence of a in C is not within a quotation and is not a variable $(x : \alpha)$ immediately preceded by Λ, λ, $\exists$, ι, or ϵ.

5. $A \equiv A$.

6. $A \equiv B \subseteq C \equiv D$

   where D is the result of replacing one occurrence of A in C by an occurrence of B, provided that the occurrence of A in C is not within a quotation.

The following axiom schemas specify the general definedness laws for operators. For a kind k, let

$$\overline{k} = \begin{cases} k & \text{if } k = \mathsf{type} \\ \mathsf{C} & \text{if } \mathbf{type}_L[k] \\ k & \text{if } k = \mathsf{formula}. \end{cases}$$

Axiom Schemas 3 (General Operator Properties)

1. $e_{i_1} \downarrow k_{i_1} \wedge \cdots \wedge e_{i_m} \downarrow k_{i_m} \supset$
   $(o :: k_1, \ldots, k_n, \mathsf{type})(e_1, \ldots, e_n) =_{\mathrm{ty}}$
   $(o :: \overline{k_1}, \ldots, \overline{k_n}, \mathsf{type})(e_1, \ldots, e_n)$

   where $n \geq 1$ and $k_{i_1}, \ldots, k_{i_m}$ is the subsequence of types in the sequence $k_1, \ldots, k_n$ of kinds.

2. $e_{i_1} \downarrow k_{i_1} \wedge \cdots \wedge e_{i_m} \downarrow k_{i_m} \supset$
   $(o :: k_1, \ldots, k_n, \beta)(e_1, \ldots, e_n) \simeq$
   $(o :: \overline{k_1}, \ldots, \overline{k_n}, \beta)(e_1, \ldots, e_n)$

   where $n \geq 1$ and $k_{i_1}, \ldots, k_{i_m}$ is the subsequence of types in the sequence $k_1, \ldots, k_n$ of kinds.

3. $e_{i_1} \downarrow k_{i_1} \wedge \cdots \wedge e_{i_m} \downarrow k_{i_m} \supset$
   $(o :: k_1, \ldots, k_n, \mathsf{formula})(e_1, \ldots, e_n) \equiv$
   $(o :: \overline{k_1}, \ldots, \overline{k_n}, \mathsf{formula})(e_1, \ldots, e_n)$

   where $n \geq 1$ and $k_{i_1}, \ldots, k_{i_m}$ is the subsequence of types in the sequence $k_1, \ldots, k_n$ of kinds.

4. $(o :: k_1, \ldots, k_n, \beta)(e_1, \ldots, e_n)\downarrow \supset$
$(o :: k_1, \ldots, k_n, \beta)(e_1, \ldots, e_n) =$
$(o :: k_1, \ldots, k_n, \mathsf{C})(e_1, \ldots, e_n)$

   where $n \geq 0$.

5. $(a\downarrow \wedge a \uparrow \alpha) \supset$
$(o :: k_1, \ldots, k_{i-1}, \alpha, k_{i+1}, \ldots, k_n, \mathsf{type})$
$(e_1, \ldots, e_{i-1}, a, e_{i+1}, \ldots, e_n) =_{\mathrm{ty}} \mathsf{C}$

   where $n \geq 1$.

6. $(a\downarrow \wedge a \uparrow \alpha) \supset$
$(o :: k_1, \ldots, k_{i-1}, \alpha, k_{i+1}, \ldots, k_n, \beta)$
$(e_1, \ldots, e_{i-1}, a, e_{i+1}, \ldots, e_n)\uparrow$

   where $n \geq 1$.

7. $(a\downarrow \wedge a \uparrow \alpha) \supset$
$(o :: k_1, \ldots, k_{i-1}, \alpha, k_{i+1}, \ldots, k_n, \mathsf{formula})$
$(e_1, \ldots, e_{i-1}, a, e_{i+1}, \ldots, e_n) \equiv \mathsf{F}$

   where $n \geq 1$.

8. $(o :: k_1, \ldots, k_n, \beta)(e_1, \ldots, e_n)\downarrow \supset$
$(o :: k_1, \ldots, k_n, \beta)(e_1, \ldots, e_n) \downarrow \beta$

   where $n \geq 0$.

The following axiom schemas specify specific definedness laws for the built-in operators.

Axiom Schemas 4 (Built-In Operator Definedness)

1. $\ell\downarrow$.
2. $\mathsf{E}_{\mathrm{on},a} \neq_{\mathrm{ty}} \mathsf{C} \supset a\downarrow$.
3. $\mathsf{E}_a \neq_{\mathrm{ty}} \mathsf{C} \supset a\downarrow$.
4. $\mathsf{E}_{\mathrm{op},a} \neq_{\mathrm{ty}} \mathsf{C} \supset a\downarrow$.
5. $\mathsf{E}_{\mathrm{ty},a} \neq_{\mathrm{ty}} \mathsf{C} \supset a\downarrow$.
6. $\mathsf{E}_{\mathrm{te},a} \neq_{\mathrm{ty}} \mathsf{C} \supset a\downarrow$.
7. $\mathsf{E}^b_{\mathrm{te},a} \neq_{\mathrm{ty}} \mathsf{C} \supset (a\downarrow \wedge b\downarrow)$.

8. $\mathsf{E}_{\mathrm{fo},a} \neq_{\mathrm{ty}} \mathsf{C} \supset a{\downarrow}$.

9. $a \in b \supset (a{\downarrow} \wedge b{\downarrow})$.

10. $a =_\alpha b \supset (a \downarrow \alpha \wedge b \downarrow \alpha)$.

The general definedness law for variables is specified by the following axiom schema.

Axiom Schemas 5 (Variable Definedness)

1. $(x : \alpha){\downarrow} \supset (x : \alpha) \downarrow \alpha$.

The following axiom schemas specify the extensionality of types and universal quantification over the canonical empty type.

Axiom Schemas 6 (Types)

1. $\alpha =_{\mathrm{ty}} \beta \equiv (\forall x : \mathsf{C} \,.\, x \downarrow \alpha \equiv x \downarrow \beta)$.

2. $\forall x : \nabla \,.\, A$.

The following 18 axiom schemas correspond to the 18 axiom schemas that constitute the axiomatization of NBG set theory given by K. Gödel in [13].

Axiom Schemas 7 (NBG Set Theory)

1. $(x : \mathsf{C}){\downarrow}$.

2. $\forall x : \mathsf{C} \,.\, x \downarrow \mathsf{V} \equiv (\exists y : \mathsf{C} \,.\, x \in y)$.

3. $\forall x, y : \mathsf{C} \,.\, (\forall u : \mathsf{V} \,.\, u \in x \equiv u \in y) \supset x = y$.

4. $\forall u, v : \mathsf{V} \,.\, \{u, v\}{\downarrow}$.

5. $\exists x : \mathsf{C} \,.\, \forall u, v : \mathsf{V} \,.\, \langle u, v\rangle \in x \equiv u \in v$.

6. $\forall x, y : \mathsf{C} \,.\, (x \cap y){\downarrow}$.

7. $\forall x : \mathsf{C} \,.\, \overline{x}{\downarrow}$.

8. $\forall x : \mathsf{C} \,.\, \mathsf{dom}(x){\downarrow}$.

9. $\forall x : \mathsf{C} \,.\, \exists y : \mathsf{C} \,.\, \forall u, v : \mathsf{V} \,.\, \langle u, v\rangle \in y \equiv u \in x$.

10. $\forall\, x : \mathsf{C} \,.\, \exists\, y : \mathsf{C} \,.\, \forall\, u, v : \mathsf{V} \,.\, \langle u, v\rangle \in y \equiv \langle v, u\rangle \in x.$

11. $\forall\, x : \mathsf{C} \,.\, \exists\, y : \mathsf{C} \,.\, \forall\, u, v, w : \mathsf{V} \,.\, \langle u, v, w\rangle \in y \equiv \langle v, w, u\rangle \in x.$

12. $\forall\, x : \mathsf{C} \,.\, \exists\, y : \mathsf{C} \,.\, \forall\, u, v, w : \mathsf{V} \,.\, \langle u, v, w\rangle \in y \equiv \langle u, w, v\rangle \in x.$

13. $\exists\, u : \mathsf{V} \,.\, u \neq \emptyset \wedge (\forall\, v : \mathsf{V} \,.\, v \in u \supset (\exists\, w : \mathsf{V} \,.\, w \in u \wedge v \subset w)).$

14. $\forall\, u : \mathsf{V} \,.\, \mathsf{sum}(u) \downarrow \mathsf{V}.$

15. $\forall\, u : \mathsf{V} \,.\, \mathsf{power}(u) \downarrow \mathsf{V}.$

16. $\forall\, x : \mathsf{C} \,.\, \mathsf{univocal}(x) \supset$
$(\forall\, u : \mathsf{V} \,.\, \exists\, v : \mathsf{V} \,.\, \forall\, t : \mathsf{V} \,.\, t \in v \equiv (\exists\, s : \mathsf{V} \,.\, s \in u \wedge \langle s, t\rangle \in x)).$

17. $\forall\, x : \mathsf{C} \,.\, x \neq \emptyset \supset (\exists\, u : \mathsf{V} \,.\, u \in x \wedge \forall\, v : \mathsf{V} \,.\, \neg(v \in u \wedge v \in x)).$

18. $\exists\, f : \mathsf{C} \,.\, \mathsf{fun}(f) \wedge (\forall\, u : \mathsf{V} \,.\, u \neq \emptyset \supset f(u) \in u).$

Conditional terms are specified by two axiom schemas.

Axiom Schemas 8 (Conditional Terms)

1. $A \supset \mathsf{if}(A, b, c) \simeq b.$

2. $\neg A \supset \mathsf{if}(A, b, c) \simeq c.$

Definite description is specified by the Definite Description rule of inference given above and the following two axiom schemas, one for proper definite descriptions and one for improper.

Axiom Schemas 9 (Definite Description)

1. $(\exists\,!\, x : \alpha \,.\, A) \supset (\iota\, x : \alpha \,.\, A) \downarrow \alpha).$

2. $\neg(\exists\,!\, x : \alpha \,.\, A) \supset (\iota\, x : \alpha \,.\, A)\uparrow .$

Indefinite description is specified by the Indefinite Description rule of inference given above and the following three axiom schemas, one for proper indefinite descriptions, one for improper indefinite descriptions, and one for formulas that are satisfied by the same classes.

Axiom Schemas 10 (Indefinite Description)

1. $(\exists\, x : \alpha \;.\; A) \supset (\epsilon\, x : \alpha \;.\; A) \downarrow \alpha)$.
2. $\neg(\exists\, x : \alpha \;.\; A) \supset (\epsilon\, x : \alpha \;.\; A)\uparrow$.
3. $(\forall\, x : \mathsf{C} \;.\; A \equiv B) \supset (\epsilon\, x : \mathsf{C} \;.\; A) = (\epsilon\, x : \mathsf{C} \;.\; B)$.

Quotation is specified by the following three axiom schemas. Note that the quotation of a symbol and the quotation of an operator name are not fully specified.

Axiom Schemas 11 (Quotation)

1. $\ulcorner s \urcorner \downarrow \mathsf{E}_{\mathrm{sy}}$

   where $s \in \mathcal{S}$.
2. $\ulcorner o \urcorner \downarrow \mathsf{E}_{\mathrm{on}}$

   where $o \in \mathcal{O}$.
3. $\ulcorner s_1 \urcorner \neq \ulcorner s_2 \urcorner$

   where $s_1, s_2 \in \mathcal{S} \cup \mathcal{O}$ *with* $s_1 \neq s_2$.
4. $\ulcorner (e_1, \ldots, e_n) \urcorner =_{\mathsf{E}} [\ulcorner e_1 \urcorner, \ldots, \ulcorner e_n \urcorner]$

   where $e_1, \ldots, e_n \in \mathcal{E}_L$ *and* $n \geq 0$.

Evaluation is specified by the following six axiom schemas. This set of axiom schemas expresses the same information as parts 1–6 of Lemma 6.3.1.

Axiom Schemas 12 (Evaluation)

1. $\mathsf{is\text{-}eval\text{-}free}(\ulcorner \alpha \urcorner) \supset [\![\ulcorner \alpha \urcorner]\!]_{\mathrm{ty}} =_{\mathrm{ty}} \alpha$.
2. $\neg\mathsf{gea}(b, \ulcorner \mathsf{type} \urcorner) \supset [\![b]\!]_{\mathrm{ty}} =_{\mathrm{ty}} \mathsf{C}$.
3. $\mathsf{is\text{-}eval\text{-}free}(\ulcorner a \urcorner) \supset [\![\ulcorner a \urcorner]\!]_{\alpha} = \mathsf{if}(a \downarrow \alpha, a, \bot_{\mathsf{C}})$.
4. $\neg\mathsf{gea}(b, \ulcorner \alpha \urcorner) \supset [\![b]\!]_{\alpha} \simeq \bot_{\mathsf{C}}$.
5. $\mathsf{is\text{-}eval\text{-}free}(\ulcorner A \urcorner) \supset [\![\ulcorner A \urcorner]\!]_{\mathrm{fo}} \equiv A$.
6. $\neg\mathsf{gea}(b, \ulcorner \mathsf{formula} \urcorner) \supset [\![b]\!]_{\mathrm{fo}} \equiv \mathsf{F}$.

The following eight axiom schemas specify the eight kinds of construction types. The type operators named expr-sym and expr-op-name are specified by their properties. expr is specified from expr-sym and expr-op-name by induction. And those named expr-op, expr-type, expr-term-type, expr-term, and expr-formula are defined by mutual recursion.

Axiom Schemas 13 (Construction Types)

1. $\ell = \mathsf{term}(\mathsf{E}_{\mathrm{on}})$.

2. $\mathsf{E}_{\mathrm{sy}} \cup \mathsf{E}_{\mathrm{on}} \ll \mathsf{V}$.

3. $\mathsf{E}_{\mathrm{sy}} \cap \mathsf{E}_{\mathrm{on}} =_{\mathrm{ty}} \nabla$.

4. $\mathsf{countably\text{-}infinite}(\mathsf{term}(\mathsf{E}_{\mathrm{sy}}))$.

5. $\exists\, u : \mathsf{V} \,.\, \mathsf{countably\text{-}infinite}(u) \wedge \mathsf{term}(\mathsf{E}_{\mathrm{on}}) \subseteq u$.

6. $\forall\, x : \mathsf{E}_{\mathrm{sy}} \cup \mathsf{E}_{\mathrm{on}} \,.\, x \neq \emptyset \wedge \neg\, \mathsf{is\text{-}ord\text{-}pair}(x)$.

7. $\forall\, u : \mathsf{L}, u = \mathsf{term}(\mathsf{E}_{\mathrm{on},u})$.

8. $\forall\, u : \mathsf{L}, v : \mathsf{V} \,.$
   $((\forall\, u_0 : \mathsf{E}_{\mathrm{sy}} \cup \mathsf{E}_{\mathrm{on},u} \,.\, u_0 \in v) \wedge$
   $\emptyset \in v \wedge$
   $(\forall\, u_1, u_2 : \mathsf{V} \,.\, (u_1 \in v \wedge u_2 \in v \wedge u_2 \uparrow \mathsf{E}_{\mathrm{sy}} \cup \mathsf{E}_{\mathrm{on},u}) \supset \langle u_1, u_2 \rangle \in v))$
   $\supset \mathsf{term}(\mathsf{E}_u) \subseteq v$.

9. $\forall\, u : \mathsf{L}, x : \mathsf{C} \,.\, x \downarrow \mathsf{E}_{\mathrm{op},u} \equiv$
   $(\exists\, e : \mathsf{E}_{\mathrm{on},u} \,.\, x = \ulcorner(\lfloor e \rfloor :: \mathsf{type})\urcorner \vee$
   $\quad (\exists\, e' : \mathsf{E}_{\mathrm{ty},u} \,.\, x = \ulcorner(\lfloor e \rfloor :: \lfloor e' \rfloor)\urcorner) \vee$
   $\quad x = \ulcorner(\lfloor e \rfloor :: \mathsf{formula})\urcorner) \vee$
   $(\exists\, e : \mathsf{E}_{\mathrm{op},u} \,.\, x = e\hat{}[\ulcorner\mathsf{type}\urcorner] \vee$
   $\quad (\exists\, e' : \mathsf{E}_{\mathrm{ty},u} \,.\, x = e\hat{}[e'])$
   $\quad x = e\hat{}[\ulcorner\mathsf{formula}\urcorner])$.

10. $\forall\, u : \mathsf{L}, x : \mathsf{C} \,.\, x \downarrow \mathsf{E}_{\mathrm{ty},u} \equiv$
    $(\exists\, e : \mathsf{E}_{\mathrm{on},u} \,.\, x = [\ulcorner\mathsf{op\text{-}app}\urcorner, [\ulcorner\mathsf{op}\urcorner, e, \ulcorner\mathsf{type}\urcorner]]) \vee$
    $(\exists\, e_1 : \mathsf{E}_{\mathrm{on},u}, e_2, e_3 : \mathsf{E}_u \,.$
    $\quad [\ulcorner\mathsf{op\text{-}app}\urcorner, [\ulcorner\mathsf{op}\urcorner, e_1]\hat{}e_2\hat{}[\ulcorner\mathsf{type}\urcorner]]\hat{}e_3 \in \mathsf{E}_{\mathrm{ty},u} \wedge$
    $\quad ((\exists\, e : \mathsf{E}_{\mathrm{ty},u} \,.$
    $\quad\quad x = [\ulcorner\mathsf{op\text{-}app}\urcorner, [\ulcorner\mathsf{op}\urcorner, e_1]\hat{}e_2\hat{}[\ulcorner\mathsf{type}\urcorner, \ulcorner\mathsf{type}\urcorner]]\hat{}e_3\hat{}[e]) \vee$
    $\quad\quad (\exists\, e : \mathsf{E}_{\mathrm{ty},u}, e' : \mathsf{E}_{\mathrm{te},u} \,.$

$$
\begin{array}{l}
\qquad x = [\ulcorner\mathsf{op\text{-}app}\urcorner, [\ulcorner\mathsf{op}\urcorner, e_1]\hat{}e_2\hat{}[e, \ulcorner\mathsf{type}\urcorner]]\hat{}e_3\hat{}[e']) \\
\quad (\exists\, e : \mathsf{E}_{\mathrm{fo},u}\ .\ \\
\qquad x = [\ulcorner\mathsf{op\text{-}app}\urcorner, [\ulcorner\mathsf{op}\urcorner, e_1]\hat{}e_2\hat{}[\ulcorner\mathsf{formula}\urcorner, \ulcorner\mathsf{type}\urcorner]]\hat{}e_3\hat{}[e]) \\
\quad)) \vee \\
(\exists\, e_1 : \mathsf{E}_{\mathrm{ty},u}, e_2 : \mathsf{E}_{\mathrm{te},u}\ .\ x = \ulcorner\lfloor e_1\rfloor(\lfloor e_2\rfloor)\urcorner) \vee \\
(\exists\, e_1 : \mathsf{E}_{\mathrm{on},u}, e_2, e_3 : \mathsf{E}_{\mathrm{ty},u}\ .\ x = \ulcorner(\Lambda\, \lfloor e_1\rfloor : \lfloor e_2\rfloor\ .\ \lfloor e_3\rfloor)\urcorner) \vee \\
(\exists\, e : \mathsf{E}_{\mathrm{te},u}\ .\ x = \ulcorner[\![\lfloor e\rfloor]\!]_{\mathrm{ty}}\urcorner).
\end{array}
$$

11.
$$
\begin{array}{l}
\forall\, u : \mathsf{L}, \hat{e} : \mathsf{E}_{\mathrm{ty},u}, x : \mathsf{C}\ .\ x \downarrow \mathsf{E}^{\hat{e}}_{\mathrm{te},u} \equiv \\
\quad (\exists\, e : \mathsf{E}_{\mathrm{on},u}\ .\ x = [\ulcorner\mathsf{op\text{-}app}\urcorner, [\ulcorner\mathsf{op}\urcorner, e, \hat{e}]]) \vee \\
\quad (\exists\, e_1 : \mathsf{E}_{\mathrm{on},u}, e_2, e_3 : \mathsf{E}_{u}\ . \\
\qquad [\ulcorner\mathsf{op\text{-}app}\urcorner, [\ulcorner\mathsf{op}\urcorner, e_1]\hat{}e_2\hat{}[\hat{e}]]\hat{}e_3 \in \mathsf{E}^{\hat{e}}_{\mathrm{te},u} \wedge \\
\qquad ((\exists\, e : \mathsf{E}_{\mathrm{ty},u}\ . \\
\qquad\quad x = [\ulcorner\mathsf{op\text{-}app}\urcorner, [\ulcorner\mathsf{op}\urcorner, e_1]\hat{}e_2\hat{}[\ulcorner\mathsf{type}\urcorner, \hat{e}]]\hat{}e_3\hat{}[e]) \vee \\
\qquad\ (\exists\, e : \mathsf{E}_{\mathrm{ty},u}, e' : \mathsf{E}_{\mathrm{te},u}\ . \\
\qquad\quad x = [\ulcorner\mathsf{op\text{-}app}\urcorner, [\ulcorner\mathsf{op}\urcorner, e_1]\hat{}e_2\hat{}[e, \hat{e}]]\hat{}e_3\hat{}[e']) \\
\qquad\ (\exists\, e : \mathsf{E}_{\mathrm{fo},u}\ . \\
\qquad\quad x = [\ulcorner\mathsf{op\text{-}app}\urcorner, [\ulcorner\mathsf{op}\urcorner, e_1]\hat{}e_2\hat{}[\ulcorner\mathsf{formula}\urcorner, \hat{e}]]\hat{}e_3\hat{}[e]))) \vee \\
\quad (\exists\, e : \mathsf{E}_{\mathrm{sy},u}\ .\ x = \ulcorner(\lfloor e\rfloor : \lfloor \hat{e}\rfloor)\urcorner) \vee \\
\quad (\exists\, e_1 : \mathsf{E}_{\mathrm{ty},u}, e_2 : \mathsf{E}^{e_1}_{\mathrm{te},u}, e_3 : \mathsf{E}_{\mathrm{te},u}\ . \\
\qquad x = \ulcorner\lfloor e_2\rfloor(\lfloor e_3\rfloor)\urcorner \wedge \hat{e} = \ulcorner\lfloor e_1\rfloor(\lfloor e_3\rfloor)\urcorner) \vee \\
\quad (\exists\, e_1 : \mathsf{E}_{\mathrm{sy},u}, e_2, e_3 : \mathsf{E}_{\mathrm{ty},u}, e_4 : \mathsf{E}^{e_3}_{\mathrm{te},u}\ . \\
\qquad x = \ulcorner\lambda\, \lfloor e_1\rfloor : \lfloor e_2\rfloor\ .\ \lfloor e_4\rfloor\urcorner \wedge \hat{e} = \ulcorner\Lambda\, \lfloor e_1\rfloor : \lfloor e_2\rfloor\ .\ \lfloor e_3\rfloor\urcorner) \vee \\
\quad (\exists\, e_1 : \mathsf{E}_{\mathrm{fo},u}, e_2, e_3 : \mathsf{E}_{\mathrm{ty},u}, e_4 : \mathsf{E}^{e_2}_{\mathrm{te},u}, e_5 : \mathsf{E}^{e_3}_{\mathrm{te},u}\ . \\
\qquad x = \ulcorner\mathsf{if}, \lfloor e_1\rfloor, \lfloor e_4\rfloor, \lfloor e_5\rfloor\urcorner \wedge \hat{e} = \mathsf{if}(e_2 = e_3, e_2, \ulcorner\mathsf{C}\urcorner)) \vee \\
\quad (\exists\, e_1 : \mathsf{E}_{\mathrm{sy},u}, e_2 : \mathsf{E}_{\mathrm{fo},u}\ .\ x = \ulcorner(\iota\, \lfloor e_1\rfloor : \lfloor \hat{e}\rfloor\ .\ \lfloor e_2\rfloor)\urcorner) \vee \\
\quad (\exists\, e_1 : \mathsf{E}_{\mathrm{sy},u}, e_2 : \mathsf{E}_{\mathrm{fo},u}\ .\ x = \ulcorner(\epsilon\, \lfloor e_1\rfloor : \lfloor \hat{e}\rfloor\ .\ \lfloor e_2\rfloor)\urcorner) \vee \\
\quad (\exists\, e : \mathsf{E}_{u}\ .\ x = \ulcorner\lfloor e\rfloor\urcorner \wedge \hat{e} = \ulcorner\mathsf{E}_u\urcorner) \vee \\
\quad (\exists\, e : \mathsf{E}_{\mathrm{te},u}\ .\ x = \ulcorner[\![\lfloor e\rfloor]\!]_{\lfloor \hat{e}\rfloor}\urcorner).
\end{array}
$$

12. $\forall\, u : \mathsf{L}, x : \mathsf{C}\ .\ x \downarrow \mathsf{E}_{\mathrm{te},u} \equiv (\exists\, \hat{e} : \mathsf{E}_{\mathrm{ty},u}\ .\ x \downarrow \mathsf{E}^{\hat{e}}_{\mathrm{te},u})$.

13.
$$
\begin{array}{l}
\forall\, u : \mathsf{L}, x : \mathsf{C}\ .\ x \downarrow \mathsf{E}_{\mathrm{fo},u} \equiv \\
\quad (\exists\, e : \mathsf{E}_{\mathrm{sy},u}\ .\ x = [\ulcorner\mathsf{op\text{-}app}\urcorner, [\ulcorner\mathsf{op}\urcorner, e, \ulcorner\mathsf{formula}\urcorner]]) \vee \\
\quad (\exists\, e_1 : \mathsf{E}_{\mathrm{sy},u}, e_2, e_3 : \mathsf{E}_{u}\ . \\
\qquad [\ulcorner\mathsf{op\text{-}app}\urcorner, [\ulcorner\mathsf{op}\urcorner, e_1]\hat{}e_2\hat{}[\ulcorner\mathsf{formula}\urcorner]]\hat{}e_3 \in \mathsf{E}_{\mathrm{fo},u} \wedge \\
\qquad ((\exists\, e : \mathsf{E}_{\mathrm{ty},u}\ . \\
\qquad\quad x = [\ulcorner\mathsf{op\text{-}app}\urcorner, [\ulcorner\mathsf{op}\urcorner, e_1]\hat{}e_2\hat{}[\ulcorner\mathsf{type}\urcorner, \ulcorner\mathsf{formula}\urcorner]]\hat{}e_3\hat{}[e]) \vee \\
\qquad\ (\exists\, e : \mathsf{E}_{\mathrm{ty},u}, e' : \mathsf{E}_{\mathrm{te},u}\ . \\
\qquad\quad x = [\ulcorner\mathsf{op\text{-}app}\urcorner, [\ulcorner\mathsf{op}\urcorner, e_1]\hat{}e_2\hat{}[e, \ulcorner\mathsf{formula}\urcorner]]\hat{}e_3\hat{}[e']) \\
\qquad\ (\exists\, e : \mathsf{E}_{\mathrm{fo},u}\ .
\end{array}
$$

$$
\begin{array}{l}
\qquad x = [\ulcorner \mathsf{op\text{-}app} \urcorner, [\ulcorner \mathsf{op} \urcorner, e_1] \hat{} e_2 \hat{} [\ulcorner \mathsf{formula} \urcorner, \ulcorner \mathsf{formula} \urcorner]] \\
\qquad\quad \hat{} e_3 \hat{} [e]])) \vee \\
(\exists\, e_1 : \mathsf{E}_{\mathrm{sy},u}, e_2 : \mathsf{E}_{\mathrm{ty},u}, e_3 : \mathsf{E}_{\mathrm{fo},u} \,.\, x = \ulcorner (\exists\, \lfloor e_1 \rfloor : \lfloor e_2 \rfloor \,.\, \lfloor e_3 \rfloor) \urcorner) \vee \\
(\exists\, e : \mathsf{E}_{\mathrm{te},u} \,.\, x = \ulcorner [\![\lfloor e \rfloor]\!]_{\mathrm{fo}} \urcorner).
\end{array}
$$

8.4 Soundness

Fix a normal theory $T = (L, \Gamma)$ for the rest of this subsection. We will prove that $\mathbf{C}_L$ is sound (with respect to the set of all normal theories over L and the set of all formulas of L) by showing that its rules of inference preserve validity in every standard model of T and its axioms are valid in T.

Proposition 8.4.1 (Propositional Connectives) *The built-in operators and defined operators that represent propositional connectives in Chiron—the formula operators named* formula-equal*,* not*,* or*,* true*,* false*,* and*, and* implies*—have their usual meanings in every standard model of T.*

Proof Let M be a standard model of T. The built-in operator names formula-equal, not, and or are assigned their usual meanings in M by the definition of a standard model. The defined operators true, false, and, and implies are given their usual meanings in M by their definitions. □

Proposition 8.4.2 (Quantification over Empty Types) *Let $M = (S, V)$ be a standard model of T and $\varphi \in \mathsf{assign}(S)$. If $V_\varphi(\alpha)$ is empty, then:*

1. $V_\varphi(\exists\, x : \alpha \,.\, A) = \mathrm{F}$.
2. $V_\varphi(\forall\, x : \alpha \,.\, A) = \mathrm{T}$.

Proof $V_\varphi(\exists\, x : \alpha \,.\, A) = \mathrm{F}$ by the definition of V on existential quantifications. This implies $V_\varphi(\forall\, x : \alpha \,.\, A) = \mathrm{T}$ by the notational definition of $\forall$ and Proposition 8.4.1. □

Lemma 8.4.3 (Modus Ponens) *The rule Modus Ponens preserves validity in every standard model of T.*

Proof Let $M = (S, V)$ be a standard model of T. Suppose $M \models A$ and $M \models A \supset B$. It follows immediately by Proposition 8.4.1 that $M \models B$. Therefore, Modus Ponens preserves validity in every standard model of T. □

Lemma 8.4.4 (Universal Generalization) *The rule Universal Generalization preserves validity in every standard model of T.*

Proof Let $M = (S, V)$ be a standard model of T. Suppose $M \models A$. Then (i) $V_\varphi(A) = \mathrm{T}$ for all $\varphi \in \mathsf{assign}(S)$. We need to show that $M \models \forall\, x : \alpha \;.\; A$. That is, we need to show (ii) $V_\varphi(\forall\, x : \alpha \;.\; A) = \mathrm{T}$ for all $\varphi \in \mathsf{assign}(S)$. Let $\varphi \in \mathsf{assign}(S)$. If $V_\varphi(\alpha)$ is empty, then $V_\varphi(\forall\, x : \alpha \;.\; A) = \mathrm{T}$ by Proposition 8.4.2. So we may assume that $V_\varphi(\alpha)$ is nonempty. Let d be in $V_\varphi(\alpha)$. Then $\varphi[x \mapsto d](x)$ is in $V_\varphi(\alpha)$, and so by (i), $V_{\varphi[x \mapsto d]}(A) = \mathrm{T}$. This implies $V_\varphi(\forall\, x : \alpha \;.\; A) = \mathrm{T}$ by the notational definition of $\forall$ and the definition of V on existential quantifications. Therefore, (ii) holds and thus Universal Generalization preserves validity in every standard model of T. □

Lemma 8.4.5 (Universal Quantifier Shifting) *The rule Universal Quantifier Shifting preserves validity in every standard model of T.*

Proof Let $M = (S, V)$ be a standard model of T. Suppose (i) $M \models \neg\mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner A \urcorner)$ and (ii) $V_\varphi(\forall\, x : \alpha \;.\; (A \vee B)) = \mathrm{T}$ for $\varphi \in \mathsf{assign}(S)$. We must show $V_\varphi(A \vee (\forall\, x : \alpha \;.\; B)) = \mathrm{T}$. Our argument is by cases:

> **Case 1**: $V_\varphi(A) = \mathrm{T}$. Hence, by Proposition 8.4.1, $V_\varphi(A \vee (\forall\, x : \alpha \;.\; B)) = \mathrm{T}$.
>
> **Case 2**: $V_\varphi(A) = \mathrm{F}$. It suffices to show (iii) $V_\varphi(\forall\, x : \alpha \;.\; B) = \mathrm{T}$ since this implies $V_\varphi(A \vee (\forall\, x : \alpha \;.\; B)) = \mathrm{T}$ by Proposition 8.4.1. If $V_\varphi(\alpha)$ is empty, then (iii) holds by Proposition 8.4.2. So let d be in $V_\varphi(\alpha)$. By the notational definition of $\forall$, the definition of V on existential quantifications, and (ii), $V_{\varphi[x \mapsto d]}(A \vee B) = \mathrm{T}$. This implies $V_{\varphi[x \mapsto d]}(A) = \mathrm{T}$ or $V_{\varphi[x \mapsto d]}(B) = \mathrm{T}$ by Proposition 8.4.1. By the hypothesis, (i), and Lemma 6.4.5, $V_{\varphi[x \mapsto d]}(A) = \mathrm{F}$. This implies $V_{\varphi[x \mapsto d]}(B) = \mathrm{T}$, and hence (iii) holds by the notational definition of $\forall$ and the definition of V on existential quantifications.

□

Lemma 8.4.6 (Universal Instantiation) *The rule Universal Instantiation preserves validity in every standard model of T.*

Proof Let $M = (S, V)$ be a standard model of T. Suppose (i) $M \models \mathsf{sub}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner A \urcorner)\!\downarrow$, (ii) $M \models \mathsf{free\text{-}for}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner A \urcorner)$, and (iii) $V_\varphi((\forall\, x : \alpha \;.\; A) \wedge a \downarrow \alpha) = \mathrm{T}$ for $\varphi \in \mathsf{assign}(S)$. We must show

$$V_\varphi([\![\mathsf{sub}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner A \urcorner)]\!]_{\mathrm{fo}}) = \mathrm{T}.$$

By Proposition 8.4.1, (iii) implies (iv) $V_\varphi(\forall\, x : \alpha\ .\ A) = \mathrm{T}$ and (v) $V_\varphi(a \downarrow \alpha) = \mathrm{T}$. (v) implies (vi) $V_\varphi(a) \neq \bot$ and (vii) $V_\varphi(a)$ is in $V_\varphi(\alpha)$. (iv) and (vii) imply (viii) $V_{\varphi[x \mapsto V_\varphi(a)]}(A) = \mathrm{T}$ by the notational definition of $\forall$ and the definition of V on existential quantifications. (i), (ii), (vi), and (viii) imply

$$V_\varphi([\![\mathsf{sub}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner A \urcorner)]\!]_{\mathrm{fo}}) = V_{\varphi[x \mapsto V_\varphi(a)]}(A) = \mathrm{T}$$

by Lemma 6.4.10. □

Lemma 8.4.7 (Definite Description) *The rule Definite Description preserves validity in every standard model of T.*

Proof Let $M = (S, V)$ be a standard model of T. Suppose (i) $M \models \mathsf{sub}(\ulcorner (\iota\, x : \alpha\ .\ A) \urcorner, \ulcorner x \urcorner, \ulcorner A \urcorner)\downarrow$, (ii) $M \models \mathsf{free\text{-}for}(\ulcorner (\iota\, x : \alpha\ .\ A) \urcorner, \ulcorner x \urcorner, \ulcorner A \urcorner)$ and (iii) $V_\varphi(\exists !\, x : \alpha\ .\ A) = \mathrm{T}$ for $\varphi \in \mathsf{assign}(S)$. We need to show

$$V_\varphi([\![\mathsf{sub}(\ulcorner (\iota\, x : \alpha\ .\ A) \urcorner, \ulcorner x \urcorner, \ulcorner A \urcorner)]\!]_{\mathrm{fo}}) = \mathrm{T}.$$

By the notational definition of unique existential quantification and the definition of V on existential quantifications, (iii) implies (iv) there is a unique d in $V_\varphi(\alpha)$ such that $V_{\varphi[x \mapsto d]}(A) = \mathrm{T}$, and by the definition of V on definite descriptions, (iv) implies (v) there is a d in $V_\varphi(\alpha)$ such that $V_\varphi(\iota\, x : \alpha\ .\ A) = d$. (v) implies (vi) $V_\varphi(\iota\, x : \alpha\ .\ A) \neq \bot$. (i), (ii), (iv), (v), and (vi) imply

$$\begin{aligned} V_\varphi([\![\mathsf{sub}(\ulcorner (\iota\, x : \alpha\ .\ A) \urcorner, \ulcorner x \urcorner, \ulcorner A \urcorner)]\!]_{\mathrm{fo}}) &= V_{\varphi[x \mapsto V_\varphi(\iota\, x : \alpha.A)]}(A) \\ &= V_{\varphi[x \mapsto d]}(A) \\ &= \mathrm{T} \end{aligned}$$

by Lemma 6.4.10. □

Lemma 8.4.8 (Indefinite Description) *The rule Indefinite Description preserves validity in every standard model of T.*

Proof Similar to the proof of Lemma 8.4.7. □

Lemma 8.4.9 (Functions) *The four rules of inference for functions preserve validity in every standard model of T.*

Proof The rule Type Application specifies the value of a type application $\alpha(a)$ by Proposition 8.4.1, the definitions of $\mathsf{free\text{-}in}$, fun, and $\mathsf{ord\text{-}pair}$, and the definition of V on type applications. The rule Dependent Function Type specifies the value of a dependent function type $(\Lambda\, x : \alpha\ .\ \beta)$ by

Proposition 8.4.1, the definition of free-in and fun, and the definition of V on dependent function types. The rule Function Application specifies the value of a function application $f(a)$ by Proposition 8.4.1, the definitions of free-in, and fun, and ord-pair, and the definition of V on type and function applications. And, finally, the rule Function Abstraction specifies the value of a function abstraction $(\lambda\, x : \alpha\ .\ b)$ of a function abstraction by Proposition 8.4.1, the definition of free-in, and the definition of V on dependent function types, function applications, and function abstractions. Therefore, each of the four rules of inference for functions preserves validity in every standard model of T. □

Lemma 8.4.10 (Axiom Schemas 1) *Each instance of the axiom schemas in Axiom Schemas 1 is valid in T.*

Proof The instances of these five axiom schemas are tautologies under the usual interpretations of the propositional connectives. Therefore, by Proposition 8.4.1, each such instance is valid in T. □

Lemma 8.4.11 (Axiom Schemas 2) *Each instance of the axiom schemas in Axiom Schemas 2 is valid in T.*

Proof The instances of these six axiom schemas are valid in T by clauses n, o, and p of the definition of I in a structure for L, the notational definition of $\simeq$, and the compositionality of the definition of the standard valuation. □

Lemma 8.4.12 (Axiom Schemas 3) *Each instance of the axiom schemas in Axiom Schemas 3 is valid in T.*

Proof The instances of the eight axiom schemas are valid in T by Proposition 8.4.1, the definition of I in a structure for L, and the definition of the standard valuation on operator applications. □

Lemma 8.4.13 (Axiom Schemas 4) *Each instance of the axiom schemas in Axiom Schemas 4 is valid in T.*

Proof The instances of these ten axiom schemas are valid in T by Proposition 8.4.1 and clauses c–d and f–m, respectively, of the definition of I in a structure for L. □

Lemma 8.4.14 (Axiom Schemas 5) *Each instance of the axiom schemas in Axiom Schemas 5 is valid in T.*

Proof The instances of this axiom schema are valid in T by Proposition 8.4.1 and the definition of the standard valuation on variables. □

Lemma 8.4.15 (Axiom Schemas 6) *Each instance of the axiom schemas in Axiom Schemas 6 is valid in T.*

Proof The instances of the first axiom schema are valid in T by the definition of D_{s} and clause n of the definition of I in a structure for L. The instances of the second axiom schema are valid in T by the definition of ∇ and Proposition 8.4.2. □

Lemma 8.4.16 (Axiom Schemas 7) *Each instance of the axiom schemas in Axiom Schemas 7 is valid in T.*

Proof Let $M = (S, V)$ be a standard model of T. The set-theoretic built-in operator names class, set, and in are given their usual meanings in M by the definition of a standard model. The defined set-theoretic operators empty-set, pair, ord-pair, subclass, intersection, complement, fun, dom, sum, and power are given their usual meanings in M by their definitions. S is constructed from a prestructure $(D, \in)$ that satisfies the axioms of NBG set theory. The instances of the axiom schemas in Axiom Schema 7 are exactly these axioms expressed in the language of Chiron. Therefore, the instances are valid in M and thus valid in T. □

Lemma 8.4.17 (Axiom Schemas 8) *Each instance of the axiom schemas in Axiom Schemas 8 is valid in T.*

Proof The instances of these two axiom schemas are valid in T by Proposition 8.4.1, the definition of the standard valuation on conditional terms, and the notational definition of $\simeq$. □

Lemma 8.4.18 (Axiom Schemas 9) *Each instance of the axiom schemas in Axiom Schemas 9 is valid in T.*

Proof Let $M = (S, V)$ be a standard model of T and $\varphi \in \mathsf{assign}(S)$.

Schema 1 Follows from the proof of Lemma 8.4.7.

Schema 2 Assume (i) $V_\varphi(\neg(\exists\,!\,x : \alpha\ .\ A)) = \mathrm{T}$. By Proposition 8.4.1, we must show (ii) $V_\varphi((\iota\, x : \alpha\ .\ A)\uparrow) = \mathrm{T}$. By the notational definition of unique existential quantification and the definition of V on existential quantifications, (i) implies (iii) there is no unique d in $V_\varphi(\alpha)$ such that $V_{\varphi[x \mapsto d]}(A) = \mathrm{T}$, and by the definition of V on definite descriptions, (iii) implies (ii). □

Lemma 8.4.19 (Axiom Schemas 10) *Each instance of the axiom schemas in Axiom Schemas 10 is valid in T.*

Proof The proof for the first two axiom schema is similar to the proof of Lemma 8.4.18. The proof for the third axiom schema follows immediately from the definition of the standard valuation on indefinite descriptions. □

Lemma 8.4.20 (Axiom Schemas 11) *Each instance of the axiom schemas in Axiom Schemas 11 is valid in T.*

Proof The instances of these three axiom schemas are valid in T by the definition of H in a structure for L, the definition of the defined operator named ord-pair, and the definition of the standard valuation on quotations. □

Lemma 8.4.21 (Axiom Schemas 12) *Each instance of the axiom schemas in Axiom Schemas 12 is valid in T.*

Proof The instances of these six axiom schemas are valid in T by Proposition 8.4.1, Lemma 6.3.1, and the definition of the defined operator named gea. □

Lemma 8.4.22 (Axiom Schemas 13) *Each instance of the axiom schemas in Axiom Schemas 13 is valid in T.*

Proof Let $M = (S, V)$ be a standard model of T and I be the last component of S. The first schema specifies op-names to be I(op-names). The second to ninth schemas specify expr-sym and expr-op-name to be I(expr-sym) and I(expr-op-name), respectively. Using induction, the first tenth schema specifies expr to be I(expr). Using mutual recursion, the remaining schemas define expr-op, expr-type, expr-term-type, expr-term, and expr-formula to be I(expr-op), I(expr-type), I(expr-term-type), I(expr-term), and I(expr-formula), respectively. Therefore, each instance of these schemas is valid in T. □

Theorem 8.4.23 (Soundness) *$\boldsymbol{C}_L$ is sound with respect to the set of all normal theories over L and the set of all formulas of L.*

Proof Let $T = (L, \Gamma)$ be a normal theory of Chiron and A be a formula of L. By Lemma 8.4.3–8.4.9, the rules of inference of $\mathbf{C}_L$ preserve validity in every standard model of T. By Lemmas 8.4.10–8.4.22, each axiom of $\mathbf{C}_L$ is valid in T. Therefore, if $T \vdash A$, then $T \models A$, and hence $\mathbf{C}_L$ is sound with respect to all formulas of L. □

8.5 Some Metatheorems

Fix an eval-free normal theory $T = (L, \Gamma)$ for the rest of this subsection.

Let $\mathbf{C}^*_L$ be $\mathbf{C}_L$ where the schema variables in Rules 3–10 are restricted to eval-free expressions. Let $T \vdash^* A$ mean there is a proof of A from T in $\mathbf{C}^*_L$. T is *consistent** if there is some formula A of L such that $T \vdash^* A$ does not hold. Obviously $T \vdash^* A$ implies $T \vdash A$.

Theorem 8.5.1 (Tautology) *If $T \vdash^* A_1$, ..., $T \vdash^* A_n$ and $(A_1 \wedge \cdots \wedge A_n) \supset B$ is a tautology for $n \geq 1$, then $T \vdash^* B$. Also, if B is tautology, then $T \vdash^* B$.*

Proof Follows from Axiom Schemas 1 and the Modus Ponens rule of inference by a standard argument. □

Lemma 8.5.2 *Let a be an eval-free term, x be a symbol, and e, e' be eval-free proper expressions of L.*

1. *If x is free in e, then $T \vdash^* \mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner e \urcorner)$.*
2. *If x is not free in e, $T \vdash^* \neg\mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner e \urcorner)$.*
3. *If a is free for x in e, then $T \vdash^* \mathsf{free\text{-}for}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner e \urcorner)$.*
4. *If a is not free for x in e, then $T \vdash^* \neg\mathsf{free\text{-}for}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner e \urcorner)$.*
5. *If e is syntactically closed, then $T \vdash^* \mathsf{syn\text{-}closed}(\ulcorner e \urcorner)$.*
6. *If $T^L_{\mathrm{ker}} \models \mathsf{cleanse}(\ulcorner e \urcorner) = \ulcorner e \urcorner$, then $T \vdash^* \mathsf{cleanse}(\ulcorner e \urcorner) = \ulcorner e \urcorner$.*
7. *If $T^L_{\mathrm{ker}} \models \mathsf{sub}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner e \urcorner) = \ulcorner e' \urcorner$, then $T \vdash^* \mathsf{sub}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner e \urcorner) = \ulcorner e' \urcorner$.*

Proof By induction on the length of e using the Tautology theorem, Axiom Schemas 2, and the definitions of free-in, free-for, syn-closed, cleanse, and sub. The Universal Generalization rule of inference is also needed for part 5. □

Theorem 8.5.3 (Deduction) *Let A be a syntactically closed, eval-free formula of L and $T' = (L, \Gamma \cup \{A\})$. If $T' \vdash^* B$, then $T \vdash^* A \supset B$.*

Proof Follows from Lemmas 6.4.1 and 8.5.2; Axiom Schemas 1; and the Modus Ponens, Universal Generalization, and Universal Quantifier Shifting rules of inference by a standard argument. (A standard argument fails if $\mathbf{C}_L$ is used in place of $\mathbf{C}^*_L$.) □

Lemma 8.5.4 *Let A be a syntactically closed, eval-free formula of L and $T' = (L, \Gamma \cup \{A\})$. If T' is not consistent*, then $T \vdash^* \neg A$.*

Proof It follows from the hypotheses of the lemma and the Tautology and Deduction theorems by a standard argument that $T \vdash^* \neg A$. □

Lemma 8.5.5 *If A is an eval-free formula of L, then*

$$T \vdash^* (\forall\, x : \mathsf{C}\ .\ A) \supset A.$$

Proof By the Deduction theorem, the Universal Instantiation rule of inference, the first axiom schema of Axiom Schemas 7, Lemmas 6.4.1 and 8.5.2, the definition of sub, and the fifth axiom schema of Axiom Schemas 12. □

8.6 Completeness

Let $\mathcal{T}$ be the set of all eval-free normal theories over L and $\mathcal{F}$ be the set of all eval-free formulas of L. We will prove that $\mathbf{C}^*_L$ is complete with respect to $\mathcal{T}$ and $\mathcal{F}$. This will then immediately imply that $\mathbf{C}_L$ is complete with respect to $\mathcal{T}$ and $\mathcal{F}$ as well. The key component of the proof is the following lemma that says a standard model can be constructed for any consistent* eval-free normal theory. The proof of this lemma is long and tedious.

Lemma 8.6.1 (Model Construction) *Let $T = (L, \Gamma)$ be a consistent* eval-free normal theory. Then T has a standard model $M = (S, V)$ such that, for every class x in M, there is a syntactically closed, eval-free term a of L whose value in M (with respect to any φ) is x.*

Proof Let $T' = (L, \Gamma')$ be an extension of T that is maximal consistent*. We will tacitly use the Tautology theorem, Lemma 8.5.2, and the fact that T is maximal consistent* throughout this proof.

Step 1 Let $\mathcal{C}$ be the set of syntactically closed terms c of L such that $c = c'$ is in Γ' for some syntactically closed, eval-free term c' of L. Let $\mathcal{C}'$ be the set of syntactically closed terms c of L such that $c \in \mathcal{C}$ or $c\!\uparrow$ is in Γ'. And let $\mathcal{T}$ be the set of types α such that (1) $\alpha = \mathsf{C}$ or (2) for all $c \notin \mathcal{C}$, $c \uparrow \alpha$ is in Γ'. For $c, d \in \mathcal{C}$, define $c \sim d$ iff $c = d$ is in Γ'. By Axiom Schemas 2, $\sim$ is an equivalence relation on $\mathcal{C}$. For each $c \in \mathcal{C}$, let

$$\tilde{c} = \{d \in \mathcal{C} \mid c \sim d\}$$

be the equivalence class of c. Note that each $\tilde{c}$ contains an eval-free term.

Define $D_{\rm c} = \{\tilde{c} \mid c \in \mathcal{C}\}$. For $c, d \in \mathcal{C}$, define $\tilde{c} \in \tilde{d}$ iff $c \in d$ is in Γ'. By Axiom Schemas 2, the definition of $\in$ is well defined. Define $D_{\rm v}$, $D_{\rm s}$, $D_{\rm f}$, $D_{\rm o}$, $D_{\rm e}$, T, F, and $\perp$ as in the definition of a structure for L. Let ξ be any choice function on $D_{\rm s}$. H and I will be defined later.

Step 2 For a syntactically closed type α of L, define

$$U(\alpha) = \{\tilde{c} \in D_{\rm c} \mid c \in \mathcal{C} \text{ and } c \downarrow \alpha \text{ is in } \Gamma'\}$$

if $\alpha \in \mathcal{T}$ and $U(\alpha) = \mathsf{C}$ if $\alpha \notin \mathcal{T}$. For a syntactically closed term a of L, define $U(a) = \tilde{a}$ if $a \in \mathcal{C}$ and $U(a) = \perp$ if $a \notin \mathcal{C}$. For a syntactically closed formula A of L, define $U(A) = \mathrm{T}$ if A in Γ' and $U(A) = \mathrm{F}$ if $\neg A$ is in Γ'.

Claim 1

1. For all syntactically closed types α and β of L with $\alpha, \beta \in \mathcal{T}$, $U(\alpha) = U(\beta)$ iff $\alpha =_{\rm ty} \beta$ is in Γ'.

2. For all syntactically closed terms a and b of L with $a, b \in \mathcal{C}'$, $U(a) = U(b)$ iff $a \simeq b$ is in Γ'.

3. For all syntactically closed formula A and B of L, $U(A) = U(B)$ iff $A \equiv B$ is in Γ'.

Proof

1. $U(\alpha) = U(\beta)$ iff $\{\tilde{c} \in D_{\rm c} \mid c \in \mathcal{C} \text{ and } c \downarrow \alpha \text{ is in } \Gamma'\} = \{\tilde{c} \in D_{\rm c} \mid c \in \mathcal{C} \text{ and } c \downarrow \beta \text{ is in } \Gamma'\}$ iff for all $\tilde{c} \in D_{\rm c}$, ($c \downarrow \alpha$ in Γ' iff $c \downarrow \beta$ in Γ') iff for all $\tilde{c} \in D_{\rm c}$, $c \downarrow \alpha \equiv c \downarrow \beta$ is in Γ' by the Tautology theorem iff $\forall x : \mathsf{C} \,.\, x \downarrow \alpha \equiv x \downarrow \beta$ is in Γ' by the Universal Instantiation rule of inference iff $\alpha =_{\rm ty} \beta$ is in Γ' by the first axiom schema of Axiom Schemas 6. Therefore, $U(\alpha) = U(\beta)$ iff $\alpha =_{\rm ty} \beta$ is in Γ'.

2. $U(a) = U(b) \neq \perp$ iff $\tilde{a} = \tilde{b}$ iff $a \sim b$ iff (1) $a = b$ is in Γ'. $U(a) = U(b) = \perp$ iff (2) $a{\uparrow}$ and $b{\uparrow}$ are in Γ'. (1) and (2) hold iff $a \simeq b$ is in Γ' by the definition of quasi-equal. Therefore, $U(a) = U(b)$ iff $a \simeq b$ is in Γ'.

3. $U(A) = U(B) = \mathrm{T}$ iff (1) A and B are in Γ', and $U(A) = U(B) = \mathrm{F}$ iff (2) $\neg A$ and $\neg B$ are in Γ'. (1) and (2) hold iff $A \equiv B$ is in Γ'. Therefore, $U(A) = U(B)$ iff $A \equiv B$ is in Γ'.

This completes the proof of Claim 1.

Step 3 Recall that each quotation is syntactically closed and eval-free. Let G be the mapping from $\mathcal{S} \cup \mathcal{O}$ to D_{c} such that, for all $s \in \mathcal{S} \cup \mathcal{O}$, $G(s) = U(\ulcorner s \urcorner)$. By the first axiom schema of Axiom Schemas 11 and the fourth axiom schema of Axiom Schemas 13, the range of G is a subset of D_{v}. By part 2 of Claim 1 and the third axiom schema of Axiom Schemas 11, G is injective. Define H from G as in the definition of a structure for L.

Step 4 Let o be an operator name of L with the signature form $s_1, \ldots, s_{n+1}$ where $n \geq 0$. Then let $k_1, \ldots, k_{n+1}$ be the signature where $k_i = s_i$ if s_i is type or $\mathsf{formula}$ and $k_i = \mathsf{C}$ if s_i is term for all i with $1 \leq i \leq n+1$, and let $D_i = D_{\mathrm{s}}$ if s_i is type, $D_i = D_{\mathrm{c}} \cup \{\bot\}$ if s_i is term, and $D_i = \{\mathrm{T}, \mathrm{F}\}$ if s_i is $\mathsf{formula}$ for all i with $1 \leq i \leq n+1$. Define $U(o)$ to be any operation σ from $D_1 \times \cdots \times D_n$ into D_{n+1} such that, for all syntactically closed expressions $e_1, \ldots, e_n$ where e_i is a type in $\mathcal{T}$ if $s_i = \mathsf{type}$, a term in $\mathcal{C}'$ if $s_i = \mathsf{term}$, and a formula if $s_i = \mathsf{formula}$ for all i with $1 \leq i \leq n$,

$$\sigma(U(e_1), \ldots, U(e_n)) = U((o :: k_1, \ldots, k_{n+1})(e_1, \ldots, e_n)).$$

By Claim 1 and Axiom Schemas 2, $\sigma(U(e_1), \ldots, U(e_n))$ is well defined by this equation.

Claim 2 U on operator names satisfies the specification for I in the definition of a structure for L.

Proof Let o be an operator name of L with the signature form $s_1, \ldots, s_{n+1}$ where $n \geq 0$. We must show that $U(o)$ satisfies the specification for $I(o)$ in the definition of a structure for L. First, we have to show that $U(o)$ is an operation from $D_1 \times \cdots \times D_n$ into D_{n+1} where the D_i are defined as above. This is true immediately by the definition of $U(o)$. Second, if o is a built-in operator name of Chiron, we have to verify that the clause of part 5 of the definition of I corresponding to o is satisfied.

Clause a: set.

$$\begin{aligned}
U(\mathsf{set})(\,) &= U(\mathsf{V}) \\
&= \{\tilde{c} \in D_{\mathrm{c}} \mid c \downarrow \mathsf{V} \text{ is in } \Gamma'\} \\
&= \{\tilde{c} \in D_{\mathrm{c}} \mid \exists\, y : \mathsf{C} \,.\, c \in y \text{ is in } \Gamma'\} \\
&= \{\tilde{c} \in D_{\mathrm{c}} \mid \text{for some } \tilde{d} \in D_{\mathrm{c}}, c \in d \text{ is in } \Gamma'\} \\
&= \{\tilde{c} \in D_{\mathrm{c}} \mid \text{for some } \tilde{d} \in D_{\mathrm{c}}, \tilde{c} \in \tilde{d}\} \\
&= D_{\mathrm{v}}.
\end{aligned}$$

The second line is by the definition of U on syntactically closed types; the third line is by the second axiom schema of Axiom Schemas 7 and the Universal Instantiation rule of inference; fourth line is by the Indefinite Description rule of inference; the fifth line is by the definition of $\in$ on D_{c}; and sixth line is by the definition of D_{v}. Therefore, $U(\mathsf{set})$ satisfies clause a.

Clause b: class.

$$\begin{aligned} U(\mathsf{class})(\,) &= U(\mathsf{C}) \\ &= \{\tilde{c} \in D_{\mathrm{c}} \mid c\downarrow \mathsf{C} \text{ is in } \Gamma'\} \\ &= \{\tilde{c} \in D_{\mathrm{c}} \mid c\downarrow \text{ is in } \Gamma'\} \\ &= D_{\mathrm{c}}. \end{aligned}$$

The second line is by the definition of U on syntactically closed types; the third line is by the definition of the notation $c\downarrow$; and the fourth line is by the definition of D_{c}. Therefore, $U(\mathsf{class})$ satisfies clause b.

Clauses c–l: op-names, lang, expr-sym, expr-op-name, expr, expr-op, expr-type, expr-term-type, expr-formula. Follows from Axiom Schemas 4 and 13.

Clause m: in. Let x and y be in $D_{\mathrm{c}} \cup \{\bot\}$. Then $x = U(c)$ and $y = U(d)$ for some syntactically closed terms c and d. $U(\mathsf{in})(x, y) = U(\mathsf{in})(U(c), U(d)) = U(c \in d)$. $U(c \in d) = \mathrm{T}$ iff $c \in d$ is in Γ' iff $U(c), U(d) \in D_{\mathrm{c}}$ and $U(c) \in U(d)$ by the definition of $\in$ on $\mathcal{C}$. $U(c \in d) = \mathrm{F}$ iff $c \notin d$ is in Γ' iff either (1) $U(c), U(d) \in D_{\mathrm{c}}$ and $U(c) \notin U(d)$ by the definition of $\in$ on $\mathcal{C}$ or (2) $U(c) = \bot$ or $U(d) = \bot$ by the second axiom schema of Axiom Schema 4. Therefore, $U(\mathsf{in})$ satisfies clause m.

Clauses n–p: type-equal, term-equal, formula-equal. Follow from Axiom Schemas 2. The third axiom schema of Axiom Schema 4 is needed for term-equal.

Clauses q–r: not, or. Follow from Axiom Schemas 1.

This completes the proof of Claim 2.

If we define $I(o) = U(o)$ for all $o \in \mathcal{O}$, then

$$S = (D_{\mathrm{v}}, D_{\mathrm{c}}, D_{\mathrm{s}}, D_{\mathrm{f}}, D_{\mathrm{o}}, D_{\mathrm{e}}, \in, \mathrm{T}, \mathrm{F}, \bot, \xi, H, I)$$

is a structure for L *provided* $(D_{\mathrm{c}}, \in)$ *is a prestructure*. We will show that $(D_{\mathrm{c}}, \in)$ is a prestructure in Step 6 of the proof.

Step 5 Let e be an eval-free type, term, or formula, and let $\varphi \in \mathsf{assign}(S)$. Assume that the members of $\mathcal{S} \cup \mathcal{O}$ are linearly ordered by $<_{\mathcal{S}\cup\mathcal{O}}$. For $n \geq 0$, let

$$\mathsf{S}^{a_1 \cdots a_n}_{x_1 \cdots x_n} e = \begin{cases} e & \text{if } n = 0 \\ [\![\mathsf{sub}(\ulcorner a_1 \urcorner, \ulcorner x_1 \urcorner, \ulcorner \mathsf{S}^{a_2 \cdots a_{n-1}}_{x_2 \cdots x_{n-1}} e \urcorner)]\!]_{k[e]} & \text{if } n > 0 \end{cases}$$

where

$$\{x_1, \ldots, x_n\} = \{x \in \mathcal{S} \cup \mathcal{O} \mid T \vdash^* \mathsf{free\text{-}in}(\ulcorner x \urcorner, \ulcorner e \urcorner)\},$$

$x_1 <_{\mathcal{S}\cup\mathcal{O}} \cdots <_{\mathcal{S}\cup\mathcal{O}} x_n$, and a_i is the first eval-free member (in some fixed enumeration) of $\mathcal{C}$ such that $\varphi(x_i) = U(a_i)$ for all i with $1 \leq i \leq n$. Then define

$$e^{\varphi} = \mathsf{S}^{a_1 \cdots a_n}_{x_1 \cdots x_n} e$$

and

$$e^{\varphi - x_i} = \mathsf{S}^{a_1 a_2 \cdots a_{i-1} a_{i+1} \cdots a_n}_{x_1 x_2 \cdots x_{i-1} x_{i+1} \cdots x_n} e$$

where $1 \leq i \leq n$.

Claim 3 $\mathsf{S}^{a_i}_{x_i}(e^{\varphi - x_i}) = e^{\varphi}$.

Proof Follows from e being eval-free and $a_1, \ldots, a_n$ being syntactically closed and eval-free. This completes the proof of Claim 3.

Let V be defined by (1) $V_{\varphi}(e)$ is undefined when e is improper, (2) $V_{\varphi}(e)$ is $I(o)$ when $e = (o :: k_1, \ldots, k_{n+1})$ is proper, (3) $V_{\varphi}(e) = U(e^{\varphi})$ when e is an eval-free type, term, or formula, and (4) $V_{\varphi}(e)$ is the standard valuation for S (provided $(D_{\mathrm{c}}, \in)$ is a prestructure) applied to e and φ when e is a non-eval-free type, term, or formula. When e is an eval-free type, term, or formula, e^{φ} is clearly a syntactically closed, eval-free type, term, or formula, and so V is a valuation for S (provided $(D_{\mathrm{c}}, \in)$ is a prestructure).

Claim 4 V is the standard valuation for S (provided $(D_{\mathrm{c}}, \in)$ is a prestructure).

Proof We must show that, for all $e \in \mathcal{E}_L$ and $\varphi \in \mathsf{assign}(S)$, V is the standard evaluation applied to e and φ. This is true by the definition of V

when e is improper, an operator, or a non-eval-free type, term, or formula. So let e be an eval-free type, term, or formula and $\varphi \in \mathsf{assign}(S)$. Our proof will be by induction on the length of e. There are 14 cases corresponding to the 14 clauses of the definition of a standard valuation. The proofs for cases 1, 2, and 14 are trivial. The proofs for cases 4 and 10 are given in detail. And the proofs for the remaining cases are briefly sketched.

Case 1: e is improper. This case does not occur because e is proper.

Case 2: $e = (o :: k_1, \ldots, k_{n+1})$. This case does not occur because e is not an operator.

Case 3: $e = O(e_1, \ldots, e_n)$ and $O = (o :: k_1, \ldots, k_{n+1})$. Follows from Axiom Schemas 3.

Case 4: $e = (x : \alpha)$. Let a be some eval-free member of $\mathcal{C}$ such that $\varphi(x) = U(a)$. By the induction hypothesis, $V_\varphi(\alpha) = U(\alpha^\varphi)$ is the standard valuation applied to α and φ.

If $\varphi(x) \in V_\varphi(\alpha)$, then $U(a) \in U(\alpha^\varphi)$ and

$$\begin{aligned}
V_\varphi((x : \alpha)) &= U((x : \alpha)^\varphi) \\
&= U([\![\mathsf{sub}(\ulcorner a \urcorner, \ulcorner x \urcorner, \ulcorner (x : \alpha)^{\varphi - x} \urcorner)]\!]_{\mathrm{te}}) \\
&= U([\![\ulcorner \mathsf{if}(a \downarrow \lfloor \mathsf{sub}(\ulcorner a \urcorner, \ulcorner x \urcorner, [\![\alpha^{\varphi - x}]\!])\rfloor, \\
&\qquad \lfloor \mathsf{cleanse}(\ulcorner a \urcorner)\rfloor, \\
&\qquad \bot_{\mathsf{C}}) \urcorner]\!]_{\mathrm{te}}) \\
&= U(\mathsf{if}(a \downarrow [\![\mathsf{sub}(\ulcorner a \urcorner, \ulcorner x \urcorner, [\![\alpha^{\varphi - x}]\!])]\!]_{\mathrm{ty}}, \\
&\qquad [\![\mathsf{cleanse}(\ulcorner a \urcorner)]\!]_{\mathrm{te}}, \\
&\qquad \bot_{\mathsf{C}})) \\
&= U(\mathsf{if}(a \downarrow \alpha^\varphi, [\![\ulcorner a \urcorner]\!]_{\mathrm{te}}, \bot_{\mathsf{C}}) \\
&= U(\mathsf{if}(a \downarrow \alpha^\varphi, a, \bot_{\mathsf{C}})) \\
&= U(a) \\
&= \varphi(x)
\end{aligned}$$

The first line is by the definition of V; the second line is by Claim 3; the third line is by the definition of sub; the fourth line is by Lemmas 6.3.1 and 6.3.2; the fifth is by Claim 3 and part 6 of Lemma 8.5.2; the sixth is by the first schema of Axiom Schemas 7, by Axiom Schemas 8, and by the third schema of Axiom Schemas 12; and the seventh line is

by Axiom Schemas 8 and the fact that $U(a) \in U(\alpha^{\varphi})$. Therefore, if $\varphi(x) \in V_{\varphi}(\alpha)$, then $V_{\varphi}((x : \alpha)) = \varphi(x)$, and so in this case $V_{\varphi}(x : \alpha)$ is the standard valuation applied to $(x : \alpha)$ and φ.

Similarly, if $\varphi(x) \notin V_{\varphi}(\alpha)$, then $V_{\varphi}(e) = \bot$.

Case 5: $e = \alpha(a)$. Follows from the Type Application rule of inference.

Case 6: $e = (\Lambda\, x : \alpha\ .\ \beta)$. Follows from the Dependent Function Types rule of inference.

Case 7: $e = f(a)$. Follows from the Function Application rule of inference.

Case 8: $e = (\lambda\, x : \alpha\ .\ b)$. Follows from the Function Abstraction rule of inference.

Case 9: $e = \mathsf{if}(A, b, c)$. Follows from Axiom Schemas 8.

Case 10: $e = (\exists\, x : \alpha\ .\ B)$. By the induction hypothesis, $V_{\varphi}(\alpha) = U(\alpha^{\varphi})$ is the standard valuation applied to α and φ and $V_{\psi}(B) = U(B^{\psi})$ is the standard valuation applied to B and ψ for all $\psi \in \mathsf{assign}(S)$. Then

$$
\begin{aligned}
& V_{\varphi}((\exists\, x : \alpha\ .\ B)) = \textsc{t} \\
\text{iff}\quad & U((\exists\, x : \alpha\ .\ B)^{\varphi}) = \textsc{t} \\
\text{iff}\quad & U(\exists\, x : \alpha^{\varphi}\ .\ B^{\varphi - x}) = \textsc{t} \\
\text{iff}\quad & U(\llbracket \mathsf{sub}(\ulcorner c \urcorner, \ulcorner x \urcorner, \ulcorner B^{\varphi - x} \urcorner) \rrbracket_{\mathrm{fo}}) = \textsc{t} \text{ and } U(c \downarrow \alpha^{\varphi}) = \textsc{t} \\
\text{iff}\quad & U(B^{\varphi[x \mapsto U(c)]}) = \textsc{t} \text{ and } U(c) \in U(\alpha^{\varphi}) \\
\text{iff}\quad & V_{\varphi[x \mapsto V_{\varphi}(c)]}(B) = \textsc{t} \text{ and } V_{\varphi}(c) \in V_{\varphi}(\alpha)
\end{aligned}
$$

where c is $(\epsilon\, x : \alpha^{\varphi}\ .\ B^{\varphi - x})$. The second line is by the definition of V; the third line is by the definitions of e^{φ} and sub; the fourth line is by Lemma 8.5.2, the Indefinite Description rule of inference and the first schema of Axiom Schemas 10 since c is both syntactically closed and eval-free; the fifth line is by Claim 3 and the definition of $\mathsf{defined\text{-}in}$; and the sixth line is by the induction hypothesis. Therefore, $V_{\varphi}((\exists\, x : \alpha\ .\ B)) = \textsc{t}$ iff there is some d in $V_{\varphi}(\alpha)$ such that $V_{\varphi[x \mapsto d]}(B) = \textsc{t}$, and so in this case $V_{\varphi}((\exists\, x : \alpha\ .\ B))$ is the standard valuation applied to $(\exists\, x : \alpha\ .\ B)$ and φ.

Similarly, $V_{\varphi}((\exists\, x : \alpha\ .\ B)) = \textsc{f}$ iff there is no d in $V_{\varphi}(\alpha)$ such that $V_{\varphi[x \mapsto d]}(B) = \textsc{t}$.

Case 11: $e = (\iota\, x : \alpha\ .\ B)$. Follows from the Definite Description rule of inference and Axiom Schemas 9.

Case 12: $e = (\epsilon\, x : \alpha\ .\ B)$. Follows from the Indefinite Description rule of inference and Axiom Schemas 10.

Case 13: $e = \ulcorner e' \urcorner$. Follows from Axiom Schemas 11.

Case 14: $e = [\![a]\!]_k$. This case does not occur because e is eval-free.

This completes the proof of Claim 4.

Step 6 We have only now to show that $(D_{\mathrm{c}}, \in)$ is a prestructure and that M is a standard model for T. Let A be an eval-free formula in Γ'. Then $U(A) = \mathrm{T}$, and so by Claim 4, $V_\varphi(A) = U(A^\varphi) = U(A) = \mathrm{T}$ for all $\varphi \in \mathsf{assign}(S)$. Each instance of Axiom Schemas 7 is eval-closed and a member of Γ'. Thus, by Axiom Schemas 7, $(D_{\mathrm{c}}, \in)$ satisfies the axioms of NBG set theory and hence M is a model for L. And so $M \models A$ for each eval-free formula A in Γ'. Since $\Gamma \subset \Gamma'$ and each member of Γ is eval-free, $M \models A$ for each A in Γ. Therefore, M is a standard model for T. □

Theorem 8.6.2 (Consistency and Satisfiability) *Let $T = (L, \Gamma)$ be an eval-free normal theory of Chiron. Then T is consistent* iff T is satisfiable.*

Proof Let T be consistent*. Then there is a standard model M of T by the Model Construction lemma. Therefore, T is satisfiable.

Now let T be satisfiable. Then there is a standard model M of T. Assume T is not consistent*. Then $T \vdash^* \mathsf{F}$. By the Soundness theorem, $T \models \mathsf{F}$, which contradicts the definition of a standard model. Therefore, T is consistent*. □

Theorem 8.6.3 (Completeness) $\boldsymbol{C}_L$ *is complete with respect to the set of all eval-free normal theories over L and the set of all eval-free formulas of L.*

Proof Let $T = (L, \Gamma)$ be an eval-free normal theory and A be an eval-free formula of L such that $T \models A$. We need to show $T \vdash A$. Obviously it suffices to show $T \vdash^* A$. Let A' be a universal closure of A. (A' exists by Lemma 6.4.1.) A' is obviously eval-free and is syntactically closed by Lemma 6.4.3. The hypothesis implies $T \models A'$, and this implies that there is no standard model of $T' = (L, \Gamma \cup \{\neg A'\})$. Hence T' is not consistent* by the Consistency and Satisfiability theorem. Therefore, $T \vdash^* \neg\neg A'$ by Lemma 8.5.4 and hence $T \vdash^* A'$ by the Tautology theorem. Finally, $T \vdash^* A$ by Lemma 8.5.5. □

9 Interpretations

A theory interpretation [3, 4, 24] is a meaning-preserving mapping from the expressions of one theory to the expressions of another theory. An interpretation serves as a conduit for passing information (in the form of formulas) from an abstract theory to a more concrete theory, or an equally abstract theory. Interpretations are the basis for the *little theories method* [9] for organizing mathematics where mathematical knowledge and reasoning is distributed across a network of theories.

9.1 Translations

Let $L_i = (\mathcal{O}_i, \theta_i)$ be a language of Chiron for $i = 1, 2$. A *translation from* L_1 *to* L_2 is an injective, total mapping $\Phi : \mathcal{O}_1 \to \mathcal{O}_2$ such that, for each $o \in \mathcal{O}_1$, $\theta_1(o) = \theta_2(\Phi(o))$. Φ is extended to an injective, total mapping

$$\widehat{\Phi} : \mathcal{E}_{L_1} \to \mathcal{E}_{L_2}$$

by the following rules:

1. Let $e \in \mathcal{S}$. Then $\widehat{\Phi}(e) = e$.
2. Let $e \in \mathcal{O}_1$. Then $\widehat{\Phi}(e) = \Phi(e)$.
3. Let $e = (e_1, \ldots, e_n) \in \mathcal{E}_{L_1}$. Then $\widehat{\Phi}(e) = (\widehat{\Phi}(e_1), \ldots, \widehat{\Phi}(e_n))$.

Hence, for every $e \in \mathcal{E}_{L_1}$, $\widehat{\Phi}(e)$ is exactly the same as e except that each operator name o in e has been replaced by the operator name $\Phi(o)$.

Proposition 9.1.1 *Let Φ be a translation from L_1 to L_2. If e is a symbol, an operator name, an operator, a type, a term, a term of type α, or a formula of L_1, then $\widehat{\Phi}(e)$ is a symbol, an operator name, an operator, a type, a term, a term of type $\widehat{\Phi}(\alpha)$, or a formula of L_2, respectively.*

9.2 Interpretations

Let $T_i = (L_i, \Gamma_i)$ be a normal theory of Chiron for $i = 1, 2$ and Φ be a translation from L_1 to L_2. Φ *fixes* a language $L = (\mathcal{O}, \theta) \leq L_1$ if $\Phi(o) = o$ for all $o \in \mathcal{O} \setminus \{\mathsf{op\text{-}names}, \mathsf{lang}\}$.

Proposition 9.2.1 *Suppose Φ fixes $L \leq L_1$. Then $\widehat{\Phi}(e) = e$ for all expressions e of L that do not contain the operator names* op-names *or* lang.

Recall from subsection 5.4 that (eval, a, k, b) is the relativization of (eval, a, k) to the language denoted by b and that $(\mathsf{eval}, a, k, \ell)$ is logically equivalent to (eval, a, k). The *relativization* of an expression e, written $\overline{e}$, is the expression obtained from e by repeatedly replacing each occurrence in e of an expression of the form (eval, a, k) that is not within a quotation with the expression $(\mathsf{eval}, a, k, \ell)$ until every original occurrence of this kind has been replaced. Clearly, $\overline{e} = e$ if e is eval-free. The next proposition follows immediately from the fact that $(\mathsf{eval}, a, k, \ell)$ is logically equivalent to (eval, a, k).

Proposition 9.2.2 *For all $e \in \mathcal{E}_L$, e and $\overline{e}$ are logically equivalent.*

Let Φ be *normal for* T_2 if:

1. $\mathcal{O}_1$ is finite, i.e., $\mathcal{O}_1 = \{o_1, \ldots, o_n\}$ for some $n \geq 1$.
2. Φ fixes L_{ker}.
3. $T_2 \models \widehat{\Phi}(\ell) = \{\ulcorner \Phi(o_1) \urcorner, \ldots, \ulcorner \Phi(o_n) \urcorner\}$.
4. $T_2 \models \widehat{\Phi}(\mathsf{L}) =_{\mathrm{ty}} \mathsf{type}(\mathsf{power}(\{\ulcorner \Phi(o_1) \urcorner, \ldots, \ulcorner \Phi(o_n) \urcorner\}))$.

Lemma 9.2.3 *Let Φ be normal for T_2. Then $T_2 \models \widehat{\Phi}(\overline{A})$ for each $A \in \Gamma^{L_1}_{\mathrm{ker}}$.*

Proof Let $A \in \Gamma^{L_1}_{\mathrm{ker}}$. By Proposition 9.2.2 and the fact that T_2 is normal, (a) $T_2 \models \overline{A}$. Since Φ is normal for T_2, (b) Φ fixes L_{ker}, (c) $T_2 \models \widehat{\Phi}(\ell) \subseteq \ell$, and (d) $T_2 \models \mathsf{term}(\widehat{\Phi}(\mathsf{L})) \subseteq \mathsf{term}(\mathsf{L})$. (a), (b), (c), and (d) imply $T_2 \models \widehat{\Phi}(\overline{A})$. □

Suppose $M_2 = (S_2, V_2)$ is a standard model of T_2 where

$$S_2 = (D_{\mathrm{v}}, D_{\mathrm{c}}, D_{\mathrm{s}}, D_{\mathrm{f}}, D_{\mathrm{o}}, D_{\mathrm{e},2}, \in, \mathrm{T}, \mathrm{F}, \bot, \xi, H_2, I_2).$$

Let

$$S_1 = (D_{\mathrm{v}}, D_{\mathrm{c}}, D_{\mathrm{s}}, D_{\mathrm{f}}, D_{\mathrm{o}}, D_{\mathrm{e},1}, \in, \mathrm{T}, \mathrm{F}, \bot, \xi, H_1, I_1)$$

where:

1. $D_{\mathrm{e},1} = \{H_1(e) \mid e \in \mathcal{E}_{L_1}\}$.
2. $H_1(e) = H_2(\widehat{\Phi}(e))$ for each $e \in \mathcal{E}_{L_1}$.
3. $I_1(o) = I_2(\Phi(o))$ for each $o \in \mathcal{O}_1$.

Lemma 9.2.4 *Suppose Φ is normal for T_2.*

1. *S_1 is a structure for L_1.*
2. *$M_1 = (S_1, V_1)$, where V_1 is the standard valuation for S_1, is a standard model for L_1.*
3. *For all proper expressions e of L_1 and $\varphi \in \mathsf{assign}(S_1)$,*

$$V_{1,\varphi}(e) = V_{2,\varphi}(\widehat{\Phi}(\overline{e})).$$

4. *For all formulas of L_1,*

$$M_1 \models A \quad \textit{iff} \quad M_2 \models \widehat{\Phi}(\overline{A}).$$

Proof

Part 1 Since M_2 is a standard model for L_2, we need to only show that $D_{\mathrm{e},1}$, H_1, and I_1 are defined correctly.

1. $D_{\mathrm{e},1}$ is defined correctly provided H_1 is defined correctly.
2. Let $e \in \mathcal{E}_{L_1}$. We will show that $H_1(e)$ is a correct value by induction on the length of e. If $e \in \mathcal{S}$, then $H_1(e) = H_2(\widehat{\Phi}(e)) = H_2(e)$, which is a member of D_{v} that is neither the empty set nor an ordered pair. If $e \in \mathcal{O}_1$, then $\Phi(e) \in \mathcal{O}_2$ and $H_1(e) = H_2(\widehat{\Phi}(e)) = H_2(\Phi(e))$, which is a member of D_{v} that is neither the empty set nor an ordered pair. If $e = (\,)$, then $H_1(e) = H_2(\widehat{\Phi}(e)) = H_2(e)$, which is $\emptyset$. Thus, in each of these three cases, $H_1(e)$ is a correct value. Now if $e = (e_1, \ldots, e_n) \in \mathcal{E}_{L_1}$ with $n \geq 1$, then

$$\begin{aligned} H_1(e) &= H_2(\widehat{\Phi}(e)) \\ &= H_2(\widehat{\Phi}((e_1, \ldots, e_n))) \\ &= H_2((\widehat{\Phi}(e_1), \ldots, \widehat{\Phi}(e_n))) \\ &= \langle H_2(\widehat{\Phi}(e_1)), H_2((\widehat{\Phi}(e_2), \ldots, \widehat{\Phi}(e_n)))\rangle \\ &= \langle H_1(e_1), H_1((e_2, \ldots, e_n))\rangle, \end{aligned}$$

which is a correct value by the induction hypothesis.

3. We will show that I_1 satisfies the fifth condition of the definition of a structure for L_1. Let $o \in \mathcal{O}_1$. Notice that $I_2(\Phi(o))$ is an operation of the signature form $\theta_2(\Phi(o))$ since S_2 is a structure for L_2. $\theta_1(o) = \theta_2(\Phi(o))$ since Φ is a translation. Hence $I_2(\Phi(o))$ is an operation of the signature form $\theta_1(o)$. It remains only to show that clauses a–q of this condition (for built-in operator names) are satisfied. Assume $\mathcal{O}_1 = \{o_1, \ldots, o_n\}$. Let $D_{\text{on},1} = \{H_1(o_1), \ldots, H_1(o_n)\}$ and $D_{\text{on},2} = \{H_2(o) \mid o \in \mathcal{O}_2\}$, the sets of representations of operator names in $D_{\text{e},1}$ and $D_{\text{e},2}$, respectively. $D_{\text{on},1} \subseteq D_{\text{on},2}$ since $H_1(o_i) = H_2(\widehat{\Phi}(o_i)) = H_2(\Phi(o_i))$ and $\Phi(o_i) \in \mathcal{O}_2$ for each i with $1 \leq i \leq n$.

   Clause a: $o = \mathsf{set}$. Hence

$$\begin{aligned} I_1(\mathsf{set})(\,) &= I_2(\Phi(\mathsf{set}))(\,) \\ &= I_2(\mathsf{set})(\,) \\ &= D_{\text{v}}. \end{aligned}$$

   The first line is by the definition of I_1. The second line is by the fact Φ fixes L_{ker}. And the third line is by the fact S_2 is a structure for L_2.

   Clauses b, e, m–r: Similar to clause a.

   Clause c: $o = \mathsf{op\text{-}names}$. Hence

$$\begin{aligned} I_1(\mathsf{op\text{-}names})(\,) &= I_2(\Phi(\mathsf{op\text{-}names}))(\,) \\ &= V_{2,\varphi}(\widehat{\Phi}((\mathsf{op\text{-}names} :: \mathsf{term})))(\,) \\ &= V_{2,\varphi}(\widehat{\Phi}(\ell)) \\ &= V_{2,\varphi}(\{\ulcorner \Phi(o_1) \urcorner, \ldots, \ulcorner \Phi(o_n) \urcorner\}) \\ &= \{H_2(\widehat{\Phi}(o_1)), \ldots, H_2(\widehat{\Phi}(o_n))\} \\ &= \{H_1(o_1), \ldots, H_1(o_n)\} \\ &= D_{\text{on},1}. \end{aligned}$$

   The first line is by the definition of I_1. The second and third lines are by the definitions of $\widehat{\Phi}$ and the standard valuation function on operator and operator applications, respectively. The fourth line is by the facts M_2 is a standard model of T_2 and Φ is normal for T_2. The fifth line is by the definition of the standard valuation function on quotations and the definition of $\widehat{\Phi}$. The sixth is by the definition of H_1. And the seventh is by the definition of $D_{\text{on},1}$.

   Clauses d: Similar to clause c.

Clause f: $o = \mathsf{expr\text{-}op\text{-}name}$. Let $x \in D_{\mathrm{c}} \cup \{\bot\}$.

$$
\begin{array}{rl}
 & I_1(\mathsf{expr\text{-}op\text{-}name})(x) \\
= & I_2(\Phi(\mathsf{expr\text{-}op\text{-}name}))(x) \\
= & I_2(\mathsf{expr\text{-}op\text{-}name})(x) \\
= & \left\{ \begin{array}{ll} x & \text{if } x \subseteq D_{\mathrm{on},2} \\ D_{\mathrm{c}} & \text{if } x = \bot \\ \text{not } D_{\mathrm{c}} & \text{otherwise.} \end{array} \right. \\
= & \left\{ \begin{array}{ll} x & \text{if } x \subseteq D_{\mathrm{on},1} \\ D_{\mathrm{c}} & \text{if } x = \bot \\ \text{not } D_{\mathrm{c}} & \text{otherwise.} \end{array} \right.
\end{array}
$$

The second line is by the definition of I_1. The third line is by the fact Φ fixes L_{ker}. The fourth line is by the fact S_2 is a structure for L_2. The fifth line is by $D_{\mathrm{on},1} \subseteq D_{\mathrm{on},2}$.

Clauses g–l: Similar to clause f.

Therefore, S_1 is a structure for L_1.

Part 2 This part follows immediately from part 1 of this lemma.

Part 3 Our proof is by induction on the length of e. There are 13 cases corresponding to the 13 clauses of the definition of V on proper expressions.

Case 1: $O = (\mathsf{op}, o, k_1, \ldots, k_{n+1})$ is proper where $o \in \mathcal{O}_1$. Then

$$
\begin{array}{rl}
 & V_{1,\varphi}(O) \\
= & V_{1,\varphi}((\mathsf{op}, o, k_1, \ldots, k_{n+1})) \\
= & \text{“}I_1(o) \text{ restricted by } V_{1,\varphi}(k_1), \ldots, V_{1,\varphi}(k_{n+1})\text{”} \\
= & \text{“}I_2(\Phi(o)) \text{ restricted by } V_{1,\varphi}(k_1), \ldots, V_{1,\varphi}(k_{n+1})\text{”} \\
= & \text{“}I_2(\Phi(o)) \text{ restricted by } V_{2,\varphi}(\widehat{\Phi}(\overline{k_1})), \ldots, V_{2,\varphi}(\widehat{\Phi}(\overline{k_{n+1}}))\text{”} \\
= & V_{2,\varphi}((\mathsf{op}, \Phi(o), \widehat{\Phi}(\overline{k_1}), \ldots, \widehat{\Phi}(\overline{k_{n+1}}))) \\
= & V_{2,\varphi}(\widehat{\Phi}((\mathsf{op}, o, \overline{k_1}, \ldots, \overline{k_{n+1}}))) \\
= & V_{2,\varphi}(\widehat{\Phi}(\overline{(\mathsf{op}, o, k_1, \ldots, k_{n+1})})) \\
= & V_{2,\varphi}(\widehat{\Phi}(\overline{O})).
\end{array}
$$

The third and sixth lines are by the definition of the standard valuation function on operators. The fourth line is by the definitions of I_1 and

$\widehat{\Phi}$. The fifth line is by the induction hypothesis. The seventh line is by the definition of $\hat{\Phi}$. And the eighth line is by the definition of a relativization.

Case 2: $e = (\mathsf{op\text{-}app}, O, e_1, \ldots, e_n)$ is proper. Then

$$\begin{array}{ll} & V_{1,\varphi}(e) \\ = & V_{1,\varphi}((\mathsf{op\text{-}app}, O, e_1, \ldots, e_n)) \\ = & V_{1,\varphi}(O)(V_{1,\varphi}(e_1), \ldots, V_{1,\varphi}(e_n)) \\ = & V_{2,\varphi}(\widehat{\Phi}(\overline{O}))(V_{2,\varphi}(\widehat{\Phi}(\overline{e_1})), \ldots, V_{2,\varphi}(\widehat{\Phi}(\overline{e_n}))) \\ = & V_{2,\varphi}(\mathsf{op\text{-}app}, \widehat{\Phi}(\overline{O}), \widehat{\Phi}(\overline{e_1}), \ldots, \widehat{\Phi}(\overline{e_n})) \\ = & V_{2,\varphi}(\widehat{\Phi}((\mathsf{op\text{-}app}, \overline{O}, \overline{e_1}, \ldots, \overline{e_n}))) \\ = & V_{2,\varphi}(\widehat{\Phi}(\overline{(\mathsf{op\text{-}app}, O, e_1, \ldots, e_n)})) \\ = & V_{2,\varphi}(\widehat{\Phi}(\overline{e})). \end{array}$$

The third and fifth lines are by the definition of the standard valuation function on operator applications. The fourth line is by the induction hypothesis. The sixth line is by the definition of $\hat{\Phi}$. And the seventh line is by the definition of a relativization.

Cases 3–12. Similar to case 2.

Case 13a: $e = (\mathsf{eval}, a, \mathsf{type})$. Assume (a) $V_{1,\varphi}(a)$ is a member of $D_{\mathrm{e},1}$ that represents a type and $H_1^{-1}(V_{1,\varphi}(a))$ is eval-free. Let

$$\alpha = (\mathsf{E}_{\mathrm{ty},\ell} \cup \mathsf{E}_{\mathrm{te},\ell} \cup \mathsf{E}_{\mathrm{fo},\ell}).$$

Then

$$\begin{array}{ll} & V_{1,\varphi}(e) \\ = & V_{1,\varphi}((\mathsf{eval}, a, \mathsf{type})) \\ = & V_{1,\varphi}((\mathsf{eval}, a, \mathsf{type}, \ell)) \\ = & V_{1,\varphi}(\mathsf{if}(a{\downarrow}\,\alpha, [\![a]\!]_{\mathrm{ty}}, [\![\bot_{\mathsf{C}}]\!]_{\mathrm{ty}})) \\ = & \text{if } V_{1,\varphi}(a{\downarrow}\,\alpha) = \mathrm{T} \\ & \text{then } V_{1,\varphi}(H_1^{-1}(V_{1,\varphi}(a))) \\ & \text{else } \mathsf{C} \\ = & \text{if } V_{2,\varphi}(\widehat{\Phi}(\overline{a{\downarrow}\,\alpha}) = \mathrm{T} \end{array}$$

$$
\begin{array}{rl}
 & \text{then } V_{2,\varphi}(\widehat{\Phi}(H_1^{-1}(V_{2,\varphi}(\widehat{\Phi}(\overline{a})))))\\
 & \text{else } \mathsf{C}\\
= & \text{if } V_{2,\varphi}(\widehat{\Phi}(\overline{a\downarrow \alpha}) = \textsc{t}\\
 & \text{then } V_{2,\varphi}(\widehat{\Phi}(\widehat{\Phi}^{-1}(H_2^{-1}(V_{2,\varphi}(\widehat{\Phi}(\overline{a}))))))\\
 & \text{else } \mathsf{C}\\
= & \text{if } V_{2,\varphi}(\widehat{\Phi}(\overline{a\downarrow \alpha}) = \textsc{t}\\
 & \text{then } V_{2,\varphi}(H_2^{-1}(V_{2,\varphi}(\widehat{\Phi}(\overline{a}))))\\
 & \text{else } \mathsf{C}\\
= & V_{2,\varphi}(\mathsf{if}(\widehat{\Phi}(\overline{a\downarrow \alpha}), [\![\widehat{\Phi}(\overline{a})]\!]_{\mathrm{ty}}, [\![\bot_{\mathsf{C}}]\!]_{\mathrm{ty}}))\\
= & V_{2,\varphi}(\widehat{\Phi}(\mathsf{if}(\overline{a\downarrow \alpha}, [\![\overline{a}]\!]_{\mathrm{ty}}, [\![\bot_{\mathsf{C}}]\!]_{\mathrm{ty}}))\\
= & V_{2,\varphi}(\widehat{\Phi}(\mathsf{if}(\overline{a}\downarrow \alpha, [\![\overline{a}]\!]_{\mathrm{ty}}, [\![\bot_{\mathsf{C}}]\!]_{\mathrm{ty}}))\\
= & V_{2,\varphi}(\widehat{\Phi}((\mathsf{eval}, \overline{a}, \mathsf{type}, \ell)))\\
= & V_{2,\varphi}(\widehat{\Phi}(\overline{(\mathsf{eval}, a, \mathsf{type})}))\\
= & V_{2,\varphi}(\widehat{\Phi}(\overline{e}))
\end{array}
$$

The third line is by Proposition 9.2.2. The fourth and twelveth lines are by the definition of relativized evaluation. The fifth and ninth lines are by assumption (a) and the definition of the standard evaluation function on conditional terms and evaluations. The sixth line is by assumption (a) and the inductive hypothesis. The seventh line is by the definition of H_1 and the condition $V_{2,\varphi}(\widehat{\Phi}(\overline{a\downarrow \alpha}) = \textsc{t}$. The eighth line is a simple simplification. The tenth line is by the definition of $\hat{\Phi}$. The eleventh is by the fact that α is eval-free. And the thirteenth line is by the definition of a relativization.

A similar argument works when assumption (a) is false.

Case 13b: $e = (\mathsf{eval}, a, \alpha)$. Similar to case 13a.

Case 13c: $e = (\mathsf{eval}, a, \mathsf{formula})$. Similar to case 13a.

Part 4 This part follows immediately from part 3 of this lemma. □

Lemma 9.2.5 *Suppose Φ is normal for T_2, M_2 is a standard model of T_2, and $T_2 \models \widehat{\Phi}(\overline{A})$ for each $A \in \Gamma_1$. Then M_1 is a standard model of T_1.*

Proof By part 2 of Lemma 9.2.4, M_1 is a standard model for L_1. Let $A \in \Gamma_1$. Then, by the hypothesis, $T_2 \models \widehat{\Phi}(\overline{A})$, and thus $M_2 \models \widehat{\Phi}(\overline{A})$. By

part 4 of Lemma 9.2.4, this implies $M_1 \models A$. Therefore, M_1 is a standard model of T_1. □

Φ is a *(semantic) interpretation of* T_1 *in* T_2 if:

1. Φ is normal for T_2.
2. $T_1 \models A$ implies $T_2 \models \widehat{\Phi}(\overline{A})$ for all formulas A of L_1.

Proposition 9.2.6 *Suppose* Φ *is an interpretation of* T_1 *in* T_2*. Then* $T_1 \models A$ *implies* $T_2 \models \widehat{\Phi}(A)$ *for all eval-free formulas* A *of* L_1*.*

Theorem 9.2.7 (Relative Satisfiability) *Suppose* Φ *is an interpretation of* T_1 *in* T_2*. If there is a standard model of* T_2*, then there is a standard model of* T_1*.*

Proof Let M_2 be a standard model of T_2 and M_1 be defined as above. Suppose $A \in \Gamma_1$. Then $T_1 \models A$ and so by the hypothesis $T_2 \models \widehat{\Phi}(\overline{A})$. Therefore, M_1 is a standard model of T_1 by Lemma 9.2.5. □

Theorem 9.2.8 (Interpretation) *Suppose* Φ *is normal for* T_2 *and* $T_2 \models \widehat{\Phi}(\overline{A})$ *for each* $A \in \Gamma_1$*. Then* Φ *is an interpretation of* T_1 *in* T_2*.*

Proof Let A be a formula of L_1 and suppose $T_1 \models A$. We need to only show $T_2 \models \widehat{\Phi}(\overline{A})$ since Φ is normal for T_2 by hypothesis. This holds if T_2 is unsatisfiable, so without loss of generality we may assume M_2 is a standard model of T_2. We are done if we show $M_2 \models \widehat{\Phi}(\overline{A})$. Let M_1 be defined as above. By the hypothesis and Lemma 9.2.5, M_1 is a standard model of T_1. Hence $M_1 \models A$. By the hypothesis and part 4 of Lemma 9.2.4, $M_2 \models \widehat{\Phi}(\overline{A})$. □

Φ is an *anti-interpretation of* T_1 *in* T_2 if $T_2 \models \widehat{\Phi}(\overline{A})$ implies $T_1 \models A$ for all formulas A of L_1. Φ is an *faithful interpretation of* T_1 *in* T_2 if Φ is both an interpretation and anti-interpretation of T_1 in T_2. T_2 is *conservative over* T_1 if there is a faithful interpretation of T_1 in T_2.

The following lemma shows that there are theories in Chiron that cannot be extended "conservatively".

Lemma 9.2.9 *Let* $L_i = (\mathcal{O}_i, \theta_i)$ *be a language of Chiron and* $T_i = (L_i, \Gamma_i)$ *be a normal theory of Chiron for* $i = 1, 2$*. Suppose* $T_1 \leq T_2$*,* $\mathcal{O}_1 = \{o_1, \ldots, o_n\}$*,* $\mathcal{O}_1 \subset \mathcal{O}_2$*, and* Γ_1 *contains the following formula* A*:*

$$\ell = \{\ulcorner o_1 \urcorner, \ldots, \ulcorner o_n \urcorner\}.$$

Then T_2 *is unsatisfiable, i.e.,* T_2 *does not have a standard model.*

Proof Since $T_1 \leq T_2$, $A \in \Gamma_2$ and so $T_2 \models A$. Since $\mathcal{O}_1 = \{o_1, \ldots, o_n\}$ and $\mathcal{O}_1 \subset \mathcal{O}_2$, $T_2 \models \neg A$. Therefore, T_2 is unsatisfiable. □

9.3 Pseudotranslations

We will define an alternate notion of a translation in which constants may be translated to expressions other than constants. Translations of this kind are often more convenient than regular translations, but they are not applicable to all expressions.

Let $L_i = (\mathcal{O}_i, \theta_i)$ be a language of Chiron for $i = 1, 2$ and

$$\mathcal{O}_1^- = \mathcal{O}_1 \setminus \{\mathsf{op\text{-}names}, \mathsf{lang}\}.$$

A *pseudotranslation from L_1 to L_2* is a total mapping

$$\Psi : \mathcal{O}_1^- \to \mathcal{E}_{L_2}$$

such that:

1. For each 0-ary $o \in \mathcal{O}_1^-$, either $\Psi(o) \in \mathcal{O}_2$ and $\theta_1(o) = \theta_2(\Psi(o))$ or $\Psi(o)$ is a semantically closed type, term, or formula of L_2 if $\theta_1(o)$ is type, term, or $\mathsf{formula}$, respectively.
2. For each n-ary $o \in O_1^-$ with $n > 0$, $\Psi(o) \in \mathcal{O}_2$ and $\theta_1(o) = \theta_2(\Psi(o))$.
3. Ψ is injective on the $\{o \in \mathcal{O}_1^- \mid \Psi(o) \in \mathcal{O}_2\}$.

Let $\Delta(\Psi) = \{o \in \mathcal{O}_1^- \mid \Psi(o) \notin \mathcal{O}_2\}$. (Note that $\Delta(\Psi)$ is a set of 0-ary operator names that does not contain $\mathsf{op\text{-}names}$ or lang.) Ψ is extended to a partial mapping

$$\widehat{\Psi} : \mathcal{E}_{L_1} \to \mathcal{E}_{L_2}$$

by the following rules:

1. Let $e \in \mathcal{S}$. Then $\widehat{\Psi}(e) = e$.
2. Let $e \in \mathcal{O}_1^- \setminus \Delta(\Psi)$. Then $\widehat{\Psi}(e) = \Psi(e)$.
3. Let $e \in \Delta(\Psi) \cup \{\mathsf{op\text{-}names}, \mathsf{lang}\}$. Then $\widehat{\Psi}(e)$ is undefined.
4. Let $e \in \mathcal{E}_{L_1}$ be a constant of the form $(o :: \mathsf{type})(\,)$, $(o :: \mathsf{C})(\,)$, or $(o :: \mathsf{formula})(\,)$ where $o \in \Delta(\Psi)$. Then $\widehat{\Psi}(e) = \Psi(o)$.

5. Let e be a quotation that contains an operator name in $\Delta(\Psi) \cup \{\mathsf{op\text{-}names}, \mathsf{lang}\}$. Then $\widehat{\Psi}(e)$ is undefined.

6. Let $e = (e_1, \ldots, e_n) \in \mathcal{E}_{L_1}$ such that e is neither a constant of the form $(o :: \mathsf{type})(\,)$, $(o :: \mathsf{C})(\,)$, or $(o :: \mathsf{formula})(\,)$ where $o \in \Delta(\Psi)$ nor a quotation that contains an operator name in $\Delta(\Psi) \cup \{\mathsf{op\text{-}names}, \mathsf{lang}\}$. If $\widehat{\Psi}(e_1), \ldots, \widehat{\Psi}(e_n)$ are defined, then

$$\widehat{\Psi}(e) = (\widehat{\Psi}(e_1), \ldots, \widehat{\Psi}(e_n)).$$

Otherwise $\widehat{\Psi}(e)$ is undefined.

Proposition 9.3.1 *Let Ψ be a pseudotranslation from L_1 to L_2.*

1. *If e is a symbol, an operator name, an operator, a type, a term, or a formula of L_1, then $\widehat{\Psi}(e)$ is a symbol, an operator name, an operator, a type, a term, or a formula of L_2, respectively, provided $\widehat{\Psi}(e)$ is defined.*

2. *If e is an expression of L_1 such that $\widehat{\Psi}(e)$ is defined, then the operator names* op-names *and* lang *do not occur in e and an operator name in $\Delta(\Psi)$ occurs in e, if at all, only as the name of a constant of the form $(o :: \mathsf{type})(\,)$, $(o :: \mathsf{C})(\,)$, or $(o :: \mathsf{formula})(\,)$ that is in not within a quotation.*

9.4 Pseudointerpretations

Let $T_i = (L_i, \Gamma_i)$ be a normal theory of Chiron for $i = 1, 2$ and Ψ be a pseudotranslation from L_1 to L_2. Ψ *fixes* a language $L = (\mathcal{O}, \theta) \leq L_1$ if $\Psi(o) = o$ for all $o \in \mathcal{O} \setminus \{\mathsf{op\text{-}names}, \mathsf{lang}\}$. Ψ is *normal* if $\mathcal{O}_1$ is finite and Ψ fixes L_{ker}. Assume Ψ is normal with $\mathcal{O}_1 = \{o_1, \ldots, o_n\}$.

An *associate* of Ψ is a pair (T, Φ) where $T = (L, \Gamma)$ is a theory of Chiron, $L = (\mathcal{O}, \theta)$, and Φ is a (regular) translation from L_1 to L such that:

1. $\Phi(o) = \Psi(o)$ for all $o \in \mathcal{O}_1^- \setminus \Delta(\Psi)$.

2. $\mathcal{O}_2 \cap \{\Phi(o) \mid o \in \Delta(\Psi) \cup \{\mathsf{op\text{-}names}, \mathsf{lang}\}\} = \emptyset$.

3. $\mathcal{O} = \mathcal{O}_2 \cup \{\Phi(o) \mid o \in \Delta(\Psi) \cup \{\mathsf{op\text{-}names}, \mathsf{lang}\}\}$.

4. $\theta = \theta_2 \cup \{\langle \Phi(o), \theta_1(o) \rangle \mid o \in \Delta(\Psi) \cup \{\mathsf{op\text{-}names}, \mathsf{lang}\}\}$.

5. $\Gamma_{\mathrm{ty}} = \{\widehat{\Phi}((o :: \mathsf{type})(\,)) =_{\mathrm{ty}} \Psi(o) \mid o \in \Delta(\Psi) \text{ and } \theta(o) = \mathsf{type}\}$.

6. $\Gamma_{\mathrm{te}} = \{\widehat{\Phi}((o :: \mathsf{C})(\,)) \simeq \Psi(o) \mid o \in \Delta(\Psi) \text{ and } \theta(o) = \mathsf{term}\}$.

7. $\Gamma_{\text{fo}} = \{\widehat{\Phi}((o :: \mathsf{formula})(\,)) \equiv \Psi(o) \mid o \in \Delta(\Psi) \text{ and } \theta(o) = \mathsf{formula}\}$.
8. A_1 is $\widehat{\Phi}(\ell) = \{\ulcorner\Phi(o_1)\urcorner, \ldots, \ulcorner\Phi(o_n)\urcorner\}$.
9. A_2 is $\widehat{\Phi}(\mathsf{L}) =_{\text{ty}} \mathsf{type}(\mathsf{power}(\{\ulcorner\Phi(o_1)\urcorner, \ldots, \ulcorner\Phi(o_n)\urcorner\}))$
10. $\Gamma = \Gamma_2 \cup \Gamma^L_{\text{ker}} \cup \Gamma_{\text{ty}} \cup \Gamma_{\text{te}} \cup \Gamma_{\text{fo}} \cup \{A_1, A_2\}$.

Remark 9.4.1 If $T_i = (L_i, \Gamma_i)$ is a normal theory of Chiron for $i = 1, 2$ and Ψ is a normal pseudotranslation from L_1 to L_2, an associate of Ψ can be easily constructed after an appropriate set of "new" 0-ary operator names are added to L_2 to obtain L. Moreover, two associates (T, Φ) and (T', Φ') of Ψ are identical except that Φ and Φ' may map $\Delta(\Psi) \cup \{\mathsf{op\text{-}names}, \mathsf{lang}\}$ to different sets of new 0-ary operator names.

Let L be a language of Chiron and $T = (L, \Gamma)$ be a theory of Chiron. A formula A of L is *independent of L in T* if $(L', \Gamma) \models A$ for some language L' with $L \leq L'$ implies $(L', \Gamma) \models A$ for all languages L' with $L \leq L'$.

Lemma 9.4.2 *Let $T_i = (L_i, \Gamma_i)$ be a normal theory of Chiron for $i = 1, 2$; Ψ be a normal pseudotranslation from L_1 to L_2; and (T, Φ) where $T = (L, \Gamma)$ be an associate of Ψ.*

1. *$T_2 \leq T$.*
2. *T is normal.*
3. *For all formulas A of L_1 such that $\widehat{\Psi}(A)$ is defined,*

$$T \models \widehat{\Psi}(A) \equiv \widehat{\Phi}(A).$$

4. *For all formulas A of L_1 such that $\widehat{\Psi}(A)$ is defined and independent of L_2 in T_2,*

$$T_2 \models \widehat{\Psi}(A) \text{ implies } T \models \widehat{\Phi}(A).$$

Proof

Part 1 Obvious.

Part 2 Follows immediately from part 1 and the fact that T_2 is normal.

Part 3 By the definition of $\widehat{\Psi}$ and the construction of Γ.

Part 4 Let A be a formula of L_1 such that $\widehat{\Psi}(A)$ is defined and independent of L_2 in T_2 and $T_2 \models \widehat{\Psi}(A)$. We must show $T \models \widehat{\Phi}(A)$. Let $T_2' = (L, \Gamma_2)$. $T_2' \models \widehat{\Psi}(A)$ since $\widehat{\Psi}(A)$ is independent of L_2 in T_2. By part 1, $T_2 \leq T_2' \leq T$, and so, by Lemma 4.6.1, $T \models \widehat{\Psi}(A)$. By part 3, $T \models \widehat{\Psi}(A) \equiv \widehat{\Phi}(A)$. Therefore, $T \models \widehat{\Phi}(A)$. □

A pseudotranslation Ψ from L_1 to L_2 is a *(semantic) pseudointerpretation of T_1 in T_2* if:

1. Ψ is normal.
2. For each $A \in \Gamma_1 \setminus \Gamma_{\mathrm{ker}}^{L_1}$, A is eval-free, $\widehat{\Psi}(A)$ is defined and independent of L_2 in T_2, and $T_2 \models \widehat{\Psi}(A)$.

Theorem 9.4.3 *Let $T_i = (L_i, \Gamma_i)$ be a normal theory of Chiron for $i = 1, 2$; Ψ be a pseudointerpretation of T_1 in T_2; and (T, Φ) be an associate of Ψ.*

1. *Φ is an interpretation of T_1 in T.*
2. *For all formulas A of L_1 such that $\widehat{\Psi}(A)$ is defined and independent of L_2 in T_2,*

$$T_1 \models A \text{ implies } T_2 \models \widehat{\Psi}(A).$$

Proof

Part 1 First, we must show that T is a normal theory. T is normal since $\Gamma_{\mathrm{ker}}^{L} \leq \Gamma$ by the construction of T.

Second, we must show that Φ is normal for T. Since Ψ is a pseudointerpretation, Ψ is normal and hence $\mathcal{O}_1$ is finite and Ψ fixes L_{ker}. The latter implies Φ fixes L_{ker}. The last conditions required for Φ to be normal for T are met since the axioms of T include the formulas A_1 and A_2 from the definition of an associate of a pseudotranslation.

By Theorem 9.2.8, it remains for us to show that $T \models \widehat{\Phi}(\overline{A})$ for each $A \in \Gamma_1$. Let $A \in \Gamma_1 \setminus \Gamma_{\mathrm{ker}}^{L_1}$. Then $T \models \widehat{\Phi}(\overline{A})$ by Lemma 9.2.3 and the fact that Φ is normal for T. Now let $A \in \Gamma_1 \setminus \Gamma_{\mathrm{ker}}^{L_1}$. Then, since Ψ is a pseudointerpretation, (a) A is eval-free, (b) $\widehat{\Psi}(A)$ is defined and independent of L_2 in T_2, and (c) $T_2 \models \widehat{\Psi}(A)$. (a) implies (d) $A = \overline{A}$. By part 4 of Lemma 9.4.2, (b) and (c) imply (e) $T \models \widehat{\Phi}(A)$. And (d) and (e) implies $T \models \widehat{\Phi}(\overline{A})$.

Part 2 Let A be a formula of L_1 such that (a) $\widehat{\Psi}(A)$ is defined, (b) $\widehat{\Psi}(A)$ is independent of L_2 in T_2, and (c) $T_1 \models A$. We must show $T_2 \models \widehat{\Psi}(A)$. (c) implies (d) $T \models \widehat{\Phi}(A)$ since Φ is an interpretation. By part 3 of Lemma 9.4.2, (d) implies $T \models \widehat{\Psi}(A)$. (a) and (d) imply (e) $T_2' \models \widehat{\Psi}(A)$ where $T_2' = (L, \Gamma_2)$. Finally, (b) and (e) imply $T_2 \models \widehat{\Psi}(A)$. □

Remark 9.4.4 By virtue of Theorem 9.4.3 a pseudointerpretation can be viewed as a regular interpretation in a more convenient form.

10 Conclusion

In this paper we have presented the syntax and semantics of a set theory named Chiron that is intended to be a practical, general-purpose logic for mechanizing mathematics. Several operator definitions and simple examples are given that illustrate Chiron's practical expressivity, especially its facility for reasoning about the syntax of expressions. A proof system for Chiron is presented that is intended to be a system test of Chiron's definition and a reference system for other, more practical, proof systems for Chiron. The proof system is proved to be sound and also complete in a restricted sense. And a notion of an interpretation of one theory in another is defined.

This paper is a first step in a long-range research program to design, analyze, and implement Chiron. In the future we plan to:

1. Design a practical proof system for Chiron.
2. Implement Chiron and its proof system.
3. Develop a series of applications to demonstrate Chiron's reach and level of effectiveness. As a first step, we have shown how biform theories can be formalized in Chiron [7]. A *biform theory* is a theory in which both formulas and algorithms can serve as axioms [7, 10].

Acknowledgments

The author is grateful to Marc Bender and Jacques Carette for many valuable discussions on the design and use of Chiron. Over the course of these discussions, Dr. Carette convinced the author that Chiron needs to include a powerful facility for reasoning about the syntax of expressions. The author is also grateful to Russell O'Connor for reading over the paper carefully and pointing out several mistakes.

A Appendix: Alternate Semantics

This appendix presents two alternate semantics for Chiron based on S. Kripke's framework for defining semantics with *truth-value gaps* which is described in his famous paper *Outline of a Theory of Truth* [15]. Both semantics use *value gaps* for types and terms as well as for formulas. The first defines the value gaps according to weak Kleene logic [14], while the second defines the values gaps according to a valuation scheme based on B. van Fraassen's notion of a *supervaluation* [28] that Kripke describes in [15, p. 711].

A.1 Valuations

The notion of a valuation for a structure was defined in subsection 4.4. Fix a structure S for L. Let $\mathsf{val}(S)$ be the collection of valuations for S. Given $U, V \in \mathsf{val}(S)$, U is a *subvaluation* of V, written $U \sqsubseteq V$, if, for all $e \in \mathcal{E}_L$ and $\varphi \in \mathsf{assign}(S)$, $U_\varphi(e)$ is defined implies $U_\varphi(e) = V_\varphi(e)$. A *valuation functional* for S is a mapping from $\mathsf{val}(S)$ into $\mathsf{val}(S)$. Let Ψ be a valuation functional for S. A *fixed point* of Ψ is a $V \in \mathsf{val}(S)$ such that $\Psi(V) = V$. Ψ is *monotone* if $U \sqsubseteq V$ implies $\Psi(U) \sqsubseteq \Psi(V)$ for all $U, V \in \mathsf{val}(S)$.

Theorem A.1.1 *Let Ψ be a monotone valuation functional for S. Then Ψ has a fixed point.*

Proof The construction of a fixed point of Ψ is similar to the construction of the fixed point Kripke gives in [15, pp. 703–705]. □

Ψ_1^S is the valuation functional for S defined by the following rules where $V \in \mathsf{val}(S)$ and $V' = \Psi_1^S(V)$. There is a rule for the category of improper expressions and a rule for each of the 13 categories of proper expressions. Note that only part (a) of the rule 14 (the rule for evaluation) makes use of V. Ψ_1^S defines value gaps according to the weak Kleene logic valuation scheme in which a proper expression is denoting only if all of its proper subexpressions are also denoting.

1. Let $e \in \mathcal{E}_L$ be improper. Then $V'_\varphi(e)$ is undefined.
2. Let $O = (\mathsf{op}, s, k_1, \ldots, k_n, k_{n+1})$ be proper.
    a. Let $V'_\varphi(k_i)$ be defined for all i with $1 \leq i \leq n+1$ and $\mathbf{type}_L[k_i]$. Then $I(o)$ is an n-ary operation in D_o from $D_1 \times \cdots \times D_n$ into

D_{n+1}. $V'_\varphi(O)$ is the n-ary operation in D_o from $D_1 \times \cdots \times D_n$ into D_{n+1} defined as follows. Let $(d_1, \ldots, d_n) \in D_1 \times \cdots \times D_n$ and $d = I(o)(d_1, \ldots, d_n)$. If d_i is in $V'_\varphi(k_i)$ or $d_i = \bot$ for all i such that $1 \leq i \leq n$ and $\mathbf{type}_L[k_i]$ and d is in $V'_\varphi(k_{n+1})$ or $d = \bot$ when $\mathbf{type}_L[k_{n+1}]$, then $V'_\varphi(O)(d_1, \ldots, d_n) = d$. Otherwise, $V'_\varphi(O)(d_1, \ldots, d_n)$ is D_c if $k_{n+1} = \mathsf{type}$, $\bot$ if $\mathbf{type}_L[k_{n+1}]$, and F if $k_{n+1} = \mathsf{formula}$.

b. Let $V'_\varphi(k_i)$ be undefined for some i such that $1 \leq i \leq n+1$ and $\mathbf{type}_L[k_i]$. Then $V'_\varphi(e)$ is undefined.

3. Let $e = (\mathsf{op\text{-}app}, O, e_1, \ldots, e_n)$ be proper.

a. Let $V'_\varphi(O), V'_\varphi(e_1), \ldots, V'_\varphi(e_n)$ be defined. Then

$$V'_\varphi(e) = V'_\varphi(O)(V'_\varphi(e_1), \ldots, V'_\varphi(e_n)).$$

b. Let one of $V'(O), V'_\varphi(e_1), \ldots, V'_\varphi(e_n)$ be undefined. Then $V'_\varphi(e)$ is undefined.

4. Let $a = (\mathsf{var}, x, \alpha)$ be proper.

a. Let $V'_\varphi(\alpha)$ be defined. If $\varphi(x)$ is in $V'_\varphi(\alpha)$, then $V'_\varphi(a) = \varphi(x)$. Otherwise $V'_\varphi(a) = \bot$.

b. Let $V'_\varphi(\alpha)$ be undefined. Then $V'_\varphi(a)$ is undefined.

5. Let $\beta = (\mathsf{type\text{-}app}, \alpha, a)$ be proper.

a. Let $V'_\varphi(\alpha)$ and $V'_\varphi(a)$ be defined. If $V'_\varphi(a) \neq \bot$, then $V'_\varphi(\beta) = V'_\varphi(\alpha)[V'_\varphi(a)]$. Otherwise $V'_\varphi(\beta) = D_\mathrm{c}$.

b. Let $V'_\varphi(\alpha)$ or $V'_\varphi(a)$ be undefined. Then $V'_\varphi(\beta)$ is undefined.

6. Let $\gamma = (\mathsf{dep\text{-}fun\text{-}type}, (\mathsf{var}, x, \alpha), \beta)$ be proper.

a. Let $V'_\varphi(\alpha)$ be defined, and let $V'_{\varphi[x \mapsto d]}(\beta)$ be defined for all sets d in $V'_\varphi(\alpha)$. Then $V'_\varphi(\gamma)$ is the superclass of all g in D_f such that, for all d in D_v, if $g(d)$ is defined, then d is in $V'_\varphi(\alpha)$ and $g(d)$ is in $V'_{\varphi[x \mapsto d]}(\beta)$.

b. Let $V'_\varphi(\alpha)$ be undefined, or let $V'_{\varphi[x \mapsto d]}(\beta)$ be undefined for some set d in $V'_\varphi(\alpha)$. Then $V'_\varphi(\gamma)$ is undefined.

7. Let $b = (\mathsf{fun\text{-}app}, f, a)$ be proper.

a. Let $V'_\varphi(f)$ and $V'_\varphi(a)$ be defined. If $V'_\varphi(f) \neq \bot$ and $V'_\varphi(a) \neq \bot$, then $V'_\varphi(b) = V'_\varphi(f)(V'_\varphi(a))$. Otherwise $V'_\varphi(b) = \bot$.

b. Let $V'_\varphi(f)$ or $V'_\varphi(a)$ be undefined. Then $V'_\varphi(b)$ is undefined.

8. Let $f = (\mathsf{fun\text{-}abs}, (\mathsf{var}, x, \alpha), b)$ be proper.

   a. Let $V'_\varphi(\alpha)$ be defined, and let $V'_{\varphi[x\mapsto d]}(b)$ be defined for all sets d in $V'_\varphi(\alpha)$. If

   $$g \;=\; \{\langle d, d'\rangle \mid d \text{ is a set in } V'_\varphi(\alpha) \text{ and } d' = V'_{\varphi[x\mapsto d]}(b) \text{ is a set}\}$$

   is in D_f, then $V'_\varphi(f) = g$. Otherwise $V'_\varphi(f) = \bot$.

   b. Let $V'_\varphi(\alpha)$ be undefined, or let $V'_{\varphi[x\mapsto d]}(b)$ be undefined for some set d in $V'_\varphi(\alpha)$. Then $V'_\varphi(f)$ is undefined.

9. Let $a = (\mathsf{if}, A, b, c)$ be proper.

   a. Let $V'_\varphi(A), V'_\varphi(b), V'_\varphi(c)$ be defined. If $V'_\varphi(A) = \textsc{t}$, then $V'_\varphi(a) = V'_\varphi(b)$. Otherwise $V'_\varphi(a) = V'_\varphi(c)$.

   b. Let one of $V'_\varphi(A), V'_\varphi(b), V'_\varphi(c)$ be undefined. Then $V'_\varphi(a)$ is undefined.

10. Let $A = (\mathsf{exists}, (\mathsf{var}, x, \alpha), B)$ be proper.

    a. Let $V'_\varphi(\alpha)$ be defined, and let $V'_{\varphi[x\mapsto d]}(B)$ be defined for all d in $V'_\varphi(\alpha)$. If there is some d in $V'_\varphi(\alpha)$ such that $V'_{\varphi[x\mapsto d]}(B) = \textsc{t}$, then $V'_\varphi(A) = \textsc{t}$. Otherwise, $V'_\varphi(A) = \textsc{f}$.

    b. Let $V'_\varphi(\alpha)$ be undefined, or let $V'_{\varphi[x\mapsto d]}(B)$ be undefined for some d in $V'_\varphi(\alpha)$. Then $V'_\varphi(A)$ is undefined.

11. Let $a = (\mathsf{def\text{-}des}, (\mathsf{var}, x, \alpha), B)$ be proper.

    a. Let $V'_\varphi(\alpha)$ be defined, and let $V'_{\varphi[x\mapsto d]}(B)$ be defined for all d in $V'_\varphi(\alpha)$. If there is a unique d in $V'_\varphi(\alpha)$ such that $V'_{\varphi[x\mapsto d]}(B) = \textsc{t}$, then $V'_\varphi(a) = d$. Otherwise, $V'_\varphi(a) = \bot$.

    b. Let $V'_\varphi(\alpha)$ be undefined, or let $V'_{\varphi[x\mapsto d]}(B)$ be undefined for some d in $V'_\varphi(\alpha)$. Then $V'_\varphi(a)$ is undefined.

12. Let $a = (\mathsf{indef\text{-}des}, (\mathsf{var}, x, \alpha), B)$ be proper.

a. Let $V'_\varphi(\alpha)$ be defined, and let $V'_{\varphi[x\mapsto d]}(B)$ be defined for all d in $V'_\varphi(\alpha)$. If there is some d in $V'_\varphi(\alpha)$ such that $V'_{\varphi[x\mapsto d]}(B) = \mathrm{T}$, then $V'_\varphi(a) = \xi(\Sigma)$ where Σ is the superclass of all d in $V'_\varphi(\alpha)$ such that $V'_{\varphi[x\mapsto d]}(B) = \mathrm{T}$. Otherwise, $V'_\varphi(a) = \bot$.

b. Let $V'_\varphi(\alpha)$ be undefined, or let $V'_{\varphi[x\mapsto d]}(B)$ be undefined for some d in $V'_\varphi(\alpha)$. Then $V'_\varphi(a)$ is undefined.

13. Let $a = (\mathsf{quote}, e)$ be proper. Then $V'_\varphi(a) = H(e)$.

14. Let $b = (\mathsf{eval}, a, k)$ be proper.

    a. Let $V'_\varphi(a)$ be defined and $V'_\varphi(k)$ be defined if $\mathbf{type}_L[k]$.

        i. Let $V'_\varphi(a)$ be in D_{ty} and $k = \mathsf{type}$, $V'_\varphi(a)$ be in D_{te} and $\mathbf{type}_L[k]$, or $V'_\varphi(a)$ be in D_{fo} and $k = \mathsf{formula}$.

            A. Let $V_\varphi(H^{-1}(V'_\varphi(a)))$ be defined. If $k \in \{\mathsf{type}, \mathsf{formula}\}$ or $\mathbf{type}_L[k]$ and $V_\varphi(H^{-1}(V'_\varphi(a)))$ is in $V'_\varphi(k)$, then $V'_\varphi(b) = V_\varphi(H^{-1}(V'_\varphi(a)))$. Otherwise $V'_\varphi(b)$ is $\bot$.

            B. Let $V_\varphi(H^{-1}(V'_\varphi(a)))$ be undefined. Then $V'_\varphi(b)$ is undefined.

        ii. Let $V'_\varphi(a)$ not be in D_{ty} or $k \neq \mathsf{type}$, $V'_\varphi(a)$ not be in D_{te} or not $\mathbf{type}_L[k]$, and $V'_\varphi(a)$ not be in D_{fo} or $k \neq \mathsf{formula}$. Then $V'_\varphi(b)$ is D_{c} if $k = \mathsf{type}$, $\bot$ if $\mathbf{type}_L[k]$, and F if $k = \mathsf{formula}$.

    b. Let $V'_\varphi(a)$ be undefined or $V'_\varphi(k)$ be undefined if $\mathbf{type}_L[k]$. Then $V'_\varphi(b)$ is undefined.

Lemma A.1.2 *Ψ_1^S is monotone.*

Proof Let $U, V \in \mathsf{val}(S)$ such that $U \sqsubseteq V$. Assume U'_φ and V'_φ mean $(\Psi_1^S(U))_\varphi$ and $(\Psi_1^S(V))_\varphi$, respectively. We must show that, for all $e \in \mathcal{E}_L$ and $\varphi \in \mathsf{assign}(S)$, if $U'_\varphi(e)$ is defined, then $U'_\varphi(e) = V'_\varphi(e)$. Our proof will be by induction on the number of symbols in e.

There are three cases:

1. *e is improper.* Then $U'_\varphi(e)$ is undefined by the definition of Ψ_1^S.

2. *$e = (\mathsf{eval}, a, k)$ is proper.* If either $U'_\varphi(a)$ or $U'_\varphi(k)$ is undefined, then $U'_\varphi(e)$ is undefined. So assume $U'_\varphi(a)$ and $U'_\varphi(k)$ are defined. By the induction hypothesis, $U'_\varphi(a) = V'_\varphi(a)$ and $U'_\varphi(k) = V'_\varphi(k)$. Assume $U'_\varphi(e)$ is defined. By the definition of Ψ_1^S, there are two subcases:

a. *For some* $e_1, e_2 \in \mathcal{E}_L$, $U'_\varphi(e) = U_\varphi(e_1)$ *and* $V'_\varphi(e) = V_\varphi(e_2)$. Since $U'_\varphi(a) = V'_\varphi(a)$, $e_1 = e_2$, and since $U \sqsubseteq V$, $U_\varphi(e_1) = V_\varphi(e_2)$. Hence, $U'_\varphi(e) = V'_\varphi(e)$.

b. $U'_\varphi(e)$ *and* $V'_\varphi(e)$ *both equal* D_c *if* $k = \mathsf{type}$, $\bot$ *if* $\mathbf{type}_L[k]$, and F *if* $k = \mathsf{formula}$. Hence, $U'_\varphi(e) = V'_\varphi(e)$.

3. e *is proper but not an evaluation.* Assume $U'_\varphi(e)$ is defined. Then $U'_\varphi(e')$ is defined for each subexpression e' of e. By the induction hypothesis, $U'_\varphi(e') = V'_\varphi(e')$ for each such subexpression e' of e. Hence, $U'_\varphi(e) = V'_\varphi(e)$.

□

Corollary A.1.3 Ψ_1^S *has a fixed point.*

Proof By Lemma A.1.2, Ψ_1^S is monotone. Therefore, by Theorem A.1.1, Ψ_1^S has a fixed point. □

Ψ_2^S is the valuation functional for S defined by the following three rules where $V \in \mathsf{val}(S)$ and $V' = \Psi_2^S(V)$. Ψ_2^S defines value gaps according to the supervaluation scheme.

1. Let $e \in \mathcal{E}_L$ be improper. Then $V'_\varphi(e)$ is undefined.

2. Let $e \in \mathcal{E}_L$ be proper but not an evaluation. If there is a value d such that, for all total valuations V^* with $V \sqsubseteq V^*$, $(\Psi_1^S(V^*))_\varphi(e) = d$, then $V'_\varphi(e) = d$. Otherwise $V'_\varphi(e)$ is undefined.

3. Let $e \in \mathcal{E}_L$ be proper with $e = (\mathsf{eval}, a, k)$. This rule is exactly the same as the Ψ_1^S rule for evaluations.

Lemma A.1.4 Ψ_2^S *is monotone.*

Proof The proof is exactly the same as the proof of Lemma A.1.2 except for the argument for the third case:

3. e *is proper but not an evaluation.* Assume $U'_\varphi(e)$ is defined. Then there is a value d such that, for all total valuations U^* with $U \sqsubseteq U^*$, $U^*_\varphi(e) = d$. Since $U \sqsubseteq V$, it follows that, for all total valuations V^* with $V \sqsubseteq V^*$, $V^*_\varphi(e) = d$. Hence, $U'_\varphi(e) = V'_\varphi(e)$.

□

Corollary A.1.5 Ψ_2^S *has a fixed point.*

Proof By Lemma A.1.4, Ψ_2^S is monotone. Therefore, by Theorem A.1.1, Ψ_2^S has a fixed point. □

A.2 Models

The valuation functionals Ψ_1^S and Ψ_2^S define two semantics, which we will refer to as the *weak Kleene semantics* and the *supervaluation semantics*, respectively. Clearly, the supervaluation semantics allows more expressions to be denoting than the weak Kleene semantics.

A *weak Kleene model* for L is a model $M = (S, V)$ where S is a structure for L and V is a valuation for S that is a fixed point of Ψ_1^S. A *supervaluation model* for L is a model $M = (S, V)$ where S is a structure for L and V is a valuation for S that is a fixed point of Ψ_2^S.

Theorem A.2.1 *Let L be a language of Chiron. For each structure S for L there exists a weak Kleene model and a supervaluation model for L.*

Proof Let L be a language of Chiron and S be a structure for L. By Corollary A.1.3, Ψ_1^S has a fixed point V_1. Similarly, by Corollary A.1.5, Ψ_2^S has a fixed point V_2. Therefore, $M = (S, V_1)$ is a weak Kleene model for L, and $M = (S, V_2)$ is a supervaluation model for L. □

The weak Kleene semantics defined by Ψ_1^S is "strict" in the sense that, if any proper subexpression e of a proper expression e' is nondenoting, then e' itself is nondenoting. The supervaluation semantics defined by Ψ_2^S is not strict in this sense. For example, the value of an application of the operator

$$(\mathsf{op}, \mathsf{or}, \mathsf{formula}, \mathsf{forumla}, \mathsf{formula})$$

to a pair of formulas (A, B) is T if the value of A is T and B is nondenoting or vice versa.

A.3 Discussion

There are various Kripke-style value-gap semantics for Chiron; the weak Kleene and supervaluation semantics are just two examples. The weak Kleene semantics is a conservative example: every expression that could be nondenoting is nondenoting. On the other hand, the supervaluation semantics is much more liberal: many expressions that are nondenoting in the weak Kleene semantics are denoting in the supervaluation semantics.

It is not possible to define a denoting formula checker in any Kripke-style value-gap semantics. If the operator $O = (o :: \mathsf{formula}, \mathsf{formula})$ were a denoting formula checker, then $O(e)$ would be true whenever e is denoting and false whenever e is nondenoting. However, such an operator breaks the monotonicity lemmas proved above because, if $U'_\varphi(e)$ is undefined but

$V'_\varphi(e)$ is defined, then $U'_\varphi(O(e)) = \mathrm{F} \neq \mathrm{T} = V'_\varphi(O(e))$. Similarly, it is not possible to define denoting type and term checkers in a Kripke-style value-gap semantics.

The lack of available checkers for denoting types, terms, and formulas makes reasoning in Kripke-style value-gap semantics very difficult. For example, consider the formalization of the law of excluded middle given in subsection 7.1:

$$\forall\, e : \mathsf{E}_{\mathrm{fo}} \,.\, [\![e]\!] \vee \neg[\![e]\!].$$

This formula is nondenoting in the weak Kleene semantics because, if e represents a nondenoting formula, then $[\![e]\!] \vee \neg[\![e]\!]$ is nondenoting. Since this formula is nondenoting, we cannot use it as a basis for proof by cases.

This formula is true in the supervaluations semantics because, if e represents a nondenoting formula, then $[\![e]\!] \vee \neg[\![e]\!]$ is true because $[\![e]\!] \vee \neg[\![e]\!]$ is true no matter what value is assigned to $[\![e]\!]$. Even though this formula is true, we cannot use it as a basis for proof by cases because, if e is the liar paradox, we can derive a contradiction from either $[\![e]\!]$ or $\neg[\![e]\!]$.

We expect that reasoning in the official semantics for Chiron will be much easier than in any Kripke-style value-gap semantics for Chiron.

B Appendix: An Expanded Definition of a Proper Expression

We give in this appendix an expanded definition of a proper expression with 25 proper expression categories. There are modified categories for operator applications, conditional terms, and quotations and new categories for constants (applications of 0-ary operators), four variable binders given above as notational definitions, finite sets of sets, finite lists of sets, and dependent ordered pairs

The following symbols are added to the the set $\mathcal{K}$ of key words: con, uni-exists, forall, set-cons, list-cons, class-abs, dep-type-prod, left-type, right-type, dep-ord-pair, dep-head, and dep-tail. The following formation rules define the expanded set of proper expression categories:

P-Expr-1 (Operator)

$$\frac{o \in \mathcal{O}, \mathbf{kind}_L[k_1], \ldots, \mathbf{kind}_L[k_{n+1}]}{\mathbf{operator}_L[(\mathsf{op}, o, k_1, \ldots, k_{n+1})]}$$

where $n \geq 0$; $\theta(o) = s_1, \ldots, s_{n+1}$; and $k_i = s_i = \mathsf{type}$, $\mathbf{type}_L[k_i]$ and $s_i = \mathsf{term}$, or $k_i = s_i = \mathsf{formula}$ for all i with $1 \leq i \leq n+1$.

P-Expr-2 (Operator application)

$$\frac{\mathbf{operator}_L[(\mathsf{op}, o, k_1, \ldots, k_{n+1})], \mathbf{expr}_L[e_1], \ldots, \mathbf{expr}_L[e_n]}{\mathbf{p\text{-}expr}_L[(\mathsf{op\text{-}app}, (\mathsf{op}, o, k_1, \ldots, k_{n+1}), e_1, \ldots, e_n) : k_{n+1}]}$$

where $n \geq 1$ and ($k_i = \mathsf{type}$ and $\mathbf{type}_L[e_i]$), ($\mathbf{type}_L[k_i]$ and $\mathbf{term}_L[e_i]$), or ($k_i = \mathsf{formula}$ and $\mathbf{formula}_L[e_i]$) for all i with $1 \leq i \leq n$.

P-Expr-3 (Constant)

$$\frac{\mathbf{operator}_L[(\mathsf{op}, o, k)]}{\mathbf{p\text{-}expr}_L[(\mathsf{con}, o, k) : k]}$$

P-Expr-4 (Variable)

$$\frac{x \in \mathcal{S}, \mathbf{type}_L[\alpha]}{\mathbf{term}_L[(\mathsf{var}, x, \alpha) : \alpha]}$$

P-Expr-5 (Type application)

$$\frac{\mathbf{type}_L[\alpha], \mathbf{term}_L[a]}{\mathbf{type}_L[(\mathsf{type\text{-}app}, \alpha, a)]}$$

P-Expr-6 (Dependent function type)

$$\frac{\mathbf{term}_L[(\mathsf{var}, x, \alpha)], \mathbf{type}_L[\beta]}{\mathbf{type}_L[(\mathsf{dep\text{-}fun\text{-}type}, (\mathsf{var}, x, \alpha), \beta)]}$$

P-Expr-7 (Function application)

$$\frac{\mathbf{term}_L[f : \alpha], \mathbf{term}_L[a]}{\mathbf{term}_L[(\mathsf{fun\text{-}app}, f, a) : (\mathsf{type\text{-}app}, \alpha, a)]}$$

P-Expr-8 (Function abstraction)

$$\frac{\mathbf{term}_L[(\mathsf{var}, x, \alpha)], \mathbf{term}_L[b : \beta]}{\mathbf{term}_L[(\mathsf{fun\text{-}abs}, (\mathsf{var}, x, \alpha), b) : (\mathsf{dep\text{-}fun\text{-}type}, (\mathsf{var}, x, \alpha), \beta)]}$$

P-Expr-9 (Conditional term)

$$\frac{\mathbf{formula}_L[A], \mathbf{term}_L[b : \beta], \mathbf{term}_L[c : \gamma]}{\mathbf{term}_L[(\mathsf{if}, A, b, c) : \beta \cup \gamma]}$$

P-Expr-10 (Existential quantification)

$$\frac{\mathbf{term}_L[(\mathsf{var}, x, \alpha)], \mathbf{formula}_L[B]}{\mathbf{formula}_L[(\mathsf{exists}, (\mathsf{var}, x, \alpha), B)]}$$

P-Expr-11 (Unique existential quantification)

$$\frac{\mathbf{term}_L[(\mathsf{var}, x, \alpha)], \mathbf{formula}_L[B]}{\mathbf{formula}_L[(\mathsf{uni\text{-}exists}, (\mathsf{var}, x, \alpha), B)]}$$

P-Expr-12 (Universal quantification)

$$\frac{\mathbf{term}_L[(\mathsf{var}, x, \alpha)], \mathbf{formula}_L[B]}{\mathbf{formula}_L[(\mathsf{forall}, (\mathsf{var}, x, \alpha), B)]}$$

P-Expr-13 (Definite description)

$$\frac{\mathbf{term}_L[(\mathsf{var}, x, \alpha)], \mathbf{formula}_L[B]}{\mathbf{term}_L[(\mathsf{def\text{-}des}, (\mathsf{var}, x, \alpha), B) : \alpha]}$$

P-Expr-14 (Indefinite description)

$$\frac{\mathbf{term}_L[(\mathsf{var}, x, \alpha)], \mathbf{formula}_L[B]}{\mathbf{term}_L[(\mathsf{indef\text{-}des}, (\mathsf{var}, x, \alpha), B) : \alpha]}$$

P-Expr-15 (Set construction)

$$\frac{\mathbf{term}_L[a_1 : \alpha_1], \ldots, \mathbf{term}_L[a_1 : \alpha_1]}{\mathbf{term}_L[(\mathsf{set\text{-}cons}, a_1, \ldots, a_n) : \beta]}$$

where $n \geq 0$ and $\beta = \begin{cases} \mathsf{C} & \text{if } n = 0 \\ \alpha_1 \cup \cdots \cup \alpha_n & \text{otherwise.} \end{cases}$

P-Expr-16 (List construction)

$$\frac{\mathbf{term}_L[a_1 : \alpha_1], \ldots, \mathbf{term}_L[a_1 : \alpha_1]}{\mathbf{term}_L[(\mathsf{list\text{-}cons}, a_1, \ldots, a_n) : \mathsf{list\text{-}type}(\beta)]}$$

where $n \geq 0$ and $\beta = \begin{cases} \mathsf{C} & \text{if } n = 0 \\ \alpha_1 \cup \cdots \cup \alpha_n & \text{otherwise.} \end{cases}$

P-Expr-17 (Class abstraction)

$$\frac{\mathbf{term}_L[(\mathsf{var}, x, \alpha)], \mathbf{formula}_L[B]}{\mathbf{term}_L[(\mathsf{class\text{-}abs}, (\mathsf{var}, x, \alpha), B) : \mathsf{power\text{-}type}(\alpha)]}$$

P-Expr-18 (Left type)

$$\frac{\mathbf{type}_L[\alpha]}{\mathbf{type}_L[(\mathsf{left\text{-}type}, \alpha)]}$$

P-Expr-19 (Right type)

$$\frac{\mathbf{type}_L[\alpha], \mathbf{term}_L[a]}{\mathbf{type}_L[(\mathsf{right\text{-}type}, \alpha, a)]}$$

P-Expr-20 (Dependent type product)

$$\frac{\mathbf{term}_L[(\mathsf{var}, x, \alpha)], \mathbf{type}_L[\beta]}{\mathbf{type}_L[(\mathsf{dep\text{-}type\text{-}prod}, (\mathsf{var}, x, \alpha), \beta)]}$$

P-Expr-21 (Dependent ordered pair)

$$\frac{\mathbf{term}_L[a : \alpha], \mathbf{term}_L[b : \beta]}{\mathbf{term}_L[(\mathsf{dep\text{-}ord\text{-}pair}, a, b) : (\mathsf{dep\text{-}type\text{-}prod}, (\mathsf{var}, x, \alpha), \beta)]}$$

P-Expr-22 (Dependent head)

$$\frac{\mathbf{term}_L[a : \alpha]}{\mathbf{term}_L[(\mathsf{dep\text{-}head}, a) : (\mathsf{left\text{-}type}, \alpha)]}$$

P-Expr-23 (Dependent tail)

$$\frac{\mathbf{term}_L[a : \alpha]}{\mathbf{term}_L[(\mathsf{dep\text{-}tail}, a) : (\mathsf{right\text{-}type}, \alpha, (\mathsf{dep\text{-}head}, a))]}$$

P-Expr-24 (Quotation)

$$\frac{\mathbf{expr}_L[e]}{\mathbf{term}_L[(\mathsf{quote}, e) : \alpha]}$$

where α is:

1. E_{sy} if $e \in \mathcal{S}$.
2. E_{on} if $e \in \mathcal{O}$.
3. E_{op} if e is an operator.
4. E_{ty} if e is a type.
5. $\mathsf{E}_{\mathrm{te}}^{\ulcorner\beta\urcorner}$ if e is a term of type β.
6. E_{fo} if e is a formula.

7. E if none of the above.

P-Expr-25 (Evaluation)

$$\frac{\mathbf{term}_L[a], \mathbf{kind}_L[k]}{\mathbf{p\text{-}expr}_L[(\mathsf{eval}, a, k) : k]}$$

Table 6 defines the compact notation for each of the 25 proper expression categories. Note: The definitions for the compact notations $\{a_1, \ldots, a_n\}$, $[a_1, \ldots, a_n]$, $\langle a, b \rangle$, $\mathsf{hd}(a)$, and $\mathsf{tl}(a)$ supersede the definitions for these notations given in subsection 5.1.

The semantics for the new definition of a proper expression is the same as before except for the definition of the valuation function, which is left to the reader as an exercise.

References

[1] A. Bawden. Quasiquotation in Lisp. In O. Danvy, editor, *Proceedings of the 1999 ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation*, pages 4–12, 1999. Technical report BRICS-NS-99-1, University of Aarhus, 1999.

[2] N. G. de Bruijn. Lambda-calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Indagationes Mathematicae*, 34:381–392, 1972.

[3] H. B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, 1972.

[4] W. M. Farmer. Theory interpretation in simple type theory. In J. Heering et al., editor, *Higher-Order Algebra, Logic, and Term Rewriting*, volume 816 of *Lecture Notes in Computer Science*, pages 96–123. Springer-Verlag, 1994.

[5] W. M. Farmer. STMM: A Set Theory for Mechanized Mathematics. *Journal of Automated Reasoning*, 26:269–289, 2001.

[6] W. M. Farmer. Formalizing undefinedness arising in calculus. In D. Basin and M. Rusinowitch, editors, *Automated Reasoning—IJCAR 2004*, volume 3097 of *Lecture Notes in Computer Science*, pages 475–489. Springer-Verlag, 2004.

Compact Notation	**Official Notation**
$(o :: k_1, \ldots, k_{n+1})$	$(\mathsf{op}, o, k_1, \ldots, k_{n+1})$
$(o :: k_1, \ldots, k_{n+1})(e_1, \ldots, e_n)$	$(\mathsf{op\text{-}app}, (\mathsf{op}, o, k_1, \ldots, k_{n+1}), e_1, \ldots, e_n)$
$[o :: k]$	(con, o, k)
$(x : \alpha)$	$(\mathsf{var}, x, \alpha)$
$\alpha(a)$	$(\mathsf{type\text{-}app}, \alpha, a)$
$(\Lambda\, x : \alpha\ .\ \beta)$	$(\mathsf{dep\text{-}fun\text{-}type}, (\mathsf{var}, x, \alpha), \beta)$
$f(a)$	$(\mathsf{fun\text{-}app}, f, a)$
$(\lambda\, x : \alpha\ .\ b)$	$(\mathsf{fun\text{-}abs}, (\mathsf{var}, x, \alpha), b)$
$\mathsf{if}(A, b, c)$	(if, A, b, c)
$(\exists\, x : \alpha\ .\ B)$	$(\mathsf{exists}, (\mathsf{var}, x, \alpha), B)$
$(\exists !\, x : \alpha\ .\ B)$	$(\mathsf{uni\text{-}exists}, (\mathsf{var}, x, \alpha), B)$
$(\forall\, x : \alpha\ .\ B)$	$(\mathsf{forall}, (\mathsf{var}, x, \alpha), B)$
$(\iota\, x : \alpha\ .\ B)$	$(\mathsf{def\text{-}des}, (\mathsf{var}, x, \alpha), B)$
$(\epsilon\, x : \alpha\ .\ B)$	$(\mathsf{indef\text{-}des}, (\mathsf{var}, x, \alpha), B)$
$\{a_1, \ldots, a_n\}$	$(\mathsf{set\text{-}cons}, a_1, \ldots, a_n)$
$[a_1, \ldots, a_n]$	$(\mathsf{list\text{-}cons}, a_1, \ldots, a_n)$
$(\mathrm{C}\, x : \alpha\ .\ B)$	$(\mathsf{class\text{-}abs}, (\mathsf{var}, x, \alpha), B)$
$\mathsf{left}(\alpha)$	$(\mathsf{left\text{-}type}, \alpha)$
$\mathsf{right}(\alpha, a)$	$(\mathsf{right\text{-}type}, \alpha, a)$
$(\otimes\, x : \alpha\ .\ \beta)$	$(\mathsf{dep\text{-}type\text{-}prod}, (\mathsf{var}, x, \alpha), \beta)$
$\langle a, b \rangle$	$(\mathsf{dep\text{-}ord\text{-}pair}, a, b)$
$\mathsf{hd}(a)$	$(\mathsf{dep\text{-}head}, a)$
$\mathsf{tl}(a)$	$(\mathsf{dep\text{-}tail}, a)$
$\ulcorner e \urcorner$	(quote, e)
$[\![a]\!]_k$	(eval, a, k)
$[\![a]\!]_{\mathrm{ty}}$	$(\mathsf{eval}, a, \mathsf{type})$
$[\![a]\!]_{\mathrm{te}}$	$(\mathsf{eval}, a, (\mathsf{con}, \mathsf{class}, \mathsf{type}))$
$[\![a]\!]_{\mathrm{fo}}$	$(\mathsf{eval}, a, \mathsf{formula})$

Table 6: Expanded Compact Notation

[7] W. M. Farmer. Biform theories in Chiron. In M. Kauers, M. Kerber, R. R. Miner, and W. Windsteiger, editors, *Towards Mechanized Mathematical Assistants*, volume 4573 of *Lecture Notes in Computer Science*, pages 66–79. Springer-Verlag, 2007.

[8] W. M. Farmer. Chiron: A multi-paradigm logic. In R. Matuszewski and A. Zalewska, editors, *From Insight to Proof: Festschrift in Honour of Andrzej Trybulec*, volume 10(23) of *Studies in Logic, Grammar and Rhetoric*, pages 1–19. University of Białystok, 2007.

[9] W. M. Farmer, J. D. Guttman, and F. J. Thayer. Little theories. In D. Kapur, editor, *Automated Deduction—CADE-11*, volume 607 of *Lecture Notes in Computer Science*, pages 567–581. Springer-Verlag, 1992.

[10] W. M. Farmer and M. von Mohrenschildt. An overview of a Formal Framework for Managing Mathematics. *Annals of Mathematics and Artificial Intelligence*, 38:165–191, 2003.

[11] M. J. Gabbay and A. M. Pitts. A new approach to abstract syntax involving binders. *Formal Aspects of Computing*, 13:341–363, 2002.

[12] K. Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte für Mathematik und Physik*, 38:173–198, 1931.

[13] K. Gödel. *The Consistency of the Axiom of Choice and the Generalized Continuum Hypothesis with the Axioms of Set Theory*, volume 3 of *Annals of Mathematical Studies*. Princeton University Press, 1940.

[14] S. Kleene. *Introduction to Metamathematics*. Van Nostrand, 1952.

[15] S. Kripke. Outline of a theory of truth. *Journal of Philosophy*, 72:690–716, 1975.

[16] H. Leitgeb. What theories of truth should be like (but cannot be). *Philosophy Compass*, 2:276–290, 2007.

[17] E. Mendelson. *Introduction to Mathematical Logic*. Chapman & Hall/CRC, fourth edition, 1997.

[18] L. G. Monk. Inference rules using local contexts. *Journal of Automated Reasoning*, 4:445–462, 1988.

[19] I. L. Novak. A construction for models of consistent systems. *Fundamenta Mathematicae*, 37:87–110, 1950.

[20] A. M. Pitts. Nominal Logic, a first order theory of names and binding. *Information and Computation*, 186:165–193, 2003.

[21] W. V. O. Quine. *Mathematical Logic: Revised Edition.* Harvard University Press, 2003.

[22] J. B. Rosser and H. Wang. Non-standard models for formal logics. *Journal of Symbolic Logic*, 15:113–129, 1950.

[23] J. Shoenfield. A relative consistency proof. *Journal of Symbolic Logic*, 19:21–28, 1954.

[24] J. R. Shoenfield. *Mathematical Logic.* Addison-Wesley, 1967.

[25] A. Tarski. Pojęcie prawdy w językach nauk dedukcyjnych (The concept of truth in the languages of the deductive sciences). *Prace Towarzystwa Naukowego Warszawskiego*, 3(34), 1933.

[26] A. Tarski. Der Wahrheitsbegriff in den formalisierten Sprachen. *Studia Philosophica*, 1:261–405, 1935.

[27] A. Tarski. The concept of truth in formalized languages. In J. Corcoran, editor, *Logic, Semantics, Meta-Mathematics*, pages 152–278. Hackett, second edition, 1983.

[28] B. C. van Fraassen. Singular terms, truth-value gaps, and free logic. *Journal of Philosophy*, 63:481–495, 1966.