Global Reflection

Jacques Carette William M. Farmer
Johanna Schwartzentruber

today

Abstract

1 Introduction

2 Global Reflection

3 CTT$_{\mathrm{qe}}$

The syntax, semantics, and proof system of CTT$_{\mathrm{qe}}$ are defined in [3]. Here we will only introduce the definitions and results of that are key to understanding how HOL Light QE implements CTT$_{\mathrm{qe}}$. The reader is encouraged to consult [3] when additional details are required.

3.1 Syntax

CTT$_{\mathrm{qe}}$ has the same machinery as $\mathcal{Q}_0$ plus an inductive type ϵ of syntactic values, a partial quotation operator, and a typed evaluation operator.

A *type* of CTT$_{\mathrm{qe}}$ is defined inductively by the following formation rules:

(1) *Type of individuals*: ι is a type.

(2) *Type of truth values*: o is a type.

(3) *Type of constructions*: ϵ is a type.

(4) *Function type*: If α and β are types, then $(\alpha \rightarrow \beta)$ is a type.

Let $\mathcal{T}$ denote the set of types of CTT$_{\mathrm{qe}}$. A *typed symbol* is a symbol with a subscript from $\mathcal{T}$. Let $\mathcal{V}$ be a set of typed symbols such that, for each $\alpha \in \mathcal{T}$, $\mathcal{V}$ contains denumerably many typed symbols with subscript α. A *variable of type* α of CTT$_{\mathrm{qe}}$ is a member of $\mathcal{V}$ with subscript α. $\mathbf{x}_\alpha, \mathbf{y}_\alpha, \mathbf{z}_\alpha, \ldots$ are syntactic variables ranging over variables of type α. Let $\mathcal{C}$ be a set of typed symbols disjoint from $\mathcal{V}$. A *constant of type* α of CTT$_{\mathrm{qe}}$ is a member of $\mathcal{C}$ with subscript α.

$\mathbf{c}_\alpha, \mathbf{d}_\alpha, \ldots$ are syntactic variables ranging over constants of type α. $\mathcal{C}$ contains a set of *logical constants* that include $\mathsf{app}_{\epsilon \rightarrow \epsilon \rightarrow \epsilon}$, $\mathsf{abs}_{\epsilon \rightarrow \epsilon \rightarrow \epsilon}$, and $\mathsf{quo}_{\epsilon \rightarrow \epsilon}$.

An *expression of type* α of $\textsc{ctt}_{\rm qe}$ is 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 five 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) *Quotation*: $\ulcorner \mathbf{A}_\alpha \urcorner$ is an expression of type ϵ if $\mathbf{A}_\alpha$ is eval-free.

(6) *Evaluation*: $[\![\mathbf{A}_\epsilon]\!]_{\mathbf{B}_\beta}$ is an expression of type β.

The sole purpose of the second component $\mathbf{B}_\beta$ in an evaluation $[\![\mathbf{A}_\epsilon]\!]_{\mathbf{B}_\beta}$ is to establish the type of the evaluation; we will thus write $[\![\mathbf{A}_\epsilon]\!]_{\mathbf{B}_\beta}$ as $[\![\mathbf{A}_\epsilon]\!]_\beta$.

A *construction* of $\textsc{ctt}_{\rm qe}$ is an expression of type ϵ defined inductively by:

(1) $\ulcorner \mathbf{x}_\alpha \urcorner$ is a construction.

(2) $\ulcorner \mathbf{c}_\alpha \urcorner$ is a construction.

(3) If $\mathbf{A}_\epsilon$ and $\mathbf{B}_\epsilon$ are constructions, then $\mathsf{app}_{\epsilon \rightarrow \epsilon \rightarrow \epsilon}\, \mathbf{A}_\epsilon\, \mathbf{B}_\epsilon$, $\mathsf{abs}_{\epsilon \rightarrow \epsilon \rightarrow \epsilon}\, \mathbf{A}_\epsilon\, \mathbf{B}_\epsilon$, and $\mathsf{quo}_{\epsilon \rightarrow \epsilon}\, \mathbf{A}_\epsilon$ are constructions.

The set of constructions is thus an inductive type whose base elements are quotations of variables and constants, and whose constructors are $\mathsf{app}_{\epsilon \rightarrow \epsilon \rightarrow \epsilon}$, $\mathsf{abs}_{\epsilon \rightarrow \epsilon \rightarrow \epsilon}$, and $\mathsf{quo}_{\epsilon \rightarrow \epsilon}$. As we will see shortly, constructions serve as syntactic values.

Let $\mathcal{E}$ be the function mapping eval-free expressions to constructions 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) $\mathcal{E}(\mathbf{F}_{\alpha \rightarrow \beta}\, \mathbf{A}_\alpha) = \mathsf{app}_{\epsilon \rightarrow \epsilon \rightarrow \epsilon}\, \mathcal{E}(\mathbf{F}_{\alpha \rightarrow \beta})\, \mathcal{E}(\mathbf{A}_\alpha)$.

(4) $\mathcal{E}(\lambda\, \mathbf{x}_\alpha \,.\, \mathbf{B}_\beta) = \mathsf{abs}_{\epsilon \rightarrow \epsilon \rightarrow \epsilon}\, \mathcal{E}(\mathbf{x}_\alpha)\, \mathcal{E}(\mathbf{B}_\beta)$.

(5) $\mathcal{E}(\ulcorner \mathbf{A}_\alpha \urcorner) = \mathsf{quo}_{\epsilon \rightarrow \epsilon}\, \mathcal{E}(\mathbf{A}_\alpha)$.

When $\mathbf{A}_\alpha$ is eval-free, $\mathcal{E}(\mathbf{A}_\alpha)$ is the unique construction that represents the syntax tree of $\mathbf{A}_\alpha$. That is, $\mathcal{E}(\mathbf{A}_\alpha)$ is a syntactic value that represents how $\mathbf{A}_\alpha$ is syntactically constructed. For every eval-free expression, there is a construction that represents its syntax tree, but not every construction represents the syntax tree of an eval-free expression. For example, $\mathsf{app}_{\epsilon\rightarrow\epsilon\rightarrow\epsilon}\ulcorner\mathbf{x}_\alpha\urcorner\ulcorner\mathbf{x}_\alpha\urcorner$ represents the syntax tree of $(\mathbf{x}_\alpha\,\mathbf{x}_\alpha)$ which is not an expression of $\textsc{ctt}_{\rm qe}$ since the types are mismatched. A construction is *proper* if it is in the range of $\mathcal{E}$, i.e., it represents the syntax tree of an eval-free expression.

The purpose of $\mathcal{E}$ is to define the semantics of quotation: the meaning of $\ulcorner\mathbf{A}_\alpha\urcorner$ is $\mathcal{E}(\mathbf{A}_\alpha)$.

3.2 Semantics

The semantics of $\textsc{ctt}_{\rm qe}$ is based on Henkin-style general models [7]. An expression $\mathbf{A}_\epsilon$ of type ϵ denotes a construction, and when $\mathbf{A}_\epsilon$ is a construction, it denotes itself. The semantics of the quotation and evaluation operators are defined so that the following two theorems hold:

Theorem 3.1 (Law of Quotation) $\ulcorner\boldsymbol{A}_\alpha\urcorner = \mathcal{E}(\boldsymbol{A}_\alpha)$ *is valid in* $\textsc{ctt}_{\rm qe}$.

Corollary 3.2 $\ulcorner\boldsymbol{A}_\alpha\urcorner = \ulcorner\boldsymbol{B}_\alpha\urcorner$ *is valid in* $\textsc{ctt}_{\rm qe}$ *iff* $\boldsymbol{A}_\alpha$ *and* $\boldsymbol{B}_\alpha$ *are identical expressions.*

Theorem 3.3 (Law of Disquotation) $[\![\ulcorner\boldsymbol{A}_\alpha\urcorner]\!]_\alpha = \boldsymbol{A}_\alpha$ *is valid in* $\textsc{ctt}_{\rm qe}$.

Remark 3.4 Notice that this is not the full Law of Disquotation, since only eval-free expressions can be quoted. As a result of this restriction, the liar paradox is not expressible in $\textsc{ctt}_{\rm qe}$ and the Evaluation Problem mentioned above is effectively solved.

3.3 Quasiquotation

Quasiquotation is a parameterized form of quotation in which the parameters serve as holes in a quotation that are filled with expressions that denote syntactic values. It is a very powerful syntactic device for specifying expressions and defining macros. Quasiquotation was introduced by Willard Van Orman Quine in 1940 in the first version of his book *Mathematical Logic* [8]. It has been extensively employed in the Lisp family of programming languages [2][1], and from there to other families of programming languages, most notably the ML family.

In $\textsc{ctt}_{\rm qe}$, constructing a large quotation from smaller quotations can be tedious because it requires many applications of the syntax constructors $\mathsf{app}_{\epsilon\rightarrow\epsilon\rightarrow\epsilon}$, $\mathsf{abs}_{\epsilon\rightarrow\epsilon\rightarrow\epsilon}$, and $\mathsf{quo}_{\epsilon\rightarrow\epsilon}$. Quasiquotation alleviates this problem. It can be defined straightforwardly in $\textsc{ctt}_{\rm qe}$. However, quasiquotation is not part of the official syntax of $\textsc{ctt}_{\rm qe}$; it is just a notational device used to write $\textsc{ctt}_{\rm qe}$ expressions in a compact form.

As an example, consider $\ulcorner\neg(\mathbf{A}_o \wedge \lfloor\mathbf{B}_\epsilon\rfloor)\urcorner$. Here $\lfloor\mathbf{B}_\epsilon\rfloor$ is a *hole* or *antiquotation.* Assume that $\mathbf{A}_o$ contains no holes. $\ulcorner\neg(\mathbf{A}_o\wedge\lfloor\mathbf{B}_\epsilon\rfloor)\urcorner$ is then an abbreviation

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

for the verbose expression

$$\mathsf{app}_{\epsilon\to\epsilon\to\epsilon} \ulcorner\neg_{o\to o}\urcorner\,(\mathsf{app}_{\epsilon\to\epsilon\to\epsilon}\,(\mathsf{app}_{\epsilon\to\epsilon\to\epsilon}\ulcorner\wedge_{o\to o\to o}\urcorner\ulcorner\mathbf{A}_o\urcorner)\,\mathbf{B}_\epsilon).$$

$\ulcorner\neg(\mathbf{A}_o \wedge \lfloor\mathbf{B}_\epsilon\rfloor)\urcorner$ represents the syntax tree of a negated conjunction in which the part of the tree corresponding to the second conjunct is replaced by the syntax tree represented by $\mathbf{B}_\epsilon$. If $\mathbf{B}_\epsilon$ is a quotation $\ulcorner\mathbf{C}_o\urcorner$, then the quasiquotation $\ulcorner\neg(\mathbf{A}_o \wedge \lfloor\ulcorner\mathbf{C}_o\urcorner\rfloor)\urcorner$ is *equivalent* to the quotation $\ulcorner\neg(\mathbf{A}_o \wedge \mathbf{C}_o)\urcorner$.

3.4 Proof System

The proof system for $\textsc{ctt}_{\mathrm{qe}}$ consists of the axioms for $\mathcal{Q}_0$, the single rule of inference for $\mathcal{Q}_0$, and additional axioms [3, B1–B13] that define the logical constants of $\textsc{ctt}_{\mathrm{qe}}$ (B1–B4, B5, B7), specify ϵ as an inductive type (B4, B6), state the properties of quotation and evaluation (B8, B10), and extend the rules for beta-reduction (B9, B11–13). We prove in [3] that this proof system is sound for all formulas and complete for eval-free formulas.

The axioms that express the properties of quotation and evaluation are:

B8 (Properties of Quotation)

(1) $\ulcorner\mathbf{F}_{\alpha\to\beta}\,\mathbf{A}_\alpha\urcorner = \mathsf{app}_{\epsilon\to\epsilon\to\epsilon}\ulcorner\mathbf{F}_{\alpha\to\beta}\urcorner\ulcorner\mathbf{A}_\alpha\urcorner$.

(2) $\ulcorner\lambda\,\mathbf{x}_\alpha\,.\,\mathbf{B}_\beta\urcorner = \mathsf{abs}_{\epsilon\to\epsilon\to\epsilon}\ulcorner\mathbf{x}_\alpha\urcorner\ulcorner\mathbf{B}_\beta\urcorner$.

(3) $\ulcorner\ulcorner\mathbf{A}_\alpha\urcorner\urcorner = \mathsf{quo}_{\epsilon\to\epsilon}\ulcorner\mathbf{A}_\alpha\urcorner$.

B10 (Properties of Evaluation)

(1) $[\![\ulcorner\mathbf{x}_\alpha\urcorner]\!]_\alpha = \mathbf{x}_\alpha$.

(2) $[\![\ulcorner\mathbf{c}_\alpha\urcorner]\!]_\alpha = \mathbf{c}_\alpha$.

(3) $(\mathsf{is\text{-}expr}^{\alpha\to\beta}_{\epsilon\to o}\,\mathbf{A}_\epsilon \wedge \mathsf{is\text{-}expr}^{\alpha}_{\epsilon\to o}\,\mathbf{B}_\epsilon) \supset [\![\mathsf{app}_{\epsilon\to\epsilon\to\epsilon}\,\mathbf{A}_\epsilon\,\mathbf{B}_\epsilon]\!]_\beta = [\![\mathbf{A}_\epsilon]\!]_{\alpha\to\beta}\,[\![\mathbf{B}_\epsilon]\!]_\alpha$.

(4) $(\mathsf{is\text{-}expr}^{\beta}_{\epsilon\to o}\,\mathbf{A}_\epsilon \wedge \neg(\mathsf{is\text{-}free\text{-}in}_{\epsilon\to\epsilon\to o}\ulcorner\mathbf{x}_\alpha\urcorner\ulcorner\mathbf{A}_\epsilon\urcorner)) \supset$
$[\![\mathsf{abs}_{\epsilon\to\epsilon\to\epsilon}\ulcorner\mathbf{x}_\alpha\urcorner\,\mathbf{A}_\epsilon]\!]_{\alpha\to\beta} = \lambda\,\mathbf{x}_\alpha\,.\,[\![\mathbf{A}_\epsilon]\!]_\beta$.

(5) $\mathsf{is\text{-}expr}^{\epsilon}_{\epsilon\to o}\,\mathbf{A}_\epsilon \supset [\![\mathsf{quo}_{\epsilon\to\epsilon}\,\mathbf{A}_\epsilon]\!]_\epsilon = \mathbf{A}_\epsilon$.

The axioms for extending the rules for beta-reduction are:

B9 (Beta-Reduction for Quotations)

$$(\lambda\, \mathbf{x}_\alpha \,.\, \ulcorner \mathbf{B}_\beta \urcorner)\, \mathbf{A}_\alpha = \ulcorner \mathbf{B}_\beta \urcorner.$$

B11 (Beta-Reduction for Evaluations)

(1) $(\lambda\, \mathbf{x}_\alpha \,.\, [\![\mathbf{B}_\epsilon]\!]_\beta)\, \mathbf{x}_\alpha = [\![\mathbf{B}_\epsilon]\!]_\beta$.

(2) $(\mathsf{is\text{-}expr}^{\beta}_{\epsilon\to o}\, ((\lambda\, \mathbf{x}_\alpha \,.\, \mathbf{B}_\epsilon)\, \mathbf{A}_\alpha) \wedge \neg(\mathsf{is\text{-}free\text{-}in}_{\epsilon\to\epsilon\to o}\, \ulcorner \mathbf{x}_\alpha \urcorner\, ((\lambda\, \mathbf{x}_\alpha \,.\, \mathbf{B}_\epsilon)\, \mathbf{A}_\alpha))) \supset$
$(\lambda\, \mathbf{x}_\alpha \,.\, [\![\mathbf{B}_\epsilon]\!]_\beta)\, \mathbf{A}_\alpha = [\![(\lambda\, \mathbf{x}_\alpha \,.\, \mathbf{B}_\epsilon)\, \mathbf{A}_\alpha]\!]_\beta$.

B12 ("Not Free In" means "Not Effective In")

$\neg\mathsf{IS\text{-}EFFECTIVE\text{-}IN}(\mathbf{x}_\alpha, \mathbf{B}_\beta)$
where $\mathbf{B}_\beta$ is eval-free and $\mathbf{x}_\alpha$ is not free in $\mathbf{B}_\beta$.

B13 (Beta-Reduction for Function Abstractions)

$(\neg\mathsf{IS\text{-}EFFECTIVE\text{-}IN}(\mathbf{y}_\beta, \mathbf{A}_\alpha) \vee \neg\mathsf{IS\text{-}EFFECTIVE\text{-}IN}(\mathbf{x}_\alpha, \mathbf{B}_\gamma)) \supset$
$(\lambda\, \mathbf{x}_\alpha \,.\, \lambda\, \mathbf{y}_\beta \,.\, \mathbf{B}_\gamma)\, \mathbf{A}_\alpha = \lambda\, \mathbf{y}_\beta \,.\, ((\lambda\, \mathbf{x}_\alpha \,.\, \mathbf{B}_\gamma)\, \mathbf{A}_\alpha)$
where $\mathbf{x}_\alpha$ and $\mathbf{y}_\beta$ are distinct.

Substitution is performed using the properties of beta-reduction as Andrews does in the proof system for $\mathcal{Q}_0$ [1, p. 213]. The following three beta-reduction cases require discussion:

(1) $(\lambda\, \mathbf{x}_\alpha \,.\, \lambda\, \mathbf{y}_\beta \,.\, \mathbf{B}_\gamma)\, \mathbf{A}_\alpha$ where $\mathbf{x}_\alpha$ and $\mathbf{y}_\beta$ are distinct.

(2) $(\lambda\, \mathbf{x}_\alpha \,.\, \ulcorner \mathbf{B}_\beta \urcorner)\, \mathbf{A}_\alpha$.

(3) $(\lambda\, \mathbf{x}_\alpha \,.\, [\![\mathbf{B}_\epsilon]\!]_\beta)\, \mathbf{A}_\alpha$.

The first case can normally be reduced when either (1) $\mathbf{y}_\beta$ is not free in $\mathbf{A}_\alpha$ or (2) $\mathbf{x}_\alpha$ is not free in $\mathbf{B}_\gamma$. However, due to the Variable Problem mentioned before, it is only possible to syntactically check whether a "variable is not free in an expression" when the expression is eval-free. Our solution is to replace the syntactic notion of "a variable is free in an expression" by the semantic notion of "a variable is effective in an expression" when the expression is not necessarily eval-free, and use Axiom B13 to perform the beta-reduction.

"$\mathbf{x}_\alpha$ is effective in $\mathbf{B}_\beta$" means the value of $\mathbf{B}_\beta$ depends on the value of $\mathbf{x}_\alpha$. Clearly, if $\mathbf{B}_\beta$ is eval-free, "$\mathbf{x}_\alpha$ is effective in $\mathbf{B}_\beta$" implies "$\mathbf{x}_\alpha$ is free in $\mathbf{B}_\beta$". However, "$\mathbf{x}_\alpha$ is effective in $\mathbf{B}_\beta$" is a refinement of "$\mathbf{x}_\alpha$ is free in $\mathbf{B}_\beta$" on eval-free expressions since $\mathbf{x}_\alpha$ is free in $\mathbf{x}_\alpha = \mathbf{x}_\alpha$, but $\mathbf{x}_\alpha$ is not effective in $\mathbf{x}_\alpha = \mathbf{x}_\alpha$. "$\mathbf{x}_\alpha$ is effective in $\mathbf{B}_\beta$" is expressed in CTT$_{\text{qe}}$ as $\mathsf{IS\text{-}EFFECTIVE\text{-}IN}(\mathbf{x}_\alpha, \mathbf{B}_\beta)$, an abbreviation for

$$\exists\, \mathbf{y}_\alpha \,.\, ((\lambda\, \mathbf{x}_\alpha \,.\, \mathbf{B}_\beta)\, \mathbf{y}_\alpha \neq \mathbf{B}_\beta)$$

where $\mathbf{y}_\alpha$ is any variable of type α that differs from $\mathbf{x}_\alpha$.

The second case is simple since a quotation cannot be modified by substitution — it is effectively the same as a constant. Thus beta-reduction is performed without changing $\ulcorner \mathbf{B}_\beta \urcorner$ as shown in Axiom B9 above.

The third case is handled by Axioms B11.1 and B11.2. B11.1 deals with the trivial case when $\mathbf{A}_\alpha$ is the bound variable $\mathbf{x}_\alpha$ itself. B11.2 deals with the other much more complicated situation. The condition

$$\neg(\textsf{is-free-in}_{\epsilon\to\epsilon\to o} \ulcorner \mathbf{x}_\alpha \urcorner ((\lambda\, \mathbf{x}_\alpha \,.\, \mathbf{B}_\epsilon)\, \mathbf{A}_\alpha))$$

guarantees that there is no *double substitution*. $\textsf{is-free-in}_{\epsilon\to\epsilon\to o}$ is a logical constant of CTT$_{\text{qe}}$ such that $\textsf{is-free-in}_{\epsilon\to\epsilon\to o} \ulcorner \mathbf{x}_\alpha \urcorner \ulcorner \mathbf{B}_\beta \urcorner$ says that the variable $\mathbf{x}_\alpha$ is free in the (eval-free) expression $\mathbf{B}_\beta$.

Thus we see that substitution in CTT$_{\text{qe}}$ in the presence of evaluations may require proving semantic side conditions of the following two forms:

(1) $\neg\textsf{IS-EFFECTIVE-IN}(\mathbf{x}_\alpha, \mathbf{B}_\beta)$.

(2) $\neg(\textsf{is-free-in}_{\epsilon\to\epsilon\to o} \ulcorner \mathbf{x}_\alpha \urcorner \ulcorner \mathbf{B}_\beta \urcorner)$.

3.5 The Three Design Problems

To recap, CTT$_{\text{qe}}$ solves the three design problems given in section 1. The Evaluation Problem is avoided by restricting the quotation operator to eval-free expressions and thus making it impossible to express the liar paradox. The Variable Problem is overcome by modifying Andrews' beta-reduction axioms. The Double Substitution Problem is eluded by using a beta-reduction axiom for evaluations that excludes beta-reductions that embody a double substitution.

4 HOL Light

HOL Light [6] is an open-source proof assistant developed by John Harrison. It implements a logic (HOL) which is a version of Church's type theory. It is a simple implementation of the HOL proof assistant [4] written in OCaml and hosted on GitHub at `https://github.com/jrh13/hol-light/`. Although it is a relatively small system, it has been used to formalize many kinds of mathematics and to check many proofs including the lion's share of Tom Hales' proof of the Kepler conjecture [5].

HOL Light is very well suited to serve as a foundation on which to build an implementation of CTT$_{\text{qe}}$: First, it is an open-source system that can be freely modified as long as certain very minimal conditions are satisfied. Second, it is an implementation of a version of simple type theory that is essentially $\mathcal{Q}_0$, the version of Church's type theory underlying CTT$_{\text{qe}}$, plus (1) polymorphic type variables, (2) an axiom of choice expressed by asserting that the Hilbert ϵ operator is a choice (indefinite description) operator, and (3) an axiom of infinity that asserts that `ind`, the type of individuals, is infinite [6]. The type variables in the implemented logic are not a hindrance; they actually facilitate the implementation of CTT_{qe}. The presence of the axioms of choice and infinity in HOL Light alter the semantics of CTT_{qe} without compromising in any way the semantics of quotation and evaluation. And third, HOL Light supports the definition of inductive types so that ϵ can be straightforwardly defined.

5 HOL Light QE

5.1 Overview

HOL Light QE was implemented in four stages:

(1) The set of terms was extended so that CTT_{qe} expressions could be mapped to HOL Light terms. This required the introduction of `epsilon`, the type of constructions, and term constructors for quotations and evaluations. See subsection 5.2.

(2) The proof system was modified to include the machinery in CTT_{qe} for reasoning about quotations and evaluations. This required adding new rules of inference and modifying the `INST` rule of inference that simultaneously substitutes terms $t_1, \ldots, t_n$ for the free variables $x_1, \ldots, x_n$ in a sequent. See subsection 5.3.

(3) Machinery — consisting of HOL function definitions, tactics, and theorems — was created for supporting reasoning about quotations and evaluations in the new system. See subsection 5.4.

(4) Examples were developed in the new system to test the implementation and to demonstrate the benefits of having quotation and evaluation in higher-order logic. See section **??**.

The first and second stages have been completed; both stages involved modifying the kernel of HOL Light. The third stage is sufficiently complete to enable our examples in section **??** to work well and did not involve any further changes to the HOL Light kernel. We do expect that adding further examples, which is ongoing, will require additional machinery but no changes to the kernel.

The HOL Light QE system is being developed at McMaster University and is available at

`https://github.com/JacquesCarette/hol-light-qe`.

Under the supervision of the second and third authors, the core of the system was developed by the fourth author and the system was later extended and improved by the first and fifth authors. The fourth and fifth authors did their work on undergraduate NSERC USRA research projects.

To run HOL Light QE, execute the following commands in the HOL Light QE top-level directory named `hol-light-qe`:

```
1) install opam
2) opam init --comp 4.03.0
3) opam install "camlp5=6.16"
4) opam config env
5) cd hol-light-qe
6) make
7) ocaml -I `camlp5 -where` camlp5o.cma
8) #use "hol.ml";;
   #use "Constructions/epsilon.ml";;
   #use "Constructions/pseudoquotation.ml";;
   #use "Constructions/QuotationTactics.ml";;
```

Each test can be run by an appropriate further `#use` statement.

5.2 Mapping of CTT$_{\text{qe}}$ Expressions to HOL Terms

Tables 1 and 2 illustrate how the CTT$_{\text{qe}}$ types and expressions are mapped to the HOL types and terms, respectively. The HOL types and terms are written in the internal representation employed in HOL Light QE. The type `epsilon` and the term constructors `Quote` and `Eval` are additions to HOL Light explained below. Since CTT$_{\text{qe}}$ does not have type variables, it has a logical constant $=_{\alpha\to\alpha\to o}$ representing equality for each $\alpha \in \mathcal{T}$. The members of this family of constants are all mapped to a single HOL constant with the polymorphic type `a_ty_var->a_ty_var->bool` where `a_ty_var` is any chosen HOL type variable.

The other logical constants of CTT$_{\text{qe}}$ [3, Table 1] are not mapped to primitive HOL constants. $\mathsf{app}_{\epsilon\to\epsilon\to\epsilon}$, $\mathsf{abs}_{\epsilon\to\epsilon\to\epsilon}$, and $\mathsf{quo}_{\epsilon\to\epsilon}$ are implemented by `App`, `Abs`, and `Quo`, constructors for the inductive type `epsilon` given below. The remaining logical constants are predicates on constructions that are implemented by HOL functions. The CTT$_{\text{qe}}$ type ϵ is the type of constructions, the syntactic values that represent the syntax trees of eval-free expressions. ϵ is formalized as an inductive type `epsilon`. Since types are components of terms in HOL Light, an inductive type `type` of syntactic values for HOL Light QE types (which are the same as HOL types) is also needed. Specifically:

```
define_type "type = TyVar string
                  | TyBase string
                  | TyMonoCons string type
                  | TyBiCons string type type"
```

CTT$_{\text{qe}}$ Type α	HOL Type $\mu(\alpha)$	Abbreviation for $\mu(\alpha)$
o	`Tyapp("bool",[])`	`bool`
ι	`Tyapp("ind",[])`	`ind`
ϵ	`Tyapp("epsilon",[])`	`epsilon`
$\beta \to \gamma$	`Tyapp("fun",[`$\mu(\beta),\mu(\gamma)$`])`	$\mu(\beta)$`->`$\mu(\gamma)$

Table 1: Mapping of CTT$_{\text{qe}}$ Types to HOL Types

$\mathbf{CTT_{qe}}$ **Expression** e	**HOL Term** $\nu(e)$
$\mathbf{x}_\alpha$	`Var("x",`$\mu(\alpha)$`)`
$\mathbf{c}_\alpha$	`Const("c",`$\mu(\alpha)$`)`
$=_{\alpha\to\alpha\to o}$	`Const("=",a_ty_var->a_ty_var->bool)`
$(\mathbf{F}_{\alpha\to\beta}\,\mathbf{A}_\alpha)$	`Comb(`$\nu(\mathbf{F}_{\alpha\to\beta}),\nu(\mathbf{A}_\alpha)$`)`
$(\lambda\,\mathbf{x}_\alpha\,.\,\mathbf{B}_\beta)$	`Abs(Var("x",`$\mu(\alpha)$`),`$\nu(\mathbf{B}_\beta)$`)`
$\ulcorner\mathbf{A}_\alpha\urcorner$	`Quote(`$\nu(\mathbf{A}_\alpha),\mu(\alpha)$`)`
$[\![\mathbf{A}_\epsilon]\!]_{\mathbf{B}_\beta}$	`Eval(`$\nu(\mathbf{A}_\epsilon),\mu(\beta)$`)`

Table 2: Mapping of CTT_{qe} Expressions to HOL Terms

```
define_type "epsilon = QuoVar string type
                     | QuoConst string type
                     | App epsilon epsilon
                     | Abs epsilon epsilon
                     | Quo epsilon"
```

Terms of type `type` denote the syntax trees of HOL Light QE types, while the terms of type `epsilon` denote the syntax trees of those terms that are eval-free.

The OCaml type of HOL types in HOL Light QE

```
type hol_type = Tyvar of string
              | Tyapp of string * hol_type list
```

is the same as in HOL Light, but the OCaml type of HOL terms in HOL Light QE

```
type term = Var of string * hol_type
          | Const of string * hol_type
          | Comb of term * term
          | Abs of term * term
          | Quote of term * hol_type
          | Hole of term * hol_type
          | Eval of term * hol_type
```

has three new constructors: `Quote`, `Hole`, and `Eval`.

`Quote` constructs a quotation of type `epsilon` with components t and α from a term t of type α that is is eval-free. `Eval` constructs an evaluation of type α with components t and α from a term t of type `epsilon` and a type α. `Hole` is used to construct "holes" of type `epsilon` in a quasiquotation as described in [3]. A quotation that contains holes is a quasiquotation, while a quotation without any holes is a normal quotation. The construction of terms has been modified to allow a hole (of type `epsilon`) to be used where a term of some other type is expected.

The external representation of a quotation `Quote(t,ty)` is `Q_ t _Q`. Similarly, the external representation of a hole `Hole(t,ty)` is `H_ t _H`. The external representation of an evaluation `Eval(t,ty)` is `eval t to ty`.

CTT$_{\mathbf{qe}}$ Axioms	**NewRules of Inference**
B8 (Properties of Quotation)	`LAW_OF_QUO`
B10 (Properties of Evaluation)	
B10.1	`VAR_DISQUO`
B10.2	`CONST_DISQUO`
B10.3	`APP_DISQUO`
B10.4	`ABS_DISQUO`
B10.5	`QUO_DISQUO`
B11.2 (Beta-Reduction for Evaluations)	`BETA_REDUCE_EVAL`
B12 ("Not Free In" means "Not Effective In")	`NOT_FREE_NOT_EFFECTIVE_IN`
B13 (Beta-Reduction for Function Abstractions)	`BETA_REDUCE_ABS`

Table 3: New Inference Rules in HOL Light QE

5.3 Modification of the HOL Light Proof System

The proof system for CTT$_{\mathrm{qe}}$ is obtained by extending $\mathcal{Q}_0$'s with additional axioms B1–B13 (see 3.4). Since $\mathcal{Q}_0$ and HOL Light are both complete (with respect to the semantics of Henkin-style general models), HOL Light includes the reasoning capabilities of the proof system for $\mathcal{Q}_0$ but not the reasoning capabilities embodied in the B1–B13 axioms, which must be implemented in HOL Light QE as follows. First, the logical constants defined by Axioms B1–B4, B5, and B7 are defined in HOL Light QE as HOL functions. Second, the no junk (B6) and no confusion (B4) requirements for ϵ are automatic consequences of defining `epsilon` as an inductive type. Third, Axiom B9 is implemented directly in the HOL Light code for substitution. Fourth, Axiom B11.1 is subsumed by the the BETA rule in HOL Light. Fifth, the remaining axioms — B8, B10, B11.2, B12, and B13 — are implemented by new rules of inference in as shown in Table 3.

The `INST` rule of inference is also modified. This rule simultaneously substitutes a list of terms for a list of variables in a sequent. The substitution function `vsubst` defined in the HOL Light kernel is modified so that it works like substitution (via beta-reduction rules) does in CTT$_{\mathrm{qe}}$. The main changes are:

(1) A substitution of a term `t` for a variable `x` in a function abstraction `Abs(y,s)` is performed as usual if (1) `t` is eval-free and `x` is not free in `t`, (2) there is a theorem that says `x` is not effective in `t`, (3) `s` is eval-free and `x` is not free in `s`, or (4) there is a theorem that says `x` is not effective in `s`. Otherwise, if `s` or `t` is not eval-free, (5) the function abstraction application `Comb(Abs(x,Abs(y,s)),t)` is returned and if `s` and `t` are eval-free, (6) the variable `x` is renamed and the substitution is continued. When (5) happens, this part of the substitution is finished and the user can possibly continue it by applying `BETA_REDUCE_ABS`, the rule of inference corresponding to Axiom B13.

(2) A substitution of a term `t` for a variable `x` in a quotation `Quote(e,ty)` where `e` does not contain any holes (i.e., terms of the form `Hole(e',ty')`) returns `Quote(e,ty)` unchanged (as stated in Axiom B9). If `e` does contain holes, then `t` is substituted for the variable `x` in the holes in `Quote(e,ty)`.

(3) A substitution of a term `t` for a variable `x` in an evaluation `Eval(e,ty)` returns (1) `Eval(e,ty)` when `t` is `x` and (2) the function abstraction application `Comb(Abs(x,Eval(e,ty)),t)` otherwise. (1) is valid by Axiom B11.1. When (2) happens, this part of the substitution is finished and the user can possibly continue it by applying `BETA_REDUCE_EVAL`, the rule of inference corresponding to Axiom B11.2.

5.4 Creation of Support Machinery

The HOL Light QE system contains a number of HOL functions, tactics, and theorems that are useful for reasoning about constructions, quotations, and evaluations. An important example is the HOL function `isExprType` that implements the CTT_{qe} family of logical constants $\textsf{is-expr}^{\alpha}_{\epsilon \to o}$ where α ranges over members of $\mathcal{T}$. This function takes terms s_1 and s_1 of type `epsilon` and `type`, respectively, and returns true iff s_1 represents the syntax tree of a term t, s_2 represents the syntax tree of a type α, and t is of type α.

5.5 Metatheorems

We state three important metatheorems about HOL Light QE. The proofs of these metatheorems are straightforward but also tedious. We label the metatheorems as conjectures since their proofs have not yet been fully written down.

Conjecture 5.1 *Every formula provable in* HOL Light*'s proof system is also provable in* HOL Light QE*'s proof system.*

Proof sketch. HOL Light QE's proof system extends HOL Light's proof system with new machinery for reasoning about quotations and evaluations. Thus every HOL Light proof remains valid in HOL Light QE. $\square$

Note: All the proofs loaded with the HOL Light system continue to be valid when loaded in HOL Light QE. A further test for the future would be to load a variety of large HOL Light proofs in HOL Light QE to check that their validity is preserved.

Conjecture 5.2 *The proof system for* HOL Light QE *is sound for all formulas and complete for all eval-free formulas.*

Proof sketch. The analog of this statement for CTT_{qe} is proved in [3]. It should be possible to prove this conjecture by just imitating the proof for CTT_{qe}. $\square$

Conjecture 5.3 HOL Light QE *is a model-theoretic conservative extension of* HOL Light*.*

Proof sketch. A model of HOL Light QE is a model of HOL Light with definitions of the type ϵ and several constants and interpretations for the (quasi)quotation and evaluation operators. These additions do not impinge upon the semantics of HOL Light; hence every model of HOL Light can be expanded to a model of the HOL Light QE, which is the meaning of the conjecture. □

6 Binary Arithmetic

7 Related Work

8 Conclusion

Acknowledgments

References

[1] P. B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth through Proof.* Springer, second edition, 2002.

[2] A. Bawden. Quasiquotation in Lisp. In O. Danvy, editor, *Proceedings of the 1999 ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation*, pages 4–12, 1999. Technical report BRICS-NS-99-1, University of Aarhus, 1999.

[3] W. M. Farmer. Incorporating quotation and evaluation into Church's type theory. *Information and Computation*, 260:9–50, 2018.

[4] M. J. C. Gordon and T. F. Melham. *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic.* Cambridge University Press, 1993.

[5] T. Hales et al. A formal proof of the Kepler conjecture. *Forum of Mathematics, Pi*, 5, 2017.

[6] 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.

[7] L. Henkin. Completeness in the theory of types. *Journal of Symbolic Logic*, 15:81–91, 1950.

[8] W. V. O. Quine. *Mathematical Logic: Revised Edition.* Harvard University Press, 2003.