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
14 March 2017

Abstract. $\text{ctt}_{\text{qe}}$ 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. $\text{ctt}_{\text{uqe}}$ is a variant of $\text{ctt}_{\text{qe}}$ that admits undefined expressions, partial functions, and multiple base types of individuals. It is better suited than $\text{ctt}_{\text{qe}}$ as a logic for building networks of theories connected by theory morphisms. This paper presents the syntax and semantics of $\text{ctt}_{\text{uqe}}$, defines a notion of a theory morphism from one $\text{ctt}_{\text{uqe}}$ theory to another, and gives two simple examples that illustrate the use of theory morphisms in $\text{ctt}_{\text{uqe}}$.

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

The little theories method [11] is an approach for understanding and organizing mathematical knowledge as a theory graph [14] consisting of axiomatic

* This research was supported by NSERC.
theories as nodes and theory morphisms\footnote{Theory morphisms are also known as immersions, realizations, theory interpretations, translations, and views.} as directed edges. A theory consists of a set of symbols that denote mathematical values and define a language 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 \cite{2}.

A biform theory \cite{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 set of symbols, transformers, and axioms. The symbols denote mathematical values and define a language $L$. The values include syntactic values representing the expressions of $L$. The transformers are SBMAs and other algorithms whose input and output are expressions of $L$. The axioms are statements expressed in $L$ about the values and transformers of the biform theory. Unlike traditional logics, CTT\textsubscript{qe} is well suited for formalizing biform theories. Can the little theories method be applied to biform theories formalized in CTT\textsubscript{qe}? This would require a definition of a theory morphism for CTT\textsubscript{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 be $T_1$ and $T_2$ be theories in a logic with a quotation operator $\llbracket \cdot \rrbracket$. If a theory morphism $\Phi$ 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 $\Phi$ would map the true formula $\llbracket c \rrbracket \neq \llbracket c' \rrbracket$ of $T_1$ to the false formula $\llbracket d \rrbracket \neq \llbracket d \rrbracket$ of $T_2$, and hence $\Phi$ would not be meaning preserving. Similarly, if $\Phi$ interprets $c$ as an expression $e$ in $T_2$ that is not a constant, then $\Phi$ would map a true formula like $\text{is-constant}(\llbracket c \rrbracket)$ to the false formula $\text{is-constant}(\llbracket e \rrbracket)$.

This paper defines a notion of a theory morphism that overcomes this problem in CTT\textsubscript{uqe}, a variant of CTT\textsubscript{qe} that admits undefined expressions, partial functions, and multiple base types of individuals. CTT\textsubscript{uqe} merges the machinery for quotation and evaluation found in CTT\textsubscript{qe} \cite{9} with the machinery for undefinedness found in $Q_0$ \cite{7}. Like CTT\textsubscript{qe} and $Q_0$, CTT\textsubscript{uqe} is based on $Q_0$ \cite{1}, Peter Andrews’ elegant version of Church’s type theory. See \cite{9} for references related to CTT\textsubscript{uqe}.

CTT\textsubscript{uqe} is better suited than CTT\textsubscript{qe} as a logic for the little theories method for two reasons. First, it is often convenience for a theory morphism from $T_1$ to $T_2$ to interpret different kinds of values by values of different types. Since CTT\textsubscript{qe} contains only one base type of individuals, $\iota$, all individuals in a theory $T_1$ must be interpreted by values of the same type in $T_2$. Allowing multiple base types of individuals in CTT\textsubscript{uqe} eliminates this restriction. Second, it is often convenient to interpret a type $\alpha$ in $T_1$ by a subset of the denotation of a type $\beta$ in $T_2$. As shown in \cite{4}, this naturally leads to partial functions on the type $\beta$. CTT\textsubscript{uqe} has built-in support for partial functions and undefinedness based on the traditional approach to undefinedness \cite{5}; CTT\textsubscript{qe} has no such built-in support.
The rest of the paper is organized as follows. The syntax and semantics of CTT_{uqe} are presented in sections 2 and 3. The notion of a theory morphism in CTT_{uqe} is defined in section 4. Section 5 contains two simple examples of theory morphisms in CTT_{uqe}. 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 CTT_{uqe} and about future work.

The syntax and semantics of CTT_{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 CTT_{uqe}. Due to limited space, a proof system is not given in this paper for CTT_{uqe}. A proof system for CTT_{uqe} can be straightforwardly derived by merging the proof systems for CTT_{qe} [9] and Q_0 [7].

2 Syntax

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

2.1 Types

Let \( \mathcal{B} \) be a denumerable set of symbols that contains \( o \) and \( \epsilon \). A type of CTT_{uqe} is a string of symbols defined inductively by the following formation rules:

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

Let \( \mathcal{T} \) denote the set of types of CTT_{uqe}. \( o \) and \( \epsilon \) are the logical base types of CTT_{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 \( \alpha \rightarrow \beta \rightarrow \gamma \).

2.2 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 \( \alpha \) for each \( \alpha \in \mathcal{T} \). A variable of type \( \alpha \) of CTT_{uqe} is a member of \( \mathcal{V} \) with subscript \( \alpha \). \( f_\alpha, g_\alpha, h_\alpha, u_\alpha, v_\alpha, w_\alpha, x_\alpha, y_\alpha, z_\alpha, \ldots \) are syntactic variables ranging over variables of type \( \alpha \). 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 \( \alpha \) of CTT_{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 \( \alpha \) of CTT_{uqe} is a member of \( \mathcal{C} \) with subscript \( \alpha \). The typed symbols in Table 1 are the logical constants of CTT_{uqe}. \( c_\alpha, d_\alpha, \ldots \) are syntactic variables ranging over constants of type \( \alpha \).
An expression of type $\alpha$ of $\text{CTT}_{\text{up}}$ is a string of symbols defined inductively by the formation rules below. $A, B, C, \ldots$ are syntactic variables ranging over expressions of type $\alpha$. An expression is eval-free if it is constructed using just the first six formation rules.

1. **Variable**: $x_\alpha$ is an expression of type $\alpha$.
2. **Constant**: $c_\alpha$ is an expression of type $\alpha$.
3. **Function application**: $(F_{\alpha \to \beta} A_\alpha)$ is an expression of type $\beta$.
4. **Function abstraction**: $(\lambda x_\alpha . B_\beta)$ is an expression of type $\alpha \to \beta$.
5. **Conditional**: $(\text{if } A_o B_\alpha C_\alpha)$ is an expression of type $\alpha$.
6. **Quotation**: $\langle A_\alpha \rangle$ is an expression of type $\epsilon$ if $A_\alpha$ is eval-free.
7. **Evaluation**: $[A_\epsilon]_{B_\beta}$ is an expression of type $\beta$.

The purpose of the second component $B_\beta$ in an evaluation $[A_\epsilon]_{B_\beta}$ is to establish the type of the evaluation. A formula is an expression of type $o$. A unary predicate is 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 $(G_{\alpha \to \gamma} A_\alpha) B_\beta)$ may be written as $G_{\alpha \to \gamma} A_\alpha B_\beta$.

**Remark 2.21 (Conditionals)** We will see in the next section that (if $A_\alpha B_\alpha C_\alpha$) is a conditional expression that is not strict with respect to undefinedness. For instance, if $A_\alpha$ is true, then (if $A_\alpha B_\alpha C_\alpha$) denotes the value of $B_\alpha$ even when $C_\alpha$ is undefined. We construct conditionals using an expression constructor instead of using a constant since constants always denote functions that are effectively strict with respect to undefinedness.

An occurrence of a variable $x_\alpha$ in an eval-free expression $B_\beta$ is bound [free] if (1) it is not in a quotation and (2) it is [not] in a subexpression of $B_\beta$ of the form $\lambda x_\alpha . C_\gamma$. 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 $\epsilon$ that is defined inductively as follows:

1. $\mathcal{E}(x_\alpha) = \overline{x_\alpha}$.
2. $\mathcal{E}(c_\alpha) = \overline{c_\alpha}$.
3. $\mathcal{E}(F_{\alpha \rightarrow \beta} A_\alpha) = \text{app}_{\epsilon \rightarrow \epsilon \rightarrow \epsilon} \mathcal{E}(F_{\alpha \rightarrow \beta}) \mathcal{E}(A_\alpha)$.
4. $\mathcal{E}(\lambda x_\alpha \cdot B_\beta) = \text{abs}_{\epsilon \rightarrow \epsilon \rightarrow \epsilon} \mathcal{E}(x_\alpha) \mathcal{E}(B_\beta)$.
5. $\mathcal{E}(\text{if } A_\alpha \circ B_\alpha C_\alpha) = \text{cond}_{\epsilon \rightarrow \epsilon \rightarrow \epsilon \rightarrow \epsilon} \mathcal{E}(A_\alpha) \mathcal{E}(B_\alpha) \mathcal{E}(C_\alpha)$.
6. $\mathcal{E}(\overline{\alpha_\alpha}) = \text{quo}_{\epsilon \rightarrow \epsilon} \mathcal{E}(A_\alpha)$.

A construction of \texttt{ctt uqe} is an expression in the range of $\mathcal{E}$. $\mathcal{E}$ is clearly injective.

When $A_\alpha$ is eval-free, $\mathcal{E}(A_\alpha)$ is a construction that represents the syntactic structure of $A_\alpha$. That is, $\mathcal{E}(A_\alpha)$ is a syntactic value that represents how $A_\alpha$ is constructed as an expression. In contrast to \texttt{ctt uqe}, the constructions of \texttt{ctt uqe} do not include “improper constructions” — such as $\text{app}_{\epsilon \rightarrow \epsilon \rightarrow \epsilon} \overline{x_\alpha} \overline{x_\alpha}$ — 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 $B' \subseteq B$ and $C' \subseteq C$. A type $\alpha$ of \texttt{ctt uqe} is a $B'$-type if each base type occurring in $\alpha$ is a member of $B'$. An expression $A_\alpha$ of \texttt{ctt uqe} is a $(B', C')$-expression if each base type and constant occurring in $A_\alpha$ is a member of $B'$ and $C'$, respectively. A language of \texttt{ctt uqe} is the set of all $(B', C')$-expressions for some $B' \subseteq B$ and $C' \subseteq C$ such that $B'$ contains the logical base types of \texttt{ctt uqe} (i.e., $o$ and $i$) and $C'$ contains the logical constants of \texttt{ctt uqe}. A theory of \texttt{ctt uqe} is a pair $T = (L, \Gamma)$ where $L$ is a language of \texttt{ctt uqe} and $\Gamma$ is a set of formulas in $L$ (called the axioms of $T$). $A_\alpha$ is an expression of a theory $T = (L, \Gamma)$ if $A_\alpha \in L$.

2.5 Definitions and Abbreviations

As in [9], we introduce in Table 3 several defined logical constants and abbreviations. $(A_\alpha \downarrow)$ says that $A_\alpha$ is defined, and similarly, $(A_\alpha \uparrow)$ says that $A_\alpha$ is undefined. $A_\alpha \simeq B_\alpha$ says that $A_\alpha$ and $B_\alpha$ are quasi-equal, i.e., that $A_\alpha$ and $B_\alpha$ are either both defined and equal or both undefined. $1x_\alpha$. $A_\alpha$ is a definite description. It denotes the unique $x_\alpha$ that satisfies $A_\alpha$. If there is no or more than one such $x_\alpha$, it is undefined. The defined constant $\bot_\alpha$ is a canonical undefined expression of type $\alpha$. 
| Kind          | Syntax       | Syntactic Value |
|--------------|--------------|-----------------|
| Variable     | $x_\alpha$   | $\downarrow x_\alpha$ |
| Constant     | $c_\alpha$   | $\downarrow c_\alpha$ |
| Function application | $F_{\alpha \rightarrow \beta} A_\alpha$ | $\text{app}_{\alpha \rightarrow \epsilon} \mathcal{E}(F_{\alpha \rightarrow \beta}) \mathcal{E}(A_\alpha)$ |
| Function abstraction | $\lambda x_\alpha \cdot B_\beta$ | $\text{abs}_{\alpha \rightarrow \epsilon} \mathcal{E}(x_\alpha) \mathcal{E}(B_\beta)$ |
| Conditional  | $(\text{if } A_\alpha B_\alpha C_\alpha)$ | $\text{cond}_{\alpha \rightarrow \epsilon} \mathcal{E}(A_\alpha) \mathcal{E}(B_\alpha) \mathcal{E}(C_\alpha)$ |
| Quotation    | $\overline{A_\alpha}$ | $\overline{\mathcal{E}(A_\alpha)}$ |

Table 2. Six Kinds of Eval-Free Expressions

\[
\begin{array}{ll}
\alpha = \beta & \text{stands for } \equiv_{\alpha \rightarrow \epsilon} A_\alpha B_\alpha.
\\
\alpha \equiv \beta & \text{stands for } \equiv_{\alpha \rightarrow \epsilon} A_\alpha B_\beta.
\\
\top & \text{stands for } \equiv_{\alpha \rightarrow \epsilon} A_\alpha B_\alpha.
\\
\bot & \text{stands for } \equiv_{\alpha \rightarrow \epsilon} F_\alpha.
\\
\lambda x_\alpha \cdot \beta & \text{stands for } \equiv_{\alpha \rightarrow \epsilon} (\lambda A_\alpha \cdot \beta).
\\
\forall x_\alpha \cdot A_\alpha & \text{stands for } \equiv_{\alpha \rightarrow \epsilon} (\lambda \exists y_\alpha \cdot \beta).
\\
\\neg A_\alpha & \text{stands for } \equiv_{\alpha \rightarrow \epsilon} (\lambda \exists y_\alpha \cdot \beta).
\\
\alpha \lor \beta & \text{stands for } \equiv_{\alpha \rightarrow \epsilon} (\lambda \exists y_\alpha \cdot \beta).
\\
\exists x_\alpha \cdot A_\alpha & \text{stands for } \equiv_{\alpha \rightarrow \epsilon} (\lambda \exists y_\alpha \cdot \beta).
\\
\alpha \cap \beta & \text{stands for } \equiv_{\alpha \rightarrow \epsilon} (\lambda \exists y_\alpha \cdot \beta).
\\
\llbracket A_\alpha \rrbracket_\beta & \text{stands for } \equiv_{\alpha \rightarrow \epsilon} (\lambda \exists y_\alpha \cdot \beta).
\\
A_\alpha \uparrow & \text{stands for } \equiv_{\alpha \rightarrow \epsilon} (\lambda \exists y_\alpha \cdot \beta).
\\
A_\alpha \downarrow & \text{stands for } \equiv_{\alpha \rightarrow \epsilon} (\lambda \exists y_\alpha \cdot \beta).
\\
\alpha \triangleright \beta & \text{stands for } \equiv_{\alpha \rightarrow \epsilon} (\lambda \exists y_\alpha \cdot \beta).
\\
\alpha \triangleleft \beta & \text{stands for } \equiv_{\alpha \rightarrow \epsilon} (\lambda \exists y_\alpha \cdot \beta).
\\
\text{i}_{(\alpha \rightarrow \epsilon)} & \text{stands for } \equiv_{\alpha \rightarrow \epsilon} (\lambda \exists y_\alpha \cdot \beta) \text{ where } \alpha \neq \beta.
\\
\bot & \text{stands for } F_\alpha.
\\
\top & \text{stands for } \equiv_{\alpha \rightarrow \epsilon} (\lambda \exists y_\alpha \cdot \beta) \text{ where } \alpha \neq \beta.
\end{array}
\]

Table 3. Definitions and Abbreviations
3 Semantics

The semantics of CTT_uqe is the same as the semantics of CTTqe except that the former admits undefined expressions in accordance with the traditional approach to undefinedness [5]. Two principal changes are made to the CTT_uqe 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 CTT_uqe is a collection \{D_\alpha | \alpha \in \mathcal{T}\} of domains such that:

1. \(D_\alpha = \{\top, \bot\}\), the set of standard truth values.
2. \(D_\alpha\) is the set of constructions of CTT_uqe.
3. For \(\alpha \in \mathcal{B}\) with \(\alpha \notin \{o, e\}\), \(D_\alpha\) is a nonempty set of values (called individuals).
4. For \(\alpha, \beta \in \mathcal{T}\), \(D_{\alpha \rightarrow \beta}\) is some set of total functions from \(D_\alpha\) to \(D_\beta\) if \(\beta = o\) and some set of partial and total functions from \(D_\alpha\) to \(D_\beta\) if \(\beta \neq o\).

3.2 Interpretations

An interpretation of CTT_uqe is a pair \(\langle \{D_\alpha | \alpha \in \mathcal{T}\}, I\rangle\) consisting of a frame and an interpretation function \(I\) that maps each constant in \(C\) of type \(\alpha\) to an element of \(D_\alpha\) such that:

1. For all \(\alpha \in \mathcal{T}\), \(I(\alpha)\) is the total function \(f \in D_{\alpha \rightarrow o}\) such that, for all \(d_1, d_2 \in D_\alpha\), \(f(d_1)(d_2) = T\) iff \(d_1 = d_2\).
2. For all \(\alpha \in \mathcal{T}\) with \(\alpha \neq o\), \(I(\alpha)\) is the partial function \(f \in D_{(\alpha \rightarrow o) \rightarrow \alpha}\) such that, for all \(d \in D_{\alpha \rightarrow o}\), if the predicate \(d\) represents a singleton \(\{d'\} \subseteq D_\alpha\), then \(f(d) = d'\), and otherwise \(f(d)\) is undefined.
3. \(I(\text{is-var}_{\alpha \rightarrow o})\) the the total function \(f \in D_{\alpha \rightarrow o}\) such that, for all constructions \(\alpha_\epsilon \in D_\epsilon\), \(f(\alpha_\epsilon) = T\) iff \(\alpha_\epsilon = \lceil x_\alpha \rceil\) for some variable \(x_\alpha \in \mathcal{V}\) (where \(\alpha\) can be any type).
4. For all \(\alpha \in \mathcal{T}\), \(I(\text{is-con}_{\alpha \rightarrow o})\) is the total function \(f \in D_{\alpha \rightarrow o}\) such that, for all constructions \(\alpha_\epsilon \in D_\epsilon\), \(f(\alpha_\epsilon) = T\) iff \(\alpha_\epsilon = \lceil c_\alpha \rceil\) for some constant \(c_\alpha \in \mathcal{C}\) (where \(\alpha\) can be any type).
5. For all \(\alpha \in \mathcal{T}\), \(I(\text{app}_{\alpha \rightarrow o})\) is the total function \(f \in D_{\alpha \rightarrow o}\) such that, for all constructions \(\alpha_\epsilon, B_\epsilon \in D_\epsilon\), if \(\text{app}_\epsilon(\alpha_\epsilon) B_\epsilon\) is a construction, then \(f(\alpha_\epsilon)(B_\epsilon) = \text{app}_\epsilon(\alpha_\epsilon) B_\epsilon\), and otherwise \(f(\alpha_\epsilon)(B_\epsilon)\) is undefined.
6. For all \(\alpha \in \mathcal{T}\), \(I(\text{abs}_{\alpha \rightarrow o})\) is the total function \(f \in D_{\alpha \rightarrow o}\) such that, for all constructions \(\alpha_\epsilon, B_\epsilon \in D_\epsilon\), if \(\text{abs}_\epsilon(\alpha_\epsilon) B_\epsilon\) is a construction, then \(f(\alpha_\epsilon)(B_\epsilon) = \text{abs}_\epsilon(\alpha_\epsilon) B_\epsilon\), and otherwise \(f(\alpha_\epsilon)(B_\epsilon)\) is undefined.
9. \(I(\text{cond}_{\alpha \rightarrow \beta \rightarrow \gamma \rightarrow \epsilon})\) is the partial function \(f \in D_{\alpha \rightarrow \beta \rightarrow \gamma \rightarrow \epsilon}\) such that, for all constructions \(A_\epsilon, B_\epsilon, C_\epsilon \in D_\epsilon\), if \(\text{cond}_{\alpha \rightarrow \beta \rightarrow \gamma \rightarrow \epsilon} A_\epsilon B_\epsilon C_\epsilon\) is a construction, then \(f(A_\epsilon)(B_\epsilon)(C_\epsilon) = \text{cond}_{\alpha \rightarrow \beta \rightarrow \gamma \rightarrow \epsilon} A_\epsilon B_\epsilon C_\epsilon\), and otherwise \(f(A_\epsilon)(B_\epsilon)(C_\epsilon)\) is undefined.

10. \(I(q\alpha_{\epsilon \rightarrow \delta})\) is the total function \(f \in D_{\epsilon \rightarrow \delta}\) such that, for all constructions \(A_\epsilon \in D_\epsilon\), \(f(A_\epsilon) = q\alpha_{\epsilon \rightarrow \delta} A_\epsilon\).

11. For all \(\alpha \in T\), \(I(\text{is-exp}_{\alpha \rightarrow \beta})\) is the total function \(f \in D_{\alpha \rightarrow \beta}\) such that, for all constructions \(A_\alpha, B_\alpha \in D_\alpha\), \(f(A_\alpha) = \top\) if \(A_\alpha = E(B_\alpha)\) for some (eval-free) expression \(B_\alpha\).

12. \(I(\subset_{\epsilon \rightarrow \delta})\) is the total function \(f \in D_{\epsilon \rightarrow \delta}\) such that, for all constructions \(A_\epsilon, B_\epsilon \in D_\epsilon\), \(f(A_\epsilon)(B_\epsilon) = \top\) if \(A_\epsilon\) is a proper subexpression of \(B_\epsilon\).

13. \(I(\text{is-free-in}_{\epsilon \rightarrow \delta})\) is the total function \(f \in D_{\epsilon \rightarrow \delta}\) such that, for all constructions \(A_\epsilon, B_\epsilon \in D_\epsilon\), \(f(A_\epsilon)(B_\epsilon) = \top\) if \(A_\epsilon = \neg x_\alpha\) for some \(x_\alpha \in \mathcal{V}\), \(B_\epsilon = E(C_\beta)\) for some (eval-free) expression \(C_\beta\), and \(x_\alpha\) is free in \(C_\beta\).

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

### 3.3 General Models

An interpretation \(\mathcal{M} = (\{D_\alpha \mid \alpha \in T\}, I)\) is a general model for \(\mathcal{C}\) if there is a partial binary valuation function \(V^\mathcal{M}\) such that, for all assignments \(\varphi \in \text{assign}(\mathcal{M})\) and expressions \(D_\delta\), either \(V^\mathcal{M}(D_\delta) \in D_\delta\) or \(V^\mathcal{M}(D_\delta)\) is undefined and each of the following conditions is satisfied:

1. Let \(D_\delta \in \mathcal{V}\). Then \(V^\mathcal{M}_\varphi(D_\delta) = \varphi(D_\delta)\).
2. Let \(D_\delta \in \mathcal{C}\). Then \(V^\mathcal{M}_\varphi(D_\delta) = I(D_\delta)\).
3. Let \(D_\delta = \text{F}_{\alpha \rightarrow \beta} A_\alpha\). If \(V^\mathcal{M}_\varphi(\text{F}_{\alpha \rightarrow \beta})\) is defined, then \(V^\mathcal{M}_\varphi(M_\varphi(\text{F}_{\alpha \rightarrow \beta}))\) is defined at the argument \(V^\mathcal{M}_\varphi(A_\alpha)\), then \(V^\mathcal{M}_\varphi(D_\delta) = V^\mathcal{M}_\varphi(\text{F}_{\alpha \rightarrow \beta})(V^\mathcal{M}_\varphi(A_\alpha))\).

Otherwise, \(V^\mathcal{M}_\varphi(D_\delta) = \bot\) if \(\beta = \alpha\) and \(V^\mathcal{M}_\varphi(D_\delta)\) is undefined if \(\beta \neq \alpha\).

4. Let \(D_\delta = \lambda x_\alpha . B_\beta\). Then \(V^\mathcal{M}_\varphi(D_\delta)\) is the (partial or total) function \(f \in D_{\alpha \rightarrow \beta}\) such that, for each \(d \in D_\alpha\), \(f(d) = V^\mathcal{M}_\varphi\text{[}x_\alpha \mapsto d\text{]}(B_\beta)\) if \(V^\mathcal{M}_\varphi\text{[}x_\alpha \mapsto d\text{]}(B_\beta)\) is defined and \(f(d)\) is undefined if \(V^\mathcal{M}_\varphi\text{[}x_\alpha \mapsto d\text{]}(B_\beta)\) is undefined.

5. Let \(D_\delta = (A_\alpha B_\beta C_\gamma)\). If \(V^\mathcal{M}_\varphi(A_\alpha) = \top\) and \(V^\mathcal{M}_\varphi(B_\beta)\) is defined, then \(V^\mathcal{M}_\varphi(D_\delta) = V^\mathcal{M}_\varphi(B_\beta)\). If \(V^\mathcal{M}_\varphi(A_\alpha) = \bot\) and \(V^\mathcal{M}_\varphi(C_\gamma)\) is defined, then \(V^\mathcal{M}_\varphi(D_\delta) = V^\mathcal{M}_\varphi(C_\gamma)\). Otherwise, \(V^\mathcal{M}_\varphi(D_\delta)\) is undefined.

6. Let \(D_\delta = \neg x_\alpha\). Then \(V^\mathcal{M}_\varphi(D_\delta) = E(A_\alpha)\).

\footnote{We write \(V^\mathcal{M}_\varphi(D_\delta)\) instead of \(V^\mathcal{M}_\varphi(\varphi(D_\delta))\).}
7. Let \( D_\delta \) be \( ([A_\varphi])_\beta \). If \( V^M_\varphi(\text{is-expr}_\beta^{\varphi} A_\varphi) = \top \), then
\[
V^M_\varphi(D_\delta) = V^M_\varphi(\mathcal{E}^{-1}(V^M_\varphi(A_\varphi))).
\]
Otherwise, \( V^M_\varphi(D_\delta) = F \) if \( \beta = o \) and \( V^M_\varphi(D_\delta) \) is undefined if \( \beta \neq o \).

**Proposition 3.31** General models for \( \text{ctt uqe} \) exist.

**Proof.** The proof is similar to the proof of the analogous proposition in [9]. \( \square \)

Other theorems about the semantics of \( \text{ctt uqe} \) are the same or very similar to the theorems about the semantics of \( \text{ctt qe} \) given in [9].

Let \( M \) be a general model for \( \text{ctt uqe} \).
\( A_\alpha \) is valid in \( M \), written \( M \models A_\alpha \), if \( V^M_\varphi(B_\beta) \) holds for all \( \varphi \in \text{assign}(M) \).

Let \( T = (L, \Gamma) \) be a theory of \( \text{ctt uqe} \) and \( A_\alpha \) be a formula of \( T \). A generic model for \( T \) is a general model \( M \) for \( \text{ctt uqe} \) such that \( M \models A_\alpha \) for all \( A_\alpha \in \Gamma \).

4 Theory Morphisms

In this section we define a “semantic morphism” of \( \text{ctt uqe} \) 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 \( \text{ctt uqe} \), we introduce a more general notion of theory morphism that maps base types to semantically closed unary predicates that represent sets of values of the same type. This requires mapping expressions denoting functions to expressions denoting partial functions with domains restricted to semantically closed unary predicates.

For \( i = 1, 2 \), let \( T_i = (L_i, \Gamma_i) \) be a normal theory of \( \text{ctt uqe} \) where, for some \( B_i \subseteq B \) and \( C_i \subseteq C \), \( L_i \) is the set of all \( (B_i, C_i) \)-expressions. Also for \( i = 1, 2 \), let \( \mathcal{T}_i \) be the set of all \( B_i \)-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 unary predicates in \( L_2 \).

4.1 Translations

Define \( \tau \) to be the function that maps a unary predicate of type \( \alpha \to o \) to the type \( \alpha \). When \( p_{\alpha \to o} \) and \( q_{\beta \to o} \) are semantically closed unary predicates, let
\[
P_{\alpha \to o} \rightarrow q_{\beta \to o}
\]
be an abbreviation for the following semantically closed unary predicate of type $(\alpha \rightarrow \beta) \rightarrow \alpha$:

\[
\lambda f_{\alpha \rightarrow \beta}. \forall x_\alpha. f_{\alpha \rightarrow \beta} x_\alpha \neq \bot_\beta \supset (p_{\alpha \rightarrow o} x_\alpha \land q_{\beta \rightarrow o} (f_{\alpha \rightarrow \beta} x_\alpha)).
\]

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

\[
\tau(p_{\alpha \rightarrow o} \rightarrow q_{\beta \rightarrow o}) = \alpha \rightarrow \beta = \tau(p_{\alpha \rightarrow o}) \rightarrow \tau(q_{\beta \rightarrow o}).
\]

Given a total function $\mu : B_1 \rightarrow P_2$, let $\overline{\mu} : T_1 \rightarrow P_2$ be the canonical extension of $\mu$ that is defined inductively as follows:

1. If $\alpha \in B_1$, $\overline{\mu}(\alpha) = \mu(\alpha)$.
2. If $\alpha \rightarrow \beta \in T_1$, $\overline{\mu}(\alpha \rightarrow \beta) = \overline{\mu}(\alpha) \rightarrow \overline{\mu}(\beta)$.

It is easy to see that $\mu$ is well-defined and total.

A translation from $T_1$ to $T_2$ is a pair $\Phi = (\mu, \nu)$, where $\mu : B_1 \rightarrow P_2$ is total and $\nu : V_1 \cup C_1 \rightarrow V_2 \cup C_2$ is total and injective, such that:

1. $\mu(o) = \lambda x_\alpha . T_\alpha$.
2. $\mu(\epsilon) = \lambda x_\alpha . T_\alpha$.
3. For each $x_\alpha \in V_1$, $\nu(x_\alpha)$ is a variable in $V_2$ of type $\tau(\overline{\mu}(\alpha))$.
4. For each $c_\alpha \in C_1$, $\nu(c_\alpha)$ is a constant in $C_2$ of type $\tau(\overline{\mu}(\alpha))$.

**Remark 4.11** We overcome the Constant Interpretation Problem mentioned in section 1 by requiring $\nu$ to injectively map constants to constants. We will see in the next section that this requirement comes with a cost.

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 $\nu$ defined inductively as follows:

1. If $x_\alpha \in V_1$, $\overline{\nu}(x_\alpha) = \nu(x_\alpha)$.
2. If $c_\alpha \in C_1$, $\overline{\nu}(c_\alpha) = \nu(c_\alpha)$.
3. If $F_{\alpha \rightarrow \beta} A_\alpha \in L_1$, then $\overline{\nu}(F_{\alpha \rightarrow \beta} A_\alpha) = \overline{\nu}(F_{\alpha \rightarrow \beta}) \overline{\nu}(A_\alpha)$.
4. If $\lambda x_\alpha . B_\beta \in L_1$, then $\overline{\nu}(\lambda x_\alpha . B_\beta) = \lambda \overline{\nu}(x_\alpha) . (\text{if } (\overline{\mu}(\alpha) \overline{\nu}(x_\alpha)) \overline{\nu}(B_\beta) \bot \overline{\nu}(\beta))$.
5. If $(f_{A_\alpha B_\alpha C_\alpha}) \in L_1$, $\overline{\nu}(f_{A_\alpha B_\alpha C_\alpha}) = (\overline{\nu}(A_\alpha) \overline{\nu}(B_\alpha) \overline{\nu}(C_\alpha))$.
6. If $\neg A_\alpha \in L_1$, then $\overline{\nu}(\neg A_\alpha) = \overline{\nu}(A_\alpha)$.
7. If $[A_\alpha]_{B_\beta} \in L_1$, then $\overline{\nu}([A_\alpha]_{B_\beta}) = \overline{\nu}(A_\alpha)_{\overline{\nu}(B_\beta)}$.

**Lemma 4.12**

1. $\overline{\nu}$ is well-defined, total, and injective.
2. If $A_\alpha \in L_1$, then $\overline{\nu}(A_\alpha)$ 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. □
A formula in \( L_2 \) is an obligation of \( \Phi \) if it is one of the following formulas:

1. \( \exists x_\tau(\mu(\alpha)) \cdot \mu(\alpha) x_\tau(\mu(\alpha)) \) where \( \alpha \in B_1 \).
2. \( \mathfrak{m}(\alpha) \nu(c_\alpha) \) where \( c_\alpha \in C_1 \).
3. \( \nu(=_{\alpha \rightarrow o \rightarrow o}) = \lambda x_{\alpha'} \cdot \lambda y_{\alpha'} \cdot (\mathfrak{m}(\alpha) x_{\alpha'} \land \mathfrak{m}(\alpha) y_{\alpha'}) (x_{\alpha'} =_{\alpha' \rightarrow o} y_{\alpha'}) \) where \( \alpha \in T_1 \) and \( \alpha' = \tau(\mathfrak{m}(\alpha)) \).
4. \( \nu(\langle(\alpha \rightarrow o) x_{\alpha' \rightarrow o}) = \lambda x_{\alpha' \rightarrow o} \cdot (\mathfrak{m}(\alpha \rightarrow o) x_{\alpha' \rightarrow o}) (\langle(\alpha' \rightarrow o) x_{\alpha' \rightarrow o} \land \alpha') \) where \( \alpha \in T_1 \) with \( \alpha \neq o \) and \( \alpha' = \tau(\mathfrak{m}(\alpha)) \).
5. \( \nu(c_\alpha) = c_\alpha \) where \( c_\alpha \) is is-var\_{\alpha' \rightarrow o}, is-con\_{\epsilon \rightarrow o}, app\_{\epsilon \rightarrow \epsilon}, \) cond\_{\epsilon \rightarrow \epsilon \rightarrow \epsilon}, quo\_{\epsilon \rightarrow \epsilon \rightarrow o}, \) or is-free-in\_{\epsilon \rightarrow \epsilon \rightarrow o}.
6. \( \nu(c_\beta) = c_\beta \mathfrak{m}(\beta) \) where \( c_\beta \) is is-var\_{\epsilon' \rightarrow o}, is-con\_{\epsilon' \rightarrow o}, or is-expr\_{\epsilon' \rightarrow o} and \( \beta \in T_1 \).
7. \( \mathfrak{M}(A_\alpha) \) where \( A_\alpha \in I_1 \).

Notice that each obligation of \( \Phi \) is semantically closed.

### 4.2 Semantic Morphisms

A semantic morphism from \( T_1 \) to \( T_2 \) is a translation \((\mu, \nu)\) from \( T_1 \) to \( T_2 \) such that \( T_1 \models A_\alpha \) implies \( T_2 \models \mathfrak{M}(A_\alpha) \) for all semantically closed formulas \( A_\alpha \) of \( T_1 \).

(A syntactic morphism from \( T_1 \) to \( T_2 \) would be a translation \((\mu, \nu)\) from \( T_1 \) to \( T_2 \) such that \( T_1 \vdash P \alpha_{\mathfrak{m}} A_\alpha \) implies \( T_2 \vdash P \mathfrak{M}(A_\alpha) \) for all semantically closed formulas \( A_\alpha \) of \( T_1 \) where \( P \) is some proof system for \( \text{CTT}_{\text{tue}} \).) We will prove a theorem (called the Semantic Morphism Theorem) that gives a sufficient condition for a translation to be a semantic morphism.

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

For each \( \alpha \in T_1 \), define \( D^2_\alpha(\mathfrak{m}(\alpha)) \subseteq D^2_{\mathfrak{m}(\alpha)} \) as follows:

1. \( D^2_{\mathfrak{m}(\alpha)} = D^2_\mu = D^2_\mu = \{ T, F \} \).
2. \( D^2_{\mathfrak{m}(\alpha)} = D^2_
u \) where

\[ \{ d \in D^2_\nu | d = V^M_\varphi(\mathfrak{M}(A_\alpha)) \} \]

for some construction \( A_\alpha \in L_1 \)

where \( \varphi \) is any member of \( \text{assign}(\mathcal{M}_2) \).

3. If \( \alpha \in T_1 \setminus \{ o, e \} \), \( D^2_{\mathfrak{m}(\alpha)} = \)

\[ \{ d \in D^2_\nu(\mathfrak{m}(\alpha)) | V^M_\varphi(\mathfrak{M}(\alpha))(d) = T \} \]

where \( \varphi \) is any member of \( \text{assign}(\mathcal{M}_2) \).

For each \( \alpha \in T_1 \), define \( D^1_\alpha \) inductively as follows:

1. \( D^1_\mu = \{ T, F \} \).
2. \( D^1_\nu \) is the set of constructions of \( \text{CTT}_{\text{tue}} \).
3. If \( \alpha \in B_1 \setminus \{ o, e \} \), \( D^1_\alpha = D^2_\nu(\mathfrak{m}(\alpha)) \).
4. If \( \alpha \to \beta \in \mathcal{T}_1 \), then \( \overrightarrow{D}_{\alpha \to \beta}^1 \) is the set of all \emph{total} functions from \( \overrightarrow{D}_\alpha^1 \) to \( \overrightarrow{D}_\beta^1 \) if \( \beta = o \) and the set of all \emph{partial} and \emph{total} functions from \( \overrightarrow{D}_\alpha^1 \) to \( \overrightarrow{D}_\beta^1 \) if \( \beta \neq o \).

For each \( \alpha \in \mathcal{T}_1 \), define \( \rho_\alpha : \overrightarrow{D}_\mathcal{T}^2(\mathfrak{c}(\alpha)) \to \overrightarrow{D}_\alpha^1 \) inductively as follows:

1. If \( d \in \mathcal{D}_1 \), \( \rho_\alpha(d) \) is the unique construction \( \mathbf{A}_\alpha \) such that \( \mathfrak{c}(\mathbf{A}_\alpha) = d \).
2. If \( \alpha \in B_1 \setminus \{\epsilon\} \) and \( d \in \mathcal{D}_\mathcal{T}^2(\mathfrak{c}(\alpha)) \), \( \rho_\alpha(d) = d \).
3. If \( \alpha \to \beta \in \mathcal{T}_1 \) and \( f \in \overrightarrow{D}_\mathcal{T}^2(\mathfrak{c}(\alpha \to \beta)) \), \( \rho_\alpha \circ \beta(f) \) is the unique function \( g \in \overrightarrow{D}_{\alpha \to \beta}^1 \) such that, for all \( d \in \mathcal{D}_\mathcal{T}^2(\mathfrak{c}(\alpha)) \), 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.

\textbf{Lemma 4.21} If \( \alpha \in \mathcal{T}_1 \), \( \rho_\alpha : \overrightarrow{D}_\mathcal{T}^2(\mathfrak{c}(\alpha)) \to \overrightarrow{D}_\alpha^1 \) is well defined, total, and injective.

\textit{Proof.} This lemma is proved by induction on \( \alpha \in \mathcal{T}_1 \). \( \rho_\alpha \) is well defined since \( V_\phi^M \) is identity function on constructions and \( \nu \) is injective.

For each \( \alpha \in \mathcal{T}_1 \), define \( D_\alpha^1 \subseteq \overrightarrow{D}_\alpha^1 \) as follows:

1. If \( \alpha \in B_1 \), \( D_\alpha^1 = \overrightarrow{D}_\alpha^1 \).
2. If \( \alpha \to \beta \in \mathcal{T}_1 \) and \( D_\alpha^1 \to \beta \) is the range of \( \rho_\alpha \to \beta \).
3. If \( \alpha \in B \setminus B_1 \), \( D_\alpha^1 \) is any nonempty set.
4. If \( \alpha \to \beta \in \mathcal{T} \setminus \mathcal{T}_1 \), \( D_\alpha^1 \) is the set of all \emph{total} functions from \( D_\alpha^1 \) to \( D_\beta^1 \) if \( \beta = o \) and the set of all \emph{partial} and \emph{total} functions from \( D_\alpha^1 \) to \( D_\beta^1 \) if \( \beta \neq o \).

For \( c_\alpha \in C_1 \), define \( I_1(c_\alpha) = \rho_\alpha(V_\phi^M(\mathfrak{c}(c_\alpha))) \) where \( \phi \) is any member of \( \text{assign}(\mathcal{M}_2) \). Finally, define \( \mathcal{M}_1 = \{(D_\alpha^1 \mid \alpha \in \mathcal{T}_1), I_1\} \).

\textbf{Lemma 4.22} Suppose each obligation of \( \Phi \) is valid in \( \mathcal{M}_2 \). Then \( \mathcal{M}_1 \) is a \emph{general model} for \( \mathcal{T}_2 \).

\textit{Proof.} By the first group of obligations of \( \Phi \), \( D_\alpha^1 \) is nonempty for all \( \alpha \in B_1 \), and so \( \{D_\alpha^1 \mid \alpha \in \mathcal{T}\} \) is a frame of \( \text{CTT}_\text{uqe} \). By the second to sixth groups of obligations of \( \Phi \), \( \mathcal{M}_1 \) is an interpretation of \( \text{CTT}_\text{uqe} \). For all \( A_\alpha \in \mathcal{L}_1 \) and \( \phi \in \text{assign}(\mathcal{M}_1) \), define \( V_\phi^{M_1}(A_\alpha) \) as follows:

\[
(*) \quad V_\phi^{M_1}(A_\alpha) = \rho_\alpha(V_\phi^{M_2}(\mathfrak{c}(A_\alpha))) \quad \text{if} \quad V_\phi^{M_2}(\mathfrak{c}(A_\alpha)) \quad \text{is defined and} \quad V_\phi^{M_1}(A_\alpha) \quad \text{is undefined otherwise,}
\]

where \( \mathfrak{c}(\phi) \) is any \( \psi \in \text{assign}(\mathcal{M}_2) \) such that, for all \( x_\beta \in \mathcal{V}_1 \), \( \rho_\beta(\psi(\mathfrak{c}(x_\beta))) = \phi(x_\beta) \). This definition of \( V_\phi^{M_1} \) 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 \( \text{CTT}_\text{uqe} \). Then \( (*) \) implies

\[
(**) \quad \mathcal{M}_1 \models A_\alpha \iff \mathcal{M}_2 \models \mathfrak{c}(A_\alpha)
\]

for all semantically closed formulas \( A_\alpha \in \mathcal{L}_1 \). By the seventh group of obligations of \( \Phi \), \( \mathcal{M}_2 \models \mathfrak{c}(A_\alpha) \) for all \( A_\alpha \in \Gamma_1 \), and thus \( \mathcal{M}_1 \) is a general model for \( \mathcal{T}_1 \) by \( (**) \).
Theorem 4.23 (Semantic Morphism Theorem) Let $T_1$ and $T_2$ be normal theories and $\Phi$ be a translation from $T_1$ to $T_2$. Suppose each obligation of $\Phi$ is valid in $T_2$. Then $\Phi$ is a semantic morphism of 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 $\Phi$ is valid in $T_2$. Let $A_0 \in L_1$ be semantically closed and valid in $T_1$. We must show that $\pi(A_0)$ is valid in every general model for $T_2$. Let $M_2$ be a general model for $T_1$. (We are done if there are no general models for $T_2$.) Let $M_1$ be extracted from $M_2$ as above. Obviously, each obligation of $\Phi$ is valid in $M_2$, and so $M_1$ is a general model for $T_1$ by Lemma 4.22. Therefore, $M_1 \models A_0$, and so $M_2 \models \pi(A_0)$ by $(\ast \ast)$ in the proof of Lemma 4.22.

Theorem 4.24 (Relative Satisfiability) Let $T_1$ and $T_2$ be normal theories and suppose $\Phi$ 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 $T_2$, $M_2$ be a general model for $T_1$, and $M_1$ be extracted from $M_2$ as above. Since $\Phi$ is a semantic morphism, each of its obligations is valid in $T_2$. Hence, $M_1$ is a general model for $T_1$ by Lemma 4.22.

5 Examples

We will illustrate the theory morphism machinery of $\text{ctt}_{\text{uqe}}$ with two simple examples. Let $C_{\text{log}} \subseteq C$ be the set of logical constants of $\text{ctt}_{\text{uqe}}$.

5.1 Example 1: Monoid with Left and Right Identity Elements

Define $M = (L_M, \Gamma_M)$ to be the usual theory of an abstract monoid where:

1. $B_M = \{0, e, \iota\}$.
2. $C_M = C_{\text{log}} \cup \{e_i, \ast_\iota \to \iota \to \iota\}$. (\ast_\iota \to \iota \to \iota is written as an infix operator.)
3. $L_M$ is the set of $(B_M, C_M)$ expressions.
4. $V_M$ is the set of variables in $L_M$.
5. $\Gamma_M$ contains the following axioms:
   a. $\forall x_i . \forall y_i . \forall z_i . x_i \ast_\iota \to \iota \to \iota (y_i \ast_\iota \to \iota \to \iota z_i) = (x_i \ast_\iota \to \iota \to \iota y_i) \ast_\iota \to \iota \to \iota z_i$.
   b. $\forall x_i . e_i \ast_\iota \to \iota \to \iota x_i = x_i$.
   c. $\forall x_i . x_i \ast_\iota \to \iota \to \iota e_i = x_i$.

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

1. $B_{M'} = B_M$.
2. $C_{M'} = C_{\text{log}} \cup \{e_i^{\text{left}}, e_i^{\text{right}}, \ast_\iota \to \iota \to \iota\}$. (\ast_\iota \to \iota \to \iota is written as an infix operator.)
3. $L_{M'}$ is the set of $(B_{M'}, C_{M'})$ expressions.
4. $V_{M'} = V_M$.
5. $\Gamma_M'$ contains the following axioms:
   a. $\forall x_i \forall y_i \forall z_i . x_i *_{i\rightarrow i\rightarrow i} (y_i *_{i\rightarrow i\rightarrow i} z_i) = (x_i *_{i\rightarrow i\rightarrow i} y_i) *_{i\rightarrow i\rightarrow i} z_i$.
   b. $\forall x_i . e_{i}^{\left\{\text{left}\right\}} *_{i\rightarrow i\rightarrow i} x_i = x_i$.
   c. $\forall x_i . x_i *_{i\rightarrow i\rightarrow i} e_{i}^{\left\{\text{right}\right\}} = x_i$.

   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 $\nu$ 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 $e'$ and the new axiom $e' = e'$.

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

   1. $\mu(i) = \lambda x_i . T_o$.
   2. $\nu$ is the identity function on $V_{M'} \cup C_{\text{log}} \cup \{ *_{i\rightarrow i\rightarrow i} \}$.
   3. $\nu(e_{i}^{\left\{\text{left}\right\}}) = e_i$.
   4. $\nu(e_{i}^{\left\{\text{right}\right\}}) = e'_i$.

   It is easy to see that $\Phi$ is a semantic interpretation by Theorem 4.23.

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 expressed using a definition of a theory morphism that maps base types to types. However, it can be expressed using the notion of a semantic morphism we have defined.

The desired translation interprets the type $i$ as the set $\{ e_i \}$ and the constants denoting functions involving $i$ as functions in which the domain of $i$ is replaced by $\{ e_i \}$. This is not possible since the mapping $\nu$ 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 $\mu$ are follows:

   1. For $\alpha \in \{ a, e \}$, $\mu(\alpha) = \lambda x_\alpha . T_o$.
   2. $\mu(i) = \lambda x_i . x_i = e_i$.

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

   1. $=_{\alpha\rightarrow\alpha\rightarrow\alpha} = \lambda x_\alpha' \cdot \lambda y_\alpha' . (if \ (\overline{p}(\alpha) \ x_\alpha' \ \land \overline{p}(\alpha) \ y_\alpha') \ (x_\alpha' =_{\alpha'\rightarrow\alpha'\rightarrow\alpha} y_\alpha') =_{\alpha})$
   
   where $\alpha \in \mathcal{T}$ contains $i$ and $\alpha' = \tau(\overline{p}(\alpha))$.

   Technically, $e'_i$ is a constant chosen from $C \setminus C_M$. There is no harm is assuming that such a constant already exists in $C$.
2. \( \iota'_{(\alpha \to o) \to o} = \lambda x_{\alpha' \to o} \cdot (\iota(\alpha \to o) x_{\alpha' \to o}) (\iota_{(\alpha' \to o) \to o} x_{\alpha' \to o}) \perp_{\alpha'} \) where \( \alpha \in \mathcal{T} \) contains \( \iota \) and \( \alpha' = \tau(\iota(\alpha)) \).

3. \( \eta'_{\iota \to o} = \lambda x_{\iota} \cdot \lambda y_{\iota} \cdot (x_{\iota} = e_{\iota} \land y_{\iota} = e_{\iota}) e_{\iota} \perp_{\iota} \).

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

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

It is easy to see that \( \Phi \) is a semantic interpretation by Theorem 4.23.

6 Conclusion

\( \text{CTT}_qe \) 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 \( \text{CTT}_uqe \), a variant of \( \text{CTT}_qe \) that admits undefined expressions, partial functions, and multiple base types of individuals, (2) defined a notion of a theory morphism in \( \text{CTT}_uqe \), and (3) given two simple examples that illustrate the use of theory morphisms in \( \text{CTT}_uqe \). The theory morphisms of \( \text{CTT}_uqe \) overcome the Constant Interpretation Problem discussed in section 1 by requiring constants to be injectively mapped to constants. Since \( \text{CTT}_qe \) admits partial functions, \( \text{CTT}_uqe \) 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 formally defined as \( \text{CTT}_uqe \) 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 \( \text{CTT}_uqe \) 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 \( \text{CTT}_uqe \) 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 \((\mu, \nu)\) where \( \mu \) maps base types to either types or semantically closed unary predicates, \( \nu \) maps constants to expressions that need not be constants, and \( \nu \) 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 \( \text{CTT}_qe \) 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. Guttmann, 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*, pages 22–27, June 2014.