Theory Morphisms in Church's Type Theory with Quotation and Evaluation*

William M. Farmer

Computing and Software, McMaster University, Canada
http://imps.mcmaster.ca/wmfarmer

23 July 2017
(Minor revision: 16 February 2024)

Abstract. CTT_{qe} is a version of Church's type theory with global quotation and evaluation operators that is engineered to reason about the interplay of syntax and semantics and to formalize syntax-based mathematical algorithms. CTT_{uqe} is a variant of CTT_{qe} that admits undefined expressions, partial functions, and multiple base types of individuals. It is better suited than CTT_{qe} as a logic for building networks of theories connected by theory morphisms. This paper presents the syntax and semantics of CTT_{uqe}, defines a notion of a theory morphism from one CTT_{uqe} theory to another, and gives two simple examples involving monoids that illustrate the use of theory morphisms in CTT_{uqe}.

1 Introduction

A *syntax-based mathematical algorithm (SBMA)*, such as a symbolic differentiation algorithm, manipulates mathematical expressions in a mathematically meaningful way. Reasoning about SBMAs requires reasoning about the relationship between how the expressions are manipulated and what the manipulations mean mathematically. We argue in [8] that a logic with quotation and evaluation would provide a global infrastructure for formalizing SBMAs and reasoning about the interplay of syntax and semantics that is embodied in them.

Quotation is a mechanism for referring to a syntactic value (e.g., a syntax tree) that represents the syntactic structure of an expression, while *evaluation* is a mechanism for referring to the value of the expression that a syntactic value represents. Incorporating quotation and evaluation into a traditional logic like first-order logic or simple type theory is tricky; there are several challenging problems that the logic engineer must overcome [8, 9]. CTT_{qe} [9, 10] is a version of Church's type theory with global quotation and evaluation operators inspired by the quote and eval operators in the Lisp programming language. We show

* Published in: H. Geuvers et al., eds, *Intelligent Computer Mathematics (CICM 2017)*, *Lecture Notes in Computer Science*, Vol. 10383, pp. 147–162, Springer, 2017. The final publication is available at Springer via http://dx.doi.org/10.1007/978-3-319-62075-6_11. This research was supported by NSERC.

in [9] that formula schemas and meaning formulas for SBMAs can be expressed in CTT_{qe} using quotation and evaluation and that such schemas and meaning formulas can be instantiated and proved within the proof system for CTT_{qe}.

The *little theories method* [11] is an approach for understanding and organizing mathematical knowledge as a *theory graph* [14] consisting of axiomatic *theories* as nodes and *theory morphisms*[1] as directed edges. A theory consists of a *language* of expressions that denote mathematical values and a set of *axioms* that express in the language assumptions about the values. A theory morphism is a meaning-preserving mapping from the formulas of one theory to the formulas of another theory. Theory morphisms serve as information conduits that enable definitions and theorems to be passed from an abstract theory to many other more concrete theories [2].

A *biform theory* [3, 6] is a combination of an axiomatic theory and an algorithmic theory (a collection of algorithms that perform symbolic computations). It consists of a *language* L generated from a set of *symbols*, a set of *transformers*, and a set of *axioms*. The expressions of L denote mathematical values that include syntactic values representing the expressions of L. The transformers are SBMAs and other algorithms that implement functions on the expressions of L and are represented by symbols of L. The axioms are formulas of L that express properties about the symbols and transformers of the biform theory. Unlike traditional logics, CTT_{qe} is well suited for formalizing biform theories. Can the little theories method be applied to biform theories formalized in CTT_{qe}? This would require a definition of a theory morphism for CTT_{qe} theories.

Defining a notion of a theory morphism in a logic with quotation is not as straightforward as in a logic without quotation due to the following problem:

> *Constant Interpretation Problem.* Let T_1 and T_2 be theories in a logic with a quotation operator $\ulcorner\cdot\urcorner$. If a theory morphism Φ from T_1 to T_2 interprets two distinct constants c and c' in T_1 by a single constant d in T_2, then Φ would map the true formula $\ulcorner c \urcorner \neq \ulcorner c' \urcorner$ of T_1 to the false formula $\ulcorner d \urcorner \neq \ulcorner d \urcorner$ of T_2, and hence Φ would not be meaning preserving. Similarly, if Φ interprets c as an expression e in T_2 that is not a constant, then Φ would map a true formula like $\textsf{is-constant}(\ulcorner c \urcorner)$ to the false formula $\textsf{is-constant}(\ulcorner e \urcorner)$.

This paper defines a notion of a theory morphism that overcomes this problem in CTT_{uqe}, a variant of CTT_{qe} that admits undefined expressions, partial functions, and multiple base types of individuals. CTT_{uqe} merges the machinery for quotation and evaluation found in CTT_{qe} [9] with the machinery for undefinedness found in $\mathcal{Q}_0^{\text{u}}$ [7]. Like CTT_{qe} and $\mathcal{Q}_0^{\text{u}}$, CTT_{uqe} is based on $\mathcal{Q}_0$ [1], Peter Andrews' elegant version of Church's type theory. See [9] for references related to CTT_{uqe}.

CTT_{uqe} is better suited than CTT_{qe} as a logic for the little theories method for two reasons. First, it is often convenient for a theory morphism from T_1 to T_2 to interpret different kinds of values by values of different types. Since CTT_{qe} contains only one base type of individuals, ι, all individuals in a theory T_1 must

[1] Theory morphisms are also known as *immersions*, *realizations*, *theory interpretations*, *translations*, and *views*.

be interpreted by values of the same type in T_2. Allowing multiple base types of individuals in $\textsc{ctt}_{\mathrm{uqe}}$ eliminates this restriction. Second, it is often useful to interpret a type α in T_1 by a subset of the denotation of a type β in T_2. As shown in [4], this naturally leads to partial functions on the type β. $\textsc{ctt}_{\mathrm{uqe}}$ has built-in support for partial functions and undefinedness based on the traditional approach to undefinedness [5]; $\textsc{ctt}_{\mathrm{qe}}$ has no such built-in support.[2]

The rest of the paper is organized as follows. The syntax and semantics of $\textsc{ctt}_{\mathrm{uqe}}$ are presented in sections 2 and 3. The notion of a theory morphism in $\textsc{ctt}_{\mathrm{uqe}}$ is defined in section 4. Section 5 contains two simple examples of theory morphisms in $\textsc{ctt}_{\mathrm{uqe}}$ involving monoids. The paper concludes in section 6 with a summary of the paper's results and some brief remarks about constructing theory morphisms in an implementation of $\textsc{ctt}_{\mathrm{uqe}}$ and about future work.

The syntax and semantics of $\textsc{ctt}_{\mathrm{uqe}}$ are presented as briefly as possible. The reader should consult [7] and [9] for a more in-depth discussion on the ideas underlying the syntax and semantics in $\textsc{ctt}_{\mathrm{uqe}}$. Due to limited space, a proof system is not given in this paper for $\textsc{ctt}_{\mathrm{uqe}}$. A proof system for $\textsc{ctt}_{\mathrm{uqe}}$ can be straightforwardly derived by merging the proof systems for $\textsc{ctt}_{\mathrm{qe}}$ [9] and $\mathcal{Q}_0^{\mathrm{u}}$ [7].

2 Syntax

The syntax of $\textsc{ctt}_{\mathrm{uqe}}$ is the same as the syntax of $\textsc{ctt}_{\mathrm{qe}}$ [9] except that (1) the types include denumerably many base types of individuals instead of just the single ι type, (2) the expressions include conditional expressions, and (3) the logical constants include constants for definite description and exclude $\mathsf{is\text{-}expr}_{\epsilon \rightarrow o}$ — which we will see is not needed since all constructions are "proper" in $\textsc{ctt}_{\mathrm{uqe}}$.

2.1 Types

Let $\mathcal{B}$ be a denumerable set of symbols that contains o and ϵ . A *type* of $\textsc{ctt}_{\mathrm{uqe}}$ is a string of symbols defined inductively by the following formation rules:

1. *Base type*: If $\alpha \in \mathcal{B}$, then α is a type.
2. *Function type*: If α and β are types, then $(\alpha \rightarrow \beta)$ is a type.

Let $\mathcal{T}$ denote the set of types of $\textsc{ctt}_{\mathrm{uqe}}$. o and ϵ are the *logical base types* of $\textsc{ctt}_{\mathrm{uqe}}$. $\alpha, \beta, \gamma, \ldots$ are syntactic variables ranging over types. When there is no loss of meaning, matching pairs of parentheses in types may be omitted. We assume that function type formation associates to the right so that a type of the form $(\alpha \rightarrow (\beta \rightarrow \gamma))$ may be written as $\alpha \rightarrow \beta \rightarrow \gamma$.

[2] A logic without support for partial functions and undefinedness — such as $\textsc{ctt}_{\mathrm{qe}}$ or the logic of HOL [12] — can interpret α by a type β' that is isomorphic to a subset of β. However, this approach is more complicated and farther from standard mathematics practice than interpreting α directly by a subset of β.

$=_{\alpha\rightarrow\alpha\rightarrow o}$	for all $\alpha \in \mathcal{T}$
$\iota_{(\alpha\rightarrow o)\rightarrow\alpha}$	for all $\alpha \in \mathcal{T}$ with $\alpha \neq o$
$\textsf{is-var}_{\epsilon\rightarrow o}$	
$\textsf{is-var}^{\alpha}_{\epsilon\rightarrow o}$	for all $\alpha \in \mathcal{T}$
$\textsf{is-con}_{\epsilon\rightarrow o}$	
$\textsf{is-con}^{\alpha}_{\epsilon\rightarrow o}$	for all $\alpha \in \mathcal{T}$
$\textsf{app}_{\epsilon\rightarrow\epsilon\rightarrow\epsilon}$	
$\textsf{abs}_{\epsilon\rightarrow\epsilon\rightarrow\epsilon}$	
$\textsf{cond}_{\epsilon\rightarrow\epsilon\rightarrow\epsilon\rightarrow\epsilon}$	
$\textsf{quo}_{\epsilon\rightarrow\epsilon}$	
$\textsf{is-expr}^{\alpha}_{\epsilon\rightarrow o}$	for all $\alpha \in \mathcal{T}$
$\sqsubset_{\epsilon\rightarrow\epsilon\rightarrow o}$	
$\textsf{is-free-in}_{\epsilon\rightarrow\epsilon\rightarrow o}$	

Table 1. Logical Constants

2.2 Expressions

A *typed symbol* is a symbol with a subscript from $\mathcal{T}$. Let $\mathcal{V}$ be a set of typed symbols such that $\mathcal{V}$ contains denumerably many typed symbols with subscript α for each $\alpha \in \mathcal{T}$. A *variable of type* α of $\textsc{ctt}_{\rm uqe}$ is a member of $\mathcal{V}$ with subscript α. $\mathbf{f}_\alpha, \mathbf{g}_\alpha, \mathbf{h}_\alpha, \mathbf{u}_\alpha, \mathbf{v}_\alpha, \mathbf{w}_\alpha, \mathbf{x}_\alpha, \mathbf{y}_\alpha, \mathbf{z}_\alpha, \ldots$ are syntactic variables ranging over variables of type α. We will assume that $f_\alpha, g_\alpha, h_\alpha, u_\alpha, v_\alpha, w_\alpha, x_\alpha, y_\alpha, z_\alpha, \ldots$ are actual variables of type α of $\textsc{ctt}_{\rm uqe}$.

Let $\mathcal{C}$ be a set of typed symbols disjoint from $\mathcal{V}$ that includes the typed symbols in Table 1. A *constant of type* α of $\textsc{ctt}_{\rm uqe}$ is a member of $\mathcal{C}$ with subscript α. The typed symbols in Table 1 are the *logical constants* of $\textsc{ctt}_{\rm uqe}$. $\mathbf{c}_\alpha, \mathbf{d}_\alpha, \ldots$ are syntactic variables ranging over constants of type α.

An *expression of type* α of $\textsc{ctt}_{\rm uqe}$ is a string of symbols defined inductively by the formation rules below. $\mathbf{A}_\alpha, \mathbf{B}_\alpha, \mathbf{C}_\alpha, \ldots$ are syntactic variables ranging over expressions of type α. An expression is *eval-free* if it is constructed using just the first six formation rules.

1. *Variable*: $\mathbf{x}_\alpha$ is an expression of type α.
2. *Constant*: $\mathbf{c}_\alpha$ is an expression of type α.
3. *Function application*: $(\mathbf{F}_{\alpha\rightarrow\beta}\, \mathbf{A}_\alpha)$ is an expression of type β.
4. *Function abstraction*: $(\lambda\, \mathbf{x}_\alpha \,.\, \mathbf{B}_\beta)$ is an expression of type $\alpha \rightarrow \beta$.
5. *Conditional*: $(\textsf{if}\ \mathbf{A}_o\ \mathbf{B}_\alpha\ \mathbf{C}_\alpha)$ is an expression of type α.
6. *Quotation*: $\ulcorner\mathbf{A}_\alpha\urcorner$ is an expression of type ϵ if $\mathbf{A}_\alpha$ is eval-free.
7. *Evaluation*: $[\![\mathbf{A}_\epsilon]\!]_{\mathbf{B}_\beta}$ is an expression of type β.

The purpose of the second argument $\mathbf{B}_\beta$ in an evaluation $[\![\mathbf{A}_\epsilon]\!]_{\mathbf{B}_\beta}$ is to establish the type of the evaluation.[3] A *formula* is an expression of type o. A *predicate* is

[3] It would be more natural for the second argument of an evaluation to be a type, but that would lead to an infinite family of evaluation operators, one for every type, since type variables are not available in $\textsc{ctt}_{\rm uqe}$ (as well as in $\textsc{ctt}_{\rm qe}$ and $\mathcal{Q}_0$).

an expression of a type of the form $\alpha \to o$. When there is no loss of meaning, matching pairs of parentheses in expressions may be omitted. We assume that function application formation associates to the left so that an expression of the form $((\mathbf{G}_{\alpha\to\beta\to\gamma}\,\mathbf{A}_\alpha)\,\mathbf{B}_\beta)$ may be written as $\mathbf{G}_{\alpha\to\beta\to\gamma}\,\mathbf{A}_\alpha\,\mathbf{B}_\beta$.

Remark 2.21 (Conditionals) We will see in the next section that $(\mathsf{if}\ \mathbf{A}_o\ \mathbf{B}_\alpha\ \mathbf{C}_\alpha)$ is a conditional expression that is not strict with respect to undefinedness. For instance, if $\mathbf{A}_o$ is true, then $(\mathsf{if}\ \mathbf{A}_o\ \mathbf{B}_\alpha\ \mathbf{C}_\alpha)$ denotes the value of $\mathbf{B}_\alpha$ even when $\mathbf{C}_\alpha$ is undefined. We construct conditionals using an expression constructor instead of a constant since constants always denote functions that are effectively strict with respect to undefinedness. We will use conditional expressions to restrict the domain of a function.

An occurrence of a variable $\mathbf{x}_\alpha$ in an eval-free expression $\mathbf{B}_\beta$ is *bound* [*free*] if (1) it is not in a quotation and (2) it is [not] in a subexpression of $\mathbf{B}_\beta$ of the form $\lambda\,\mathbf{x}_\alpha\,.\,\mathbf{C}_\gamma$. An eval-free expression is *closed* if no free variables occur in it.

2.3 Constructions

Let $\mathcal{E}$ be the function mapping eval-free expressions to expressions of type ϵ that is defined inductively as follows:

1. $\mathcal{E}(\mathbf{x}_\alpha) = \ulcorner\mathbf{x}_\alpha\urcorner$.
2. $\mathcal{E}(\mathbf{c}_\alpha) = \ulcorner\mathbf{c}_\alpha\urcorner$.
3. $\mathcal{E}(\mathbf{F}_{\alpha\to\beta}\,\mathbf{A}_\alpha) = \mathsf{app}_{\epsilon\to\epsilon\to\epsilon}\,\mathcal{E}(\mathbf{F}_{\alpha\to\beta})\,\mathcal{E}(\mathbf{A}_\alpha)$.
4. $\mathcal{E}(\lambda\,\mathbf{x}_\alpha\,.\,\mathbf{B}_\beta) = \mathsf{abs}_{\epsilon\to\epsilon\to\epsilon}\,\mathcal{E}(\mathbf{x}_\alpha)\,\mathcal{E}(\mathbf{B}_\beta)$.
5. $\mathcal{E}(\mathsf{if}\ \mathbf{A}_o\ \mathbf{B}_\alpha\ \mathbf{C}_\alpha) = \mathsf{cond}_{\epsilon\to\epsilon\to\epsilon\to\epsilon}\,\mathcal{E}(\mathbf{A}_o)\,\mathcal{E}(\mathbf{B}_\alpha)\,\mathcal{E}(\mathbf{C}_\alpha)$.
6. $\mathcal{E}(\ulcorner\mathbf{A}_\alpha\urcorner) = \mathsf{quo}_{\epsilon\to\epsilon}\,\mathcal{E}(\mathbf{A}_\alpha)$.

A *construction* of CTT_{uqe} is an expression in the range of $\mathcal{E}$. $\mathcal{E}$ is clearly injective. When $\mathbf{A}_\alpha$ is eval-free, $\mathcal{E}(\mathbf{A}_\alpha)$ is a construction that represents the syntactic structure of $\mathbf{A}_\alpha$. That is, $\mathcal{E}(\mathbf{A}_\alpha)$ is a syntactic value that represents how $\mathbf{A}_\alpha$ is constructed as an expression. In contrast to CTT_{qe}, the constructions of CTT_{uqe} do not include "improper constructions" — such as $\mathsf{app}_{\epsilon\to\epsilon\to\epsilon}\,\ulcorner\mathbf{x}_\alpha\urcorner\ulcorner\mathbf{x}_\alpha\urcorner$ — that do not represent the syntactic structures of eval-free expressions.

The six kinds of eval-free expressions and the syntactic values that represent their syntactic structures are given in Table 2.

2.4 Theories

Let $\mathcal{B}' \subseteq \mathcal{B}$ and $\mathcal{C}' \subseteq \mathcal{C}$. A type α of CTT_{uqe} is a $\mathcal{B}'$*-type* if each base type occurring in α is a member of $\mathcal{B}'$. An expression $\mathbf{A}_\alpha$ of CTT_{uqe} is a $(\mathcal{B}',\mathcal{C}')$-*expression* if each base type and constant occurring in $\mathbf{A}_\alpha$ is a member of $\mathcal{B}'$ and $\mathcal{C}'$, respectively. A *language* of CTT_{uqe} is the set of all $(\mathcal{B}',\mathcal{C}')$-expressions for some $\mathcal{B}' \subseteq \mathcal{B}$ and $\mathcal{C}' \subseteq \mathcal{C}$ such that $\mathcal{B}'$ contains the logical base types of CTT_{uqe} (i.e., o and ι) and $\mathcal{C}'$ contains the logical constants of CTT_{uqe}. A *theory* of CTT_{uqe} is a pair $T = (L, \mathit{\Gamma})$ where L is a language of CTT_{uqe} and $\mathit{\Gamma}$ is a set of formulas in L (called the *axioms* of T). $\mathbf{A}_\alpha$ is an *expression of a theory* T if $\mathbf{A}_\alpha \in L$.

2.5 Definitions and Abbreviations

As in [9], we introduce in Table 3 several defined logical constants and abbreviations. $(\mathbf{A}_\alpha\!\downarrow)$ says that $\mathbf{A}_\alpha$ is defined, and similarly, $(\mathbf{A}_\alpha\!\uparrow)$ says that $\mathbf{A}_\alpha$ is undefined. $\mathbf{A}_\alpha \simeq \mathbf{B}_\alpha$ says that $\mathbf{A}_\alpha$ and $\mathbf{B}_\alpha$ are *quasi-equal*, i.e., that $\mathbf{A}_\alpha$ and $\mathbf{B}_\alpha$ are either both defined and equal or both undefined. $\mathrm{I}\,\mathbf{x}_\alpha\ .\ \mathbf{A}_o$ is a *definite description*. It denotes the unique $\mathbf{x}_\alpha$ that satisfies $\mathbf{A}_o$. If there is no or more than one such $\mathbf{x}_\alpha$, it is undefined. The defined constant $\bot_\alpha$ is a canonical undefined expression of type α.

3 Semantics

The semantics of $\textsc{ctt}_{\mathrm{uqe}}$ is the same as the semantics of $\textsc{ctt}_{\mathrm{qe}}$ except that the former admits undefined expressions in accordance with the traditional approach to undefinedness [5]. Two principal changes are made to the $\textsc{ctt}_{\mathrm{qe}}$ semantics: (1) The notion of a general model is redefined to include partial functions as well as total functions. (2) The valuation function for expressions is made into a partial function that assigns a value to an expression iff the expression is defined according to the traditional approach.

3.1 Frames

A *frame* of $\textsc{ctt}_{\mathrm{uqe}}$ is a collection $\{D_\alpha \mid \alpha \in \mathcal{T}\}$ of domains such that:

1. $D_o = \{\textsc{t}, \textsc{f}\}$, the set of standard *truth values*.
2. D_ϵ is the set of *constructions* of $\textsc{ctt}_{\mathrm{uqe}}$.
3. For $\alpha \in \mathcal{B}$ with $\alpha \not\in \{o, \epsilon\}$, D_α is a nonempty set of values (called *individuals*).
4. For $\alpha, \beta \in \mathcal{T}$, $D_{\alpha\to\beta}$ is some set of *total* functions from D_α to D_β if $\beta = o$ and some set of *partial and total* functions from D_α to D_β if $\beta \neq o$.

3.2 Interpretations

An *interpretation* of $\textsc{ctt}_{\mathrm{uqe}}$ is a pair $(\{D_\alpha \mid \alpha \in \mathcal{T}\}, I)$ consisting of a frame and an interpretation function I that maps each constant in $\mathcal{C}$ of type α to an element of D_α such that:

1. For all $\alpha \in \mathcal{T}$, $I(=_{\alpha\to\alpha\to o})$ is the total function $f \in D_{\alpha\to\alpha\to o}$ such that, for all $d_1, d_2 \in D_\alpha$, $f(d_1)(d_2) = \textsc{t}$ iff $d_1 = d_2$.
2. For all $\alpha \in \mathcal{T}$ with $\alpha \neq o$, $I(\iota_{(\alpha\to o)\to\alpha})$ is the partial function $f \in D_{(\alpha\to o)\to\alpha}$ such that, for all $d \in D_{\alpha\to o}$, if the predicate d represents a singleton $\{d'\} \subseteq D_\alpha$, then $f(d) = d'$, and otherwise $f(d)$ is undefined.
3. $I(\mathsf{is\text{-}var}_{\epsilon\to o})$ the total function $f \in D_{\epsilon\to o}$ such that, for all constructions $\mathbf{A}_\epsilon \in D_\epsilon$, $f(\mathbf{A}_\epsilon) = \textsc{t}$ iff $\mathbf{A}_\epsilon = \ulcorner\mathbf{x}_\alpha\urcorner$ for some variable $\mathbf{x}_\alpha \in \mathcal{V}$ (where α can be any type).
4. For all $\alpha \in \mathcal{T}$, $I(\mathsf{is\text{-}var}^{\alpha}_{\epsilon\to o})$ is the total function $f \in D_{\epsilon\to o}$ such that, for all constructions $\mathbf{A}_\epsilon \in D_\epsilon$, $f(\mathbf{A}_\epsilon) = \textsc{t}$ iff $\mathbf{A}_\epsilon = \ulcorner\mathbf{x}_\alpha\urcorner$ for some variable $\mathbf{x}_\alpha \in \mathcal{V}$.

Kind	Syntax	Syntactic Value
Variable	$\mathbf{x}_\alpha$	$\ulcorner \mathbf{x}_\alpha \urcorner$
Constant	$\mathbf{c}_\alpha$	$\ulcorner \mathbf{c}_\alpha \urcorner$
Function application	$\mathbf{F}_{\alpha \rightarrow \beta}\, \mathbf{A}_\alpha$	$\mathsf{app}_{\epsilon \rightarrow \epsilon \rightarrow \epsilon}\, \mathcal{E}(\mathbf{F}_{\alpha \rightarrow \beta})\, \mathcal{E}(\mathbf{A}_\alpha)$
Function abstraction	$\lambda\, \mathbf{x}_\alpha \,.\, \mathbf{B}_\beta$	$\mathsf{abs}_{\epsilon \rightarrow \epsilon \rightarrow \epsilon}\, \mathcal{E}(\mathbf{x}_\alpha)\, \mathcal{E}(\mathbf{B}_\beta)$
Conditional	$(\mathsf{if}\ \mathbf{A}_o\ \mathbf{B}_\alpha\ \mathbf{C}_\alpha)$	$\mathsf{cond}_{\epsilon \rightarrow \epsilon \rightarrow \epsilon \rightarrow \epsilon}\, \mathcal{E}(\mathbf{A}_o)\, \mathcal{E}(\mathbf{B}_\alpha)\, \mathcal{E}(\mathbf{C}_\alpha)$.
Quotation	$\ulcorner \mathbf{A}_\alpha \urcorner$	$\mathsf{quo}_{\epsilon \rightarrow \epsilon}\, \mathcal{E}(\mathbf{A}_\alpha)$

Table 2. Six Kinds of Eval-Free Expressions

$(\mathbf{A}_\alpha = \mathbf{B}_\alpha)$	stands for	$=_{\alpha \rightarrow \alpha \rightarrow o}\ \mathbf{A}_\alpha\, \mathbf{B}_\alpha$.
$(\mathbf{A}_o \equiv \mathbf{B}_o)$	stands for	$=_{o \rightarrow o \rightarrow o}\ \mathbf{A}_o\, \mathbf{B}_o$.
T_o	stands for	$=_{o \rightarrow o \rightarrow o} = =_{o \rightarrow o \rightarrow o}$.
F_o	stands for	$(\lambda\, x_o \,.\, T_o) = (\lambda\, x_o \,.\, x_o)$.
$(\forall\, \mathbf{x}_\alpha \,.\, \mathbf{A}_o)$	stands for	$(\lambda\, \mathbf{x}_\alpha \,.\, T_o) = (\lambda\, \mathbf{x}_\alpha \,.\, \mathbf{A}_o)$.
$\wedge_{o \rightarrow o \rightarrow o}$	stands for	$\lambda\, x_o \,.\, \lambda\, y_o \,.$ $((\lambda\, g_{o \rightarrow o \rightarrow o} \,.\, g_{o \rightarrow o \rightarrow o}\, T_o\, T_o) =$ $(\lambda\, g_{o \rightarrow o \rightarrow o} \,.\, g_{o \rightarrow o \rightarrow o}\, x_o\, y_o))$.
$(\mathbf{A}_o \wedge \mathbf{B}_o)$	stands for	$\wedge_{o \rightarrow o \rightarrow o}\, \mathbf{A}_o\, \mathbf{B}_o$.
$\supset_{o \rightarrow o \rightarrow o}$	stands for	$\lambda\, x_o \,.\, \lambda\, y_o \,.\, (x_o = (x_o \wedge y_o))$.
$(\mathbf{A}_o \supset \mathbf{B}_o)$	stands for	$\supset_{o \rightarrow o \rightarrow o}\, \mathbf{A}_o\, \mathbf{B}_o$.
$\neg_{o \rightarrow o}$	stands for	$=_{o \rightarrow o \rightarrow o}\, F_o$.
$(\neg \mathbf{A}_o)$	stands for	$\neg_{o \rightarrow o}\, \mathbf{A}_o$.
$\vee_{o \rightarrow o \rightarrow o}$	stands for	$\lambda\, x_o \,.\, \lambda\, y_o \,.\, \neg(\neg x_o \wedge \neg y_o)$.
$(\mathbf{A}_o \vee \mathbf{B}_o)$	stands for	$\vee_{o \rightarrow o \rightarrow o}\, \mathbf{A}_o\, \mathbf{B}_o$.
$(\exists\, \mathbf{x}_\alpha \,.\, \mathbf{A}_o)$	stands for	$\neg(\forall\, \mathbf{x}_\alpha \,.\, \neg \mathbf{A}_o)$.
$(\mathbf{A}_\alpha \neq \mathbf{B}_\alpha)$	stands for	$\neg(\mathbf{A}_\alpha = \mathbf{B}_\alpha)$.
$\mathbf{A}_\epsilon \sqsubset_{\epsilon \rightarrow \epsilon \rightarrow o} \mathbf{B}_\epsilon$	stands for	$\sqsubset_{\epsilon \rightarrow \epsilon \rightarrow o}\ \mathbf{A}_\epsilon\, \mathbf{B}_\epsilon$.
$[\![\mathbf{A}_\epsilon]\!]_\beta$	stands for	$[\![\mathbf{A}_\epsilon]\!]_{\mathbf{B}_\beta}$.
$(\mathbf{A}_\alpha\!\downarrow)$	stands for	$\mathbf{A}_\alpha = \mathbf{A}_\alpha$.
$(\mathbf{A}_\alpha\!\uparrow)$	stands for	$\neg(\mathbf{A}_\alpha\!\downarrow)$.
$(\mathbf{A}_\alpha \simeq \mathbf{B}_\alpha)$	stands for	$(\mathbf{A}_\alpha\!\downarrow \vee\, \mathbf{B}_\alpha\!\downarrow) \supset \mathbf{A}_\alpha = \mathbf{B}_\alpha$.
$(\mathrm{I}\, \mathbf{x}_\alpha \,.\, \mathbf{A}_o)$	stands for	$\iota_{(\alpha \rightarrow o) \rightarrow \alpha}\, (\lambda\, \mathbf{x}_\alpha \,.\, \mathbf{A}_o)$ where $\alpha \neq o$.
$\bot_o$	stands for	F_o.
$\bot_\alpha$	stands for	$\mathrm{I}\, x_\alpha \,.\, x_\alpha \neq x_\alpha$ where $\alpha \neq o$.

Table 3. Definitions and Abbreviations

5. $I(\textsf{is-con}_{\epsilon\to o})$ is the total function $f \in D_{\epsilon\to o}$ such that, for all constructions $\mathbf{A}_\epsilon \in D_\epsilon$, $f(\mathbf{A}_\epsilon) = \text{T}$ iff $\mathbf{A}_\epsilon = \ulcorner\mathbf{c}_\alpha\urcorner$ for some constant $\mathbf{c}_\alpha \in \mathcal{C}$ (where α can be any type).
6. For all $\alpha \in \mathcal{T}$, $I(\textsf{is-con}^{\alpha}_{\epsilon\to o})$ is the total function $f \in D_{\epsilon\to o}$ such that, for all constructions $\mathbf{A}_\epsilon \in D_\epsilon$, $f(\mathbf{A}_\epsilon) = \text{T}$ iff $\mathbf{A}_\epsilon = \ulcorner\mathbf{c}_\alpha\urcorner$ for some constant $\mathbf{c}_\alpha \in \mathcal{C}$.
7. $I(\textsf{app}_{\epsilon\to\epsilon\to\epsilon})$ is the partial function $f \in D_{\epsilon\to\epsilon\to\epsilon}$ such that, for all constructions $\mathbf{A}_\epsilon, \mathbf{B}_\epsilon \in D_\epsilon$, if $\textsf{app}_{\epsilon\to\epsilon\to\epsilon}\,\mathbf{A}_\epsilon\,\mathbf{B}_\epsilon$ is a construction, then $f(\mathbf{A}_\epsilon)(\mathbf{B}_\epsilon) = \textsf{app}_{\epsilon\to\epsilon\to\epsilon}\,\mathbf{A}_\epsilon\,\mathbf{B}_\epsilon$, and otherwise $f(\mathbf{A}_\epsilon)(\mathbf{B}_\epsilon)$ is undefined.
8. $I(\textsf{abs}_{\epsilon\to\epsilon\to\epsilon})$ is the partial function $f \in D_{\epsilon\to\epsilon\to\epsilon}$ such that, for all constructions $\mathbf{A}_\epsilon, \mathbf{B}_\epsilon \in D_\epsilon$, if $\textsf{abs}_{\epsilon\to\epsilon\to\epsilon}\,\mathbf{A}_\epsilon\,\mathbf{B}_\epsilon$ is a construction, then $f(\mathbf{A}_\epsilon)(\mathbf{B}_\epsilon) = \textsf{abs}_{\epsilon\to\epsilon\to\epsilon}\,\mathbf{A}_\epsilon\,\mathbf{B}_\epsilon$, and otherwise $f(\mathbf{A}_\epsilon)(\mathbf{B}_\epsilon)$ is undefined.
9. $I(\textsf{cond}_{\epsilon\to\epsilon\to\epsilon\to\epsilon})$ is the partial function $f \in D_{\epsilon\to\epsilon\to\epsilon\to\epsilon}$ such that, for all constructions $\mathbf{A}_\epsilon, \mathbf{B}_\epsilon, \mathbf{C}_\epsilon \in D_\epsilon$, if $\textsf{cond}_{\epsilon\to\epsilon\to\epsilon\to\epsilon}\,\mathbf{A}_\epsilon\,\mathbf{B}_\epsilon\,\mathbf{C}_\epsilon$ is a construction, then $f(\mathbf{A}_\epsilon)(\mathbf{B}_\epsilon)(\mathbf{C}_\epsilon) = \textsf{cond}_{\epsilon\to\epsilon\to\epsilon\to\epsilon}\,\mathbf{A}_\epsilon\,\mathbf{B}_\epsilon\,\mathbf{C}_\epsilon$, and otherwise $f(\mathbf{A}_\epsilon)(\mathbf{B}_\epsilon)(\mathbf{C}_\epsilon)$ is undefined.
10. $I(\textsf{quo}_{\epsilon\to\epsilon})$ is the total function $f \in D_{\epsilon\to\epsilon}$ such that, for all constructions $\mathbf{A}_\epsilon \in D_\epsilon$, $f(\mathbf{A}_\epsilon) = \textsf{quo}_{\epsilon\to\epsilon}\,\mathbf{A}_\epsilon$.
11. For all $\alpha \in \mathcal{T}$, $I(\textsf{is-expr}^{\alpha}_{\epsilon\to o})$ is the total function $f \in D_{\epsilon\to o}$ such that, for all constructions $\mathbf{A}_\epsilon \in D_\epsilon$, $f(\mathbf{A}_\epsilon) = \text{T}$ iff $\mathbf{A}_\epsilon = \mathcal{E}(\mathbf{B}_\alpha)$ for some (eval-free) expression $\mathbf{B}_\alpha$.
12. $I(\sqsubset_{\epsilon\to\epsilon\to o})$ is the total function $f \in D_{\epsilon\to\epsilon\to\epsilon}$ such that, for all constructions $\mathbf{A}_\epsilon, \mathbf{B}_\epsilon \in D_\epsilon$, $f(\mathbf{A}_\epsilon)(\mathbf{B}_\epsilon) = \text{T}$ iff $\mathbf{A}_\epsilon$ is a proper subexpression of $\mathbf{B}_\epsilon$.
13. $I(\textsf{is-free-in}_{\epsilon\to\epsilon\to o})$ is the total function $f \in D_{\epsilon\to\epsilon\to\epsilon}$ such that, for all constructions $\mathbf{A}_\epsilon, \mathbf{B}_\epsilon \in D_\epsilon$, $f(\mathbf{A}_\epsilon)(\mathbf{B}_\epsilon) = \text{T}$ iff $\mathbf{A}_\epsilon = \ulcorner\mathbf{x}_\alpha\urcorner$ for some $\mathbf{x}_\alpha \in \mathcal{V}$ and $\mathbf{x}_\alpha$ is free in the expression $\mathbf{C}_\beta$ such that $\mathbf{B}_\epsilon = \mathcal{E}(\mathbf{C}_\beta)$.

An *assignment* into a frame $\{D_\alpha \mid \alpha \in \mathcal{T}\}$ is a function φ whose domain is $\mathcal{V}$ such that $\varphi(\mathbf{x}_\alpha) \in D_\alpha$ for each $\mathbf{x}_\alpha \in \mathcal{V}$. Given an assignment φ, $\mathbf{x}_\alpha \in \mathcal{V}$, and $d \in D_\alpha$, let $\varphi[\mathbf{x}_\alpha \mapsto d]$ be the assignment ψ such that $\psi(\mathbf{x}_\alpha) = d$ and $\psi(\mathbf{y}_\beta) = \varphi(\mathbf{y}_\beta)$ for all variables $\mathbf{y}_\beta$ distinct from $\mathbf{x}_\alpha$. For an interpretation $\mathcal{M} = (\{D_\alpha \mid \alpha \in \mathcal{T}\}, I)$, $\textsf{assign}(\mathcal{M})$ is the set of assignments into the frame of $\mathcal{M}$.

3.3 General Models

An interpretation $\mathcal{M} = (\{D_\alpha \mid \alpha \in \mathcal{T}), I\}$ is a *general model* for CTT_{uqe} if there is a partial binary valuation function $V^{\mathcal{M}}$ such that, for all assignments $\varphi \in \textsf{assign}(\mathcal{M})$ and expressions $\mathbf{D}_\delta$, either $V^{\mathcal{M}}_{\varphi}(\mathbf{D}_\delta) \in D_\delta$ or $V^{\mathcal{M}}_{\varphi}(\mathbf{D}_\delta)$ is undefined[4] and each of the following conditions is satisfied:

1. Let $\mathbf{D}_\delta \in \mathcal{V}$. Then $V^{\mathcal{M}}_{\varphi}(\mathbf{D}_\delta) = \varphi(\mathbf{D}_\delta)$.
2. Let $\mathbf{D}_\delta \in \mathcal{C}$. Then $V^{\mathcal{M}}_{\varphi}(\mathbf{D}_\delta) = I(\mathbf{D}_\delta)$.
3. Let $\mathbf{D}_\delta$ be $\mathbf{F}_{\alpha\to\beta}\,\mathbf{A}_\alpha$. If $V^{\mathcal{M}}_{\varphi}(\mathbf{F}_{\alpha\to\beta})$ is defined, $V^{\mathcal{M}}_{\varphi}(\mathbf{A}_\alpha)$ is defined, and the function $V^{\mathcal{M}}_{\varphi}(\mathbf{F}_{\alpha\to\beta})$ is defined at the argument $V^{\mathcal{M}}_{\varphi}(\mathbf{A}_\alpha)$, then
$$V^{\mathcal{M}}_{\varphi}(\mathbf{D}_\delta) = V^{\mathcal{M}}_{\varphi}(\mathbf{F}_{\alpha\to\beta})(V^{\mathcal{M}}_{\varphi}(\mathbf{A}_\alpha)).$$

[4] We write $V^{\mathcal{M}}_{\varphi}(\mathbf{D}_\delta)$ instead of $V^{\mathcal{M}}(\varphi, \mathbf{D}_\delta)$.

Otherwise, $V_\varphi^{\mathcal{M}}(\mathbf{D}_\delta) = \textsc{f}$ if $\beta = o$ and $V_\varphi^{\mathcal{M}}(\mathbf{D}_\delta)$ is undefined if $\beta \neq o$.

4. Let $\mathbf{D}_\delta$ be $\lambda\, \mathbf{x}_\alpha \,.\, \mathbf{B}_\beta$. Then $V_\varphi^{\mathcal{M}}(\mathbf{D}_\delta)$ is the (partial or total) function $f \in D_{\alpha\to\beta}$ such that, for each $d \in D_\alpha$, $f(d) = V_{\varphi[\mathbf{x}_\alpha \mapsto d]}^{\mathcal{M}}(\mathbf{B}_\beta)$ if $V_{\varphi[\mathbf{x}_\alpha \mapsto d]}^{\mathcal{M}}(\mathbf{B}_\beta)$ is defined and $f(d)$ is undefined if $V_{\varphi[\mathbf{x}_\alpha \mapsto d]}^{\mathcal{M}}(\mathbf{B}_\beta)$ is undefined.
5. Let $\mathbf{D}_\delta$ be $(\mathsf{if}\ \mathbf{A}_o\ \mathbf{B}_\alpha\ \mathbf{C}_\alpha)$. If $V_\varphi^{\mathcal{M}}(\mathbf{A}_o) = \textsc{t}$ and $V_\varphi^{\mathcal{M}}(\mathbf{B}_\alpha)$ is defined, then $V_\varphi^{\mathcal{M}}(\mathbf{D}_\delta) = V_\varphi^{\mathcal{M}}(\mathbf{B}_\alpha)$. If $V_\varphi^{\mathcal{M}}(\mathbf{A}_o) = \textsc{f}$ and $V_\varphi^{\mathcal{M}}(\mathbf{C}_\alpha)$ is defined, then $V_\varphi^{\mathcal{M}}(\mathbf{D}_\delta) = V_\varphi^{\mathcal{M}}(\mathbf{C}_\alpha)$. Otherwise, $V_\varphi^{\mathcal{M}}(\mathbf{D}_\delta)$ is undefined.
6. Let $\mathbf{D}_\delta$ be $\ulcorner \mathbf{A}_\alpha \urcorner$. Then $V_\varphi^{\mathcal{M}}(\mathbf{D}_\delta) = \mathcal{E}(\mathbf{A}_\alpha)$.
7. Let $\mathbf{D}_\delta$ be $[\![\mathbf{A}_\epsilon]\!]_\beta$. If $V_\varphi^{\mathcal{M}}(\mathsf{is\text{-}expr}_{\epsilon\to o}^{\beta}\, \mathbf{A}_\epsilon) = \textsc{t}$ and $V_\varphi^{\mathcal{M}}(\mathcal{E}^{-1}(V_\varphi^{\mathcal{M}}(\mathbf{A}_\epsilon)))$ is defined, then

$$V_\varphi^{\mathcal{M}}(\mathbf{D}_\delta) = V_\varphi^{\mathcal{M}}(\mathcal{E}^{-1}(V_\varphi^{\mathcal{M}}(\mathbf{A}_\epsilon))).$$

Otherwise, $V_\varphi^{\mathcal{M}}(\mathbf{D}_\delta) = \textsc{f}$ if $\beta = o$ and $V_\varphi^{\mathcal{M}}(\mathbf{D}_\delta)$ is undefined if $\beta \neq o$.

Proposition 3.31 *General models for* CTT_{uqe} *exist.*

Proof. The proof is similar to the proof of the analogous proposition in [9]. □

Other theorems about the semantics of CTT_{uqe} are the same or very similar to the theorems about the semantics of CTT_{qe} given in [9].

Let $\mathcal{M}$ be a general model for CTT_{uqe}. $\mathbf{A}_o$ is *valid in* $\mathcal{M}$, written $\mathcal{M} \vDash \mathbf{A}_o$, if $V_\varphi^{\mathcal{M}}(\mathbf{A}_o) = \textsc{t}$ for all $\varphi \in \mathsf{assign}(\mathcal{M})$. $\mathbf{A}_o$ is *valid in* CTT_{uqe}, written $\vDash \mathbf{A}_o$, if $\mathbf{A}_o$ is valid in every general model for CTT_{uqe}. An expression $\mathbf{B}_\beta$ is *semantically closed* if no variable "is effective in" it, i.e,

$$\vDash \forall\, \mathbf{y}_\alpha \,.\, ((\lambda\, \mathbf{x}_\alpha \,.\, \mathbf{B}_\beta)\, \mathbf{y}_\alpha = \mathbf{B}_\beta)$$

holds for all variables $\mathbf{x}_\alpha$ (where $\mathbf{y}_\alpha$ is any variable of type α that differs from $\mathbf{x}_\alpha$). It is easy to show that every closed eval-free expression is semantically closed. If $\mathbf{B}_\beta$ is semantically closed, then $V_\varphi^{\mathcal{M}}(\mathbf{B}_\beta)$ does not depend on $\varphi \in \mathsf{assign}(\mathcal{M})$. The notion of "$\mathbf{x}_\alpha$ is effective in $\mathbf{B}_\beta$" is discussed in detail in [9].

Let $T = (L, \mathit{\Gamma})$ be a theory of CTT_{uqe} and $\mathbf{A}_o$ be a formula of T. A *general model for* T is a general model $\mathcal{M}$ for CTT_{uqe} such that $\mathcal{M} \vDash \mathbf{A}_o$ for all $\mathbf{A}_o \in \mathit{\Gamma}$. $\mathbf{A}_o$ is *valid in* T, written $T \vDash \mathbf{A}_o$, if $\mathbf{A}_o$ is valid in every general model for T. T is *normal* if each member of $\mathit{\Gamma}$ is semantically closed.

4 Theory Morphisms

In this section we define a "semantic morphism" of CTT_{uqe} that maps the valid semantically closed formulas of one normal theory to the valid semantically closed formulas of another normal theory. Theory morphisms usually map base types to types. By exploiting the support for partial functions in CTT_{uqe}, we introduce a more general notion of theory morphism that maps base types to semantically closed predicates that represent sets of values of the same type. This requires mapping expressions denoting functions on the base type to expressions denoting functions with domains restricted to the semantically closed predicate.

For $i = 1, 2$, let $T_i = (L_i, \Gamma_i)$ be a normal theory of $\textsc{ctt}_{\mathrm{uqe}}$ where, for some $\mathcal{B}_i \subseteq \mathcal{B}$ and $\mathcal{C}_i \subseteq \mathcal{C}$, L_i is the set of all $(\mathcal{B}_i, \mathcal{C}_i)$-expressions. Also for $i = 1, 2$, let $\mathcal{T}_i$ be the set of all $\mathcal{B}_i$-types and $\mathcal{V}_i$ be the set of all variables in L_i. Finally, let $\mathcal{P}_2$ be the set of all semantically closed predicates in L_2.

4.1 Translations

In this section, we will define a translation from T_1 to T_2 to be a pair (μ, ν) of functions where μ interprets the base types of T_1 and ν interprets the variables and constants of T_1. $\overline{\mu}$ and $\overline{\nu}$ will be canonical extensions of μ and ν to the types and expressions of T_1, respectively.

Define τ to be the function that maps a predicate of type $\alpha \rightarrow o$ to the type α. When $\mathbf{p}_{\alpha \rightarrow o}$ and $\mathbf{q}_{\beta \rightarrow o}$ are semantically closed predicates, let

$$\mathbf{p}_{\alpha \rightarrow o} \rightharpoonup \mathbf{q}_{\beta \rightarrow o}$$

be an abbreviation for the following semantically closed predicate of type $(\alpha \rightarrow \beta) \rightarrow o$:

$$\lambda\, f_{\alpha \rightarrow \beta} \,.\, \forall\, x_\alpha \,.\, (f_{\alpha \rightarrow \beta}\, x_\alpha \not\simeq \bot_\beta \supset (\mathbf{p}_{\alpha \rightarrow o}\, x_\alpha \wedge \mathbf{q}_{\beta \rightarrow o}\, (f_{\alpha \rightarrow \beta}\, x_\alpha))).$$

If $\beta = o$ $[\beta \neq o]$, $\mathbf{p}_{\alpha \rightarrow o} \rightarrow \mathbf{q}_{\beta \rightarrow o}$ represents the set of total [partial and total] functions from the set of values represented by $\mathbf{p}_{\alpha \rightarrow o}$ to the set of values represented by $\mathbf{q}_{\beta \rightarrow o}$. Notice that

$$\tau(\mathbf{p}_{\alpha \rightarrow o} \rightharpoonup \mathbf{q}_{\beta \rightarrow o}) = \alpha \rightarrow \beta = \tau(\mathbf{p}_{\alpha \rightarrow o}) \rightarrow \tau(\mathbf{q}_{\beta \rightarrow o}).$$

Given a total function $\mu : \mathcal{B}_1 \rightarrow \mathcal{P}_2$, let $\overline{\mu} : \mathcal{T}_1 \rightarrow \mathcal{P}_2$ be the canonical extension of μ that is defined inductively as follows:

1. If $\alpha \in \mathcal{B}_1$, $\overline{\mu}(\alpha) = \mu(\alpha)$.
2. If $\alpha \rightarrow \beta \in \mathcal{T}_1$, $\overline{\mu}(\alpha \rightarrow \beta) = \overline{\mu}(\alpha) \rightharpoonup \overline{\mu}(\beta)$.

It is easy to see that μ is well-defined and total.

A *translation from* T_1 *to* T_2 is a pair $\Phi = (\mu, \nu)$, where $\mu : \mathcal{B}_1 \rightarrow \mathcal{P}_2$ is total and $\nu : \mathcal{V}_1 \cup \mathcal{C}_1 \rightarrow \mathcal{V}_2 \cup \mathcal{C}_2$ is total and injective, such that:

1. $\mu(o) = \lambda\, x_o \,.\, T_o$.
2. $\mu(\epsilon) = \lambda\, x_\epsilon \,.\, T_o$.
3. For each $\mathbf{x}_\alpha \in \mathcal{V}_1$, $\nu(\mathbf{x}_\alpha)$ is a variable in $\mathcal{V}_2$ of type $\tau(\overline{\mu}(\alpha))$.
4. For each $\mathbf{c}_\alpha \in \mathcal{C}_1$, $\nu(\mathbf{c}_\alpha)$ is a constant in $\mathcal{C}_2$ of type $\tau(\overline{\mu}(\alpha))$.

Throughout the rest of this section, let $\Phi = (\mu, \nu)$ be a translation from T_1 to T_2. $\overline{\nu} : L_1 \rightarrow L_2$ is the canonical extension of ν defined inductively as follows:

1. If $\mathbf{x}_\alpha \in \mathcal{V}_1$, $\overline{\nu}(\mathbf{x}_\alpha) = \nu(\mathbf{x}_\alpha)$.

2. If $\mathbf{c}_\alpha \in \mathcal{C}_1$, $\overline{\nu}(\mathbf{c}_\alpha) = \nu(\mathbf{c}_\alpha)$.
3. If $\mathbf{F}_{\alpha\to\beta}\,\mathbf{A}_\alpha \in L_1$, then $\overline{\nu}(\mathbf{F}_{\alpha\to\beta}\,\mathbf{A}_\alpha) = \overline{\nu}(\mathbf{F}_{\alpha\to\beta})\,\overline{\nu}(\mathbf{A}_\alpha)$.
4. If $\lambda\,\mathbf{x}_\alpha\,.\,\mathbf{B}_\beta \in L_1$, then $\overline{\nu}(\lambda\,\mathbf{x}_\alpha\,.\,\mathbf{B}_\beta) =$
$$\lambda\,\overline{\nu}(\mathbf{x}_\alpha)\,.\,(\mathsf{if}\;(\overline{\mu}(\alpha)\,\overline{\nu}(\mathbf{x}_\alpha))\;\overline{\nu}(\mathbf{B}_\beta)\;\bot_{\tau(\overline{\mu}(\beta))}).$$
5. If $(\mathsf{if}\;\mathbf{A}_o\;\mathbf{B}_\alpha\;\mathbf{C}_\alpha) \in L_1$, $\overline{\nu}(\mathsf{if}\;\mathbf{A}_o\;\mathbf{B}_\alpha\;\mathbf{C}_\alpha) = (\mathsf{if}\;\overline{\nu}(\mathbf{A}_o)\;\overline{\nu}(\mathbf{B}_\alpha)\;\overline{\nu}(\mathbf{C}_\alpha))$.
6. If $\ulcorner\mathbf{A}_\alpha\urcorner \in L_1$, then $\overline{\nu}(\ulcorner\mathbf{A}_\alpha\urcorner) = \ulcorner\overline{\nu}(\mathbf{A}_\alpha)\urcorner$.
7. If $[\![\mathbf{A}_\epsilon]\!]_{\mathbf{B}_\beta} \in L_1$, then $\overline{\nu}([\![\mathbf{A}_\epsilon]\!]_{\mathbf{B}_\beta}) = [\![\overline{\nu}(\mathbf{A}_\epsilon)]\!]_{\overline{\nu}(\mathbf{B}_\beta)}$.

Lemma 4.11
1. $\overline{\nu}$ is well-defined, total, and injective.
2. If $\boldsymbol{A}_\alpha \in L_1$, then $\overline{\nu}(\boldsymbol{A}_\alpha)$ is an expression of type $\tau(\overline{\mu}(\alpha))$.

Proof. The two parts of the proposition are easily proved simultaneously by induction on the structure of expressions. □

Remark 4.12 We overcome the Constant Interpretation Problem mentioned in section 1 by requiring ν to injectively map constants to constants which, by Lemma 4.11, implies that $\overline{\nu}$ injectively maps expressions to expressions. We will see in the next section that this requirement comes with a cost.

A formula in L_2 is an *obligation* of Φ if it is one of the following formulas:

1. $\exists\,x_{\tau(\mu(\alpha))}\,.\,\mu(\alpha)\,x_{\tau(\mu(\alpha))}$ where $\alpha \in \mathcal{B}_1$.
2. $\overline{\mu}(\alpha)\,\nu(\mathbf{c}_\alpha)$ where $\mathbf{c}_\alpha \in \mathcal{C}_1$.
3. $\nu(=_{\alpha\to\alpha\to o}) = \lambda\,x_{\alpha'}\,.\,\lambda\,y_{\alpha'}\,.\,(\mathsf{if}\;(\overline{\mu}(\alpha)\,x_{\alpha'} \wedge \overline{\mu}(\alpha)\,y_{\alpha'})\;(x_{\alpha'} =_{\alpha'\to\alpha'\to o} y_{\alpha'})\;\bot_o)$ where $\alpha \in \mathcal{T}_1$ and $\alpha' = \tau(\overline{\mu}(\alpha))$.
4. $\nu(\iota_{(\alpha\to o)\to\alpha}) = \lambda\,x_{\alpha'\to o}\,.\,(\mathsf{if}\;(\overline{\mu}(\alpha\to o)\,x_{\alpha'\to o})\;(\iota_{(\alpha'\to o)\to\alpha'}\,x_{\alpha'\to o})\;\bot_{\alpha'})$ where $\alpha \in \mathcal{T}_1$ with $\alpha \neq o$ and $\alpha' = \tau(\overline{\mu}(\alpha))$.
5. $\nu(\mathbf{c}_\alpha) = \mathbf{c}_\alpha$ where $\mathbf{c}_\alpha$ is $\mathsf{is\text{-}var}_{\epsilon\to o}$, $\mathsf{is\text{-}con}_{\epsilon\to o}$, $\mathsf{app}_{\epsilon\to\epsilon\to\epsilon}$, $\mathsf{abs}_{\epsilon\to\epsilon\to\epsilon}$, $\mathsf{cond}_{\epsilon\to\epsilon\to\epsilon\to\epsilon}$ $\mathsf{quo}_{\epsilon\to\epsilon}$, $\sqsubset_{\epsilon\to\epsilon\to o}$, or $\mathsf{is\text{-}free\text{-}in}_{\epsilon\to\epsilon\to o}$.
6. $\nu(\mathbf{c}^\beta_\alpha) = \mathbf{c}^{\tau(\overline{\mu}(\beta))}_\alpha$ where $\mathbf{c}_\alpha$ is $\mathsf{is\text{-}var}^\beta_{\epsilon\to o}$, $\mathsf{is\text{-}con}^\beta_{\epsilon\to o}$, or $\mathsf{is\text{-}expr}^\beta_{\epsilon\to o}$ and $\beta \in \mathcal{T}_1$.
7. $\overline{\nu}(\mathbf{A}_o)$ where $\mathbf{A}_o \in \Gamma_1$.

Notice that each obligation of Φ is semantically closed.

4.2 Semantic Morphisms

A *semantic morphism* from T_1 to T_2 is a translation (μ,ν) from T_1 to T_2 such that $T_1 \vDash \mathbf{A}_o$ implies $T_2 \vDash \overline{\nu}(\mathbf{A}_o)$ for all semantically closed formulas $\mathbf{A}_o$ of T_1. (A *syntactic morphism* from T_1 to T_2 would be a translation (μ,ν) from T_1 to T_2 such that $T_1 \vdash_P \mathbf{A}_o$ implies $T_2 \vdash_P \overline{\nu}(\mathbf{A}_o)$ for all semantically closed formulas $\mathbf{A}_o$ of T_1 where P is some proof system for CTT$_{\text{uqe}}$.) We will prove a theorem (called the Semantic Morphism Theorem) that gives a sufficient condition for a translation to be a semantic morphism.

Assume $\mathcal{M}_2 = (\{D^2_\alpha \mid \alpha \in \mathcal{T}\}, I_2)$ is a general model for T_2. Under the assumption that the obligations of Φ are valid in T_2, we will extract a general model for T_1 from $\mathcal{M}_2$.

For each $\alpha \in \mathcal{T}_1$, define $\underline{D}^2_{\tau(\overline{\mu}(\alpha))} \subseteq D^2_{\tau(\overline{\mu}(\alpha))}$ as follows:

1. $\underline{D}^2_{\tau(\overline{\mu}(o))} = \underline{D}^2_o = D^2_o = \{\text{T}, \text{F}\}$.
2. $\underline{D}^2_{\tau(\overline{\mu}(\epsilon))} = \underline{D}^2_\epsilon =$

   $$\{d \in D^2_\epsilon \mid d = V^{\mathcal{M}_2}_\varphi(\overline{\nu}(\mathbf{A}_\epsilon)) \text{ for some construction } \mathbf{A}_\epsilon \in L_1\}$$

   where φ is any member of $\mathsf{assign}(\mathcal{M}_2)$.
3. If $\alpha \in \mathcal{T}_1 \setminus \{o, \epsilon\}$, $\underline{D}^2_{\tau(\overline{\mu}(\alpha))} =$

   $$\{d \in D^2_{\tau(\overline{\mu}(\alpha))} \mid V^{\mathcal{M}_2}_\varphi(\overline{\mu}(\alpha))(d) = \text{T}\}$$

   where φ is any member of $\mathsf{assign}(\mathcal{M}_2)$.

For each $\alpha \in \mathcal{T}_1$, define $\overline{D}^1_\alpha$ inductively as follows:

1. $\overline{D}^1_o = \{\text{T}, \text{F}\}$.
2. $\overline{D}^1_\epsilon$ is the set of constructions of CTT_{uqe}.
3. If $\alpha \in \mathcal{B}_1 \setminus \{o, \epsilon\}$, $\overline{D}^1_\alpha = \underline{D}^2_{\tau(\overline{\mu}(\alpha))}$.
4. If $\alpha \rightarrow \beta \in \mathcal{T}_1$, then $\overline{D}^1_{\alpha\rightarrow\beta}$ is the set of all *total* functions from $\overline{D}^1_\alpha$ to $\overline{D}^1_\beta$ if $\beta = o$ and the set of all *partial and total* functions from $\overline{D}^1_\alpha$ to $\overline{D}^1_\beta$ if $\beta \neq o$.

For each $\alpha \in \mathcal{T}_1$, define $\rho_\alpha : \underline{D}^2_{\tau(\overline{\mu}(\alpha))} \rightarrow \overline{D}^1_\alpha$ inductively as follows:

1. If $d \in \underline{D}^2_\epsilon$, $\rho_\epsilon(d)$ is the unique construction $\mathbf{A}_\epsilon$ such that $\overline{\nu}(\mathbf{A}_\epsilon) = d$.
2. If $\alpha \in \mathcal{B}_1 \setminus \{\epsilon\}$ and $d \in \underline{D}^2_{\tau(\overline{\mu}(\alpha))}$, $\rho_\alpha(d) = d$.
3. If $\alpha \rightarrow \beta \in \mathcal{T}_1$ and $f \in \underline{D}^2_{\tau(\overline{\mu}(\alpha\rightarrow\beta))}$, $\rho_{\alpha\rightarrow\beta}(f)$ is the unique function $g \in \overline{D}^1_{\alpha\rightarrow\beta}$ such that, for all $d \in \underline{D}^2_{\tau(\overline{\mu}(\alpha))}$, either $f(d)$ and $g(\rho_\alpha(d))$ are both defined and $\rho_\beta(f(d)) = g(\rho_\alpha(d))$ or they are both undefined.

Lemma 4.21 *If $\alpha \in \mathcal{T}_1$, $\rho_\alpha : \underline{D}^2_{\tau(\overline{\mu}(\alpha))} \rightarrow \overline{D}^1_\alpha$ is well defined, total, and injective.*

Proof. This lemma is proved by induction on $\alpha \in \mathcal{T}_1$. ρ_ϵ is well defined since $V^{\mathcal{M}_2}_\varphi$ is identity function on constructions and $\overline{\nu}$ is injective by Lemma 4.11. □

For each $\alpha \in \mathcal{T}_1$, define $D^1_\alpha \subseteq \overline{D}^1_\alpha$ as follows:

1. If $\alpha \in \mathcal{B}_1$, $D^1_\alpha = \overline{D}^1_\alpha$.
2. If $\alpha \rightarrow \beta \in \mathcal{T}_1$, $D^1_{\alpha\rightarrow\beta}$ is the range of $\rho_{\alpha\rightarrow\beta}$.
3. If $\alpha \in \mathcal{B} \setminus \mathcal{B}_1$, D^1_α is any nonempty set.
4. If $\alpha \rightarrow \beta \in \mathcal{T} \setminus \mathcal{T}_1$, D^1_α is the set of all *total* functions from D^1_α to D^1_β if $\beta = o$ and the set of all *partial and total* functions from D^1_α to D^1_β if $\beta \neq o$.

For $\mathbf{c}_\alpha \in \mathcal{C}_1$, define $I_1(\mathbf{c}_\alpha) = \rho_\alpha(V^{\mathcal{M}_2}_\varphi(\overline{\nu}(\mathbf{c}_\alpha)))$ where φ is any member of $\mathsf{assign}(\mathcal{M}_2)$. Finally, define $\mathcal{M}_1 = (\{\mathcal{D}^1_\alpha \mid \alpha \in \mathcal{T}\}, I_1)$.

Lemma 4.22 *Suppose each obligation of Φ is valid in $\mathcal{M}_2$. Then $\mathcal{M}_1$ is a general model for T_2.*

Proof. By the first group of obligations of Φ, $\mathcal{D}^1_\alpha$ is nonempty for all $\alpha \in \mathcal{B}_1$, and so $\{\mathcal{D}^1_\alpha \mid \alpha \in \mathcal{T}\}$ is a frame of CTT_{uqe}. By the second to sixth groups of obligations of Φ, $\mathcal{M}_1$ is an interpretation of CTT_{uqe}. For all $\mathbf{A}_\alpha \in L_1$ and $\varphi \in \mathsf{assign}(\mathcal{M}_1)$, define $V^{\mathcal{M}_1}_\varphi(\mathbf{A}_\alpha)$ as follows:

($\star$) $V^{\mathcal{M}_1}_\varphi(\mathbf{A}_\alpha) = \rho_\alpha(V^{\mathcal{M}_2}_{\overline{\nu}(\varphi)}(\overline{\nu}(\mathbf{A}_\alpha)))$ if $V^{\mathcal{M}_2}_{\overline{\nu}(\varphi)}(\overline{\nu}(\mathbf{A}_\alpha))$ is defined and $V^{\mathcal{M}_1}_\varphi(\mathbf{A}_\alpha)$ is undefined otherwise,

where $\overline{\nu}(\varphi)$ is any $\psi \in \mathsf{assign}(\mathcal{M}_2)$ such that, for all $\mathbf{x}_\beta \in \mathcal{V}_1$, $\rho_\beta(\psi(\overline{\nu}(\mathbf{x}_\beta))) = \varphi(\mathbf{x}_\beta)$. This definition of $V^{\mathcal{M}_1}_\varphi$ can be easily extended to a valuation function on all expressions that can be shown, by induction on the structure of expressions, to satisfy the seven clauses of the definition of a general model. Therefore, $\mathcal{M}_1$ is a general model for CTT_{uqe}. Then ($\star$) implies

($\star\star$) $\mathcal{M}_1 \vDash \mathbf{A}_o$ iff $\mathcal{M}_2 \vDash \overline{\nu}(\mathbf{A}_o)$

for all semantically closed formulas $\mathbf{A}_o \in L_1$. By the seventh group of obligations of Φ, $\mathcal{M}_2 \vDash \overline{\nu}(\mathbf{A}_o)$ for all $\mathbf{A}_o \in \Gamma_1$, and thus $\mathcal{M}_1$ is a general model for T_1 by ($\star\star$). □

Theorem 4.23 (Semantic Morphism Theorem) *Let T_1 and T_2 be normal theories and Φ be a translation from T_1 to T_2. Suppose each obligation of Φ is valid in T_2. Then Φ is a semantic morphism from T_1 to T_2.*

Proof. Let $\Phi = (\mu, \nu)$ be a translation from T_1 to T_2 and suppose each obligation of Φ is valid in T_2. Let $\mathbf{A}_o \in L_1$ be semantically closed and valid in T_1. We must show that $\overline{\nu}(\mathbf{A}_o)$ is valid in every general model for T_2. Let $\mathcal{M}_2$ be a general model for T_1. (We are done if there are no general models for T_2.) Let $\mathcal{M}_1$ be extracted from $\mathcal{M}_2$ as above. Obviously, each obligation of Φ is valid in $\mathcal{M}_2$, and so $\mathcal{M}_1$ is a general model for T_1 by Lemma 4.22. Therefore, $\mathcal{M}_1 \vDash \mathbf{A}_o$, and so $\mathcal{M}_2 \vDash \overline{\nu}(\mathbf{A}_o)$ by ($\star\star$) in the proof of Lemma 4.22. □

Theorem 4.24 (Relative Satisfiability) *Let T_1 and T_2 be normal theories and suppose Φ is a semantic morphism of from T_1 to T_2. Then there is a general model for T_1 if there is a general model for T_2.*

Proof. Let $\Phi = (\mu, \nu)$ be a semantic morphism from T_1 to T_2, $\mathcal{M}_2$ be a general model for T_2, and $\mathcal{M}_1$ be extracted from $\mathcal{M}_2$ as above. Since Φ is a semantic morphism, each of its obligations is valid in T_2. Hence, $\mathcal{M}_1$ is a general model for T_1 by Lemma 4.22. □

5 Examples

We will illustrate the theory morphism machinery of CTT_{uqe} with two simple examples involving monoids, the first in which two concepts are interpreted as the same concept and second in which a type is interpreted as a subset of its denotation. Let $\mathcal{C}_{\log} \subseteq \mathcal{C}$ be the set of logical constants of CTT_{uqe}.

5.1 Example 1: Monoid with Left and Right Identity Elements

Define $M = (L_M, \mathit{\Gamma}_M)$ to be the usual theory of an abstract monoid where:

1. $\mathcal{B}_M = \{o, \epsilon, \iota\}$.
2. $\mathcal{C}_M = \mathcal{C}_{\mathrm{log}} \cup \{e_\iota, *_{\iota\to\iota\to\iota}\}$. ($*_{\iota\to\iota\to\iota}$ is written as an infix operator.)
3. L_M is the set of $(\mathcal{B}_M, \mathcal{C}_M)$ expressions.
4. $\mathcal{V}_M$ is the set of variables in L_M.
5. $\mathit{\Gamma}_M$ contains the following axioms:

   a. $\forall\, x_\iota \,.\, \forall\, y_\iota \,.\, \forall\, z_\iota \,.\, x_\iota *_{\iota\to\iota\to\iota} (y_\iota *_{\iota\to\iota\to\iota} z_\iota) = (x_\iota *_{\iota\to\iota\to\iota} y_\iota) *_{\iota\to\iota\to\iota} z_\iota$.
   b. $\forall\, x_\iota \,.\, e_\iota *_{\iota\to\iota\to\iota} x_\iota = x_\iota$.
   c. $\forall\, x_\iota \,.\, x_\iota *_{\iota\to\iota\to\iota} e_\iota = x_\iota$.

Define $M' = (L_{M'}, \mathit{\Gamma}_{M'})$ to be the alternate theory of an abstract monoid with left and right identity elements where:

1. $\mathcal{B}_{M'} = \mathcal{B}_M$.
2. $\mathcal{C}_{M'} = \mathcal{C}_{\mathrm{log}} \cup \{e_\iota^{\mathrm{left}}, e_\iota^{\mathrm{right}}, *_{\iota\to\iota\to\iota}\}$. ($*_{\iota\to\iota\to\iota}$ is written as an infix operator.)
3. $L_{M'}$ is the set of $(\mathcal{B}_{M'}, \mathcal{C}_{M'})$ expressions.
4. $\mathcal{V}_{M'} = \mathcal{V}_M$.
5. $\mathit{\Gamma}_{M'}$ contains the following axioms:

   a. $\forall\, x_\iota \,.\, \forall\, y_\iota \,.\, \forall\, z_\iota \,.\, x_\iota *_{\iota\to\iota\to\iota} (y_\iota *_{\iota\to\iota\to\iota} z_\iota) = (x_\iota *_{\iota\to\iota\to\iota} y_\iota) *_{\iota\to\iota\to\iota} z_\iota$.
   b. $\forall\, x_\iota \,.\, e_\iota^{\mathrm{left}} *_{\iota\to\iota\to\iota} x_\iota = x_\iota$.
   c. $\forall\, x_\iota \,.\, x_\iota *_{\iota\to\iota\to\iota} e_\iota^{\mathrm{right}} = x_\iota$.

We would like to construct a semantic morphism from M' to M that maps the left and right identity elements of M' to the single identity element of M. This is not possible since the mapping ν must be injective to overcome the Constant Interpretation Problem. We need to add a dummy constant to M to facilitate the definition of the semantic morphism. Let $\overline{M}$ be the definitional extension of M that contains the new constant e'_ι and the new axiom $e'_\iota = e_\iota$.[5]

Let $\mathit{\Phi} = (\mu, \nu)$ to be the translation from M' to $\overline{M}$ such that:

1. $\mu(\iota) = \lambda\, x_\iota \,.\, T_o$.
2. ν is the identity function on $\mathcal{V}_{M'} \cup \mathcal{C}_{\mathrm{log}} \cup \{*_{\iota\to\iota\to\iota}\}$.
3. $\nu(e_\iota^{\mathrm{left}}) = e_\iota$.
4. $\nu(e_\iota^{\mathrm{right}}) = e'_\iota$.

It is easy to see that $\mathit{\Phi}$ is a semantic morphism by Theorem 4.23 .

[5] Technically, e'_ι is a constant chosen from $\mathcal{C} \setminus \mathcal{C}_M$. There is no harm is assuming that such a constant already exists in $\mathcal{C}$.

5.2 Example 2: Monoid interpreted as the Trivial Monoid

The identity element of a monoid forms a submonoid of the monoid that is isomorphic with the trivial monoid consisting of a single element. There is a natural morphism from a theory of a monoid to itself in which the type of monoid elements is interpreted by the singleton set containing the identity element. This kind of morphism cannot be directly expressed using a definition of a theory morphism that maps base types to types. However, it can be directly expressed using the notion of a semantic morphism we have defined.

The desired translation interprets the type ι as the set $\{e_\iota\}$ and the constants denoting functions involving ι as functions in which the domain of ι is replaced by $\{e_\iota\}$. This is not possible since the mapping ν must map constants to constants to overcome the Constant Interpretation Problem. We need to add a set of dummy constants to M to facilitate the definition of the semantic morphism.

Define μ as follows:

1. For $\alpha \in \{o, \epsilon\}$, $\mu(\alpha) = \lambda\, x_\alpha \,.\, T_o$.
2. $\mu(\iota) = \lambda\, x_\iota \,.\, x_\iota = e_\iota$.

Let $\overline{M}$ be the definitional extension of M that contains the following the new defined constants:

1. $='_{\alpha\to\alpha\to o} = \lambda\, x_\alpha \,.\, \lambda\, y_\alpha \,.\, (\mathsf{if}\ (\overline{\mu}(\alpha)\, x_\alpha \wedge \overline{\mu}(\alpha)\, y_\alpha)\ (x_\alpha =_{\alpha\to\alpha\to o} y_\alpha)\ \bot_o)$ where $\alpha \in \mathcal{T}$ contains ι.
2. $\iota'_{(\alpha\to o)\to\alpha} = \lambda\, x_{\alpha\to o} \,.\, (\mathsf{if}\ (\overline{\mu}(\alpha \to o)\, x_{\alpha\to o})\ (\iota_{(\alpha\to o)\to\alpha}\, x_{\alpha\to o})\ \bot_\alpha)$ where $\alpha \in \mathcal{T}$ contains ι.
3. $*'_{\iota\to\iota\to\iota} = \lambda\, x_\iota \,.\, \lambda\, y_\iota \,.\, (\mathsf{if}\ (\overline{\mu}(\iota)\, x_\iota \wedge \overline{\mu}(\iota)\, y_\iota)\ (x_\iota *_{\iota\to\iota\to\iota} y_\iota)\ \bot_\iota)$.[6]

Let $\Psi = (\mu, \nu)$ to be the translation from M to $\overline{M}$ such that:

1. μ is defined as above.
2. ν is the identity function on $\mathcal{V}_{M'}$.
3. ν is the identity function on the members of $\mathcal{C}_{\log}$ except for the constants $=_{\alpha\to\alpha\to o}$ and $\iota_{(\alpha\to o)\to\alpha}$ where $\alpha \in \mathcal{T}$ contains ι.
4. $\nu(=_{\alpha\to\alpha\to o}) = =' _{\alpha\to\alpha\to o}$ for all $\alpha \in \mathcal{T}$ containing ι.
5. $\nu(\iota_{(\alpha\to o)\to\alpha}) = \iota'_{(\alpha\to o)\to\alpha}$ for all $\alpha \in \mathcal{T}$ containing ι.
6. $\nu(e_\iota) = e_\iota$.
7. $\nu(*_{\iota\to\iota\to\iota}) = *'_{\iota\to\iota\to\iota}$.

It is easy to see that Φ is a semantic morphism by Theorem 4.23 .

[6] The definition of $*'_{\iota\to\iota\to\iota}$ can be simplified by using the definition of μ and noting that $e_\iota *_{\iota\to\iota\to\iota} e_\iota$ equals e_ι.

6 Conclusion

CTT_{qe} is a version of Church's type theory with quotation and evaluation described in great detail in [9]. In this paper we have (1) presented CTT_{uqe}, a variant of CTT_{qe} that admits undefined expressions, partial functions, and multiple base types of individuals, (2) defined a notion of a theory morphism in CTT_{uqe}, and (3) given two simple examples that illustrate the use of theory morphisms in CTT_{uqe}. The theory morphisms of CTT_{uqe} overcome the Constant Interpretation Problem discussed in section 1 by requiring constants to be injectively mapped to constants. Since CTT_{uqe} admits partial functions, CTT_{uqe} theory morphisms are able to map base types to sets of values of the same type — which enables many additional natural meaning-preserving mappings between theories to be directly defined as CTT_{uqe} theory morphisms. Thus the paper demonstrates how theory morphisms can be defined in a traditional logic with quotation and evaluation and how support for partial functions in a traditional logic can be leveraged to obtain a wider class of theory morphisms.

The two examples presented in section 5 show that constructing a translation in CTT_{uqe} from a theory T_1 to a theory T_2 will often require defining new dummy constants in T_2. This is certainly a significant inconvenience. However, it is an inconvenience that can be greatly ameliorated in an implementation of CTT_{uqe} by allowing a user to define a "pre-translation" that is automatically transformed into a bona fide translation. A pre-translation from T_1 and T_2 would be a pair (μ, ν) where μ maps base types to either types or semantically closed predicates, ν maps constants to expressions that need not be constants, and ν is not required to be injective. From the pre-translation, the system would automatically extend T_2 to a theory T_2' and then construct a translation from T_1 to T_2'.

Our long-range goal is to implement a system for developing biform theory graphs utilizing logics equipped with quotation and evaluation. The next step in this direction is to implement CTT_{qe} by extending HOL Light [13], a simple implementation of HOL [12].

References

1. P. B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth through Proof, Second Edition.* Kluwer, 2002.
2. J. Barwise and J. Seligman. *Information Flow: The Logic of Distributed Systems*, volume 44 of *Tracts in Computer Science.* Cambridge University Press, 1997.
3. J. Carette and W. M. Farmer. High-level theories. In A. Autexier, J. Campbell, J. Rubio, M. Suzuki, and F. Wiedijk, editors, *Intelligent Computer Mathematics*, volume 5144 of *Lecture Notes in Computer Science*, pages 232–245. Springer, 2008.
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, 1994.
5. 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, 2004.

6. 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, 2007.
7. W. M. Farmer. Andrews' type system with undefinedness. In C. Benzmüller, C. Brown, J. Siekmann, and R. Statman, editors, *Reasoning in Simple Type Theory: Festschrift in Honor of Peter B. Andrews on his 70th Birthday*, Studies in Logic, pages 223–242. College Publications, 2008.
8. W. M. Farmer. The formalization of syntax-based mathematical algorithms using quotation and evaluation. In J. Carette, D. Aspinall, C. Lange, P. Sojka, and W. Windsteiger, editors, *Intelligent Computer Mathematics*, volume 7961 of *Lecture Notes in Computer Science*, pages 35–50. Springer, 2013.
9. W. M. Farmer. Incorporating quotation and evaluation into Church's type theory. *Computing Research Repository (CoRR)*, abs/1612.02785 (72 pp.), 2016.
10. W. M. Farmer. Incorporating quotation and evaluation into Church's type theory: Syntax and semantics. In M. Kohlhase, M. Johansson, B. Miller, L. de Moura, and F. Tompa, editors, *Intelligent Computer Mathematics*, volume 9791 of *Lecture Notes in Computer Science*, pages 83–98. Springer, 2016.
11. 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, 1992.
12. M. J. C. Gordon and T. F. Melham. *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*. Cambridge University Press, 1993.
13. J. Harrison. HOL Light: An overview. In S. Berghofer, T. Nipkow, C. Urban, and M. Wenzel, editors, *Theorem Proving in Higher Order Logics*, volume 5674 of *Lecture Notes in Computer Science*, pages 60–66. Springer, 2009.
14. M. Kohlhase. Mathematical knowledge management: Transcending the one-brain-barrier with theory graphs. *European Mathematical Society (EMS) Newsletter*, 92:22—-27, June 2014.