HOL Light QE*

Jacques Carette, William M. Farmer, and Patrick Laskowski

Computing and Software, McMaster University, Canada
http://www.cas.mcmaster.ca/~carette
http://imps.mcmaster.ca/wmfarmer

5 January 2018

Abstract. We are interested in algorithms that manipulate mathematical expressions in mathematically meaningful ways. Expressions are syntactic, but most logics do not allow one to discuss syntax. CTT_{qe} is a version of Church's type theory that includes quotation and evaluation operators that are similar to quote and eval in the Lisp programming language. Church's type theory is not so different from HOL Light, and to prove that CTT_{qe} is implementable, we decided to add quotation and evaluation to HOL Light to demonstrate this. Here we document our design and the challenges that needed to be overcome.

1 Introduction

A *syntax-based mathematical algorithm (SBMA)* manipulates mathematical expressions in a mathematically meaningful way. SBMAs are commonplace in mathematics. Examples include algorithms that compute arithmetic operations by manipulating numerals, linear transformations by manipulating matrices, and derivatives by manipulating functional expressions. Reasoning about the mathematical meaning of an SBMA requires reasoning about the relationship between how the expressions are manipulated by the SBMA and what the manipulations mean mathematically.

We argue in [4] that the combination of quotation and evaluation, along with some inference rules, provides the means to reason about the interplay between syntax and semantics, which is what is needed for reasoning about SBMAs. *Quotation* is an operation that maps an expression e to a special value called a *syntactic value* that represents the syntax tree of e. Quotation enables expressions to be manipulated as syntactic entities. *Evaluation* is an operation that maps a syntactic value s to the value of the expression that is represented by s. Evaluation enables meta-level reasoning via syntactic values to be reflected into object-level reasoning. Quotation and evaluation thus form an infrastructure for integrating meta-level and object-level reasoning. Quotation gives a form of *reification* of object-level values which allows introspection. Along with inference rules, this gives a certain amount of *logical reflection*. Evaluation adds to this some aspects of *computational reflection*.

* This research was supported by NSERC.

Incorporating quotation and evaluation operators — like quote and eval in the Lisp programming language — into a traditional logic like first-order logic or simple type theory is not a straightforward task. Several challenging design problems stand in the way. The three design problems that most concern us are the following. We will write the quotation and evaluation operators applied to an expression e as $\ulcorner e \urcorner$ and $[\![e]\!]$, respectively.

1. *Evaluation Problem.* An evaluation operator is applicable to syntactic values that represent formulas and thus is effectively a truth predicate. Hence, by the proof of Alfred Tarski's theorem on the undefinability of truth [10], if the evaluation operator is total in the context of a sufficiently strong theory like first-order Peano arithmetic, then it is possible to express the liar paradox using the quotation and evaluation operators. Therefore, the evaluation operator must be partial and the law of disquotation cannot hold universally (i.e., for some expressions e, $[\![\ulcorner e \urcorner]\!] \neq e$). As a result, reasoning with evaluation can be cumbersome and leads to undefined expressions.
2. *Variable Problem.* The variable x is not free in the expression $\ulcorner x + 3 \urcorner$ (or in any quotation). However, x is free in $[\![\ulcorner x + 3 \urcorner]\!]$ because $[\![\ulcorner x + 3 \urcorner]\!] = x + 3$. If the value of a constant c is $\ulcorner x + 3 \urcorner$, then x is free in $[\![c]\!]$ because $[\![c]\!] = [\![\ulcorner x + 3 \urcorner]\!] = x + 3$. Hence, in the presence of an evaluation operator, whether or not a variable is free in an expression may depend on the values of the expression's components. As a consequence, the substitution of an expression for the free occurrences of a variable in another expression depends on the semantics (as well as the syntax) of the expressions involved and must be integrated with the proof system for the logic. That is, a logic with quotation and evaluation requires a semantics-dependent form of substitution in which side conditions, like whether a variable is free in an expression, are proved within the proof system. This is a major departure from traditional logic.
3. *Double Substitution Problem.* By the semantics of evaluation, the value of $[\![e]\!]$ is the *value* of the expression whose syntax tree is represented by the *value* of e. Hence the semantics of evaluation involves a double valuation. This is most apparent when the value of a variable involves a syntax tree which refers to the name of that same variable. For example, if the value of a variable x is $\ulcorner x \urcorner$, then $[\![x]\!] = [\![\ulcorner x \urcorner]\!] = x = \ulcorner x \urcorner$. Hence the substitution of $\ulcorner x \urcorner$ for x in $[\![x]\!]$ requires one substitution inside the argument of the evaluation operator and another substitution after the evaluation operator is eliminated. This double substitution is another major departure from traditional logic.

CTT_{qe} [5,6] is version of Church's type theory [2] with quotation and evaluation that solves these three design problems. It is based on $\mathcal{Q}_0$ [1], Peter Andrews' version of Church's type theory. We believe CTT_{qe} is the first readily implementable version of simple type theory that includes *global* quotation and evaluation operators. We show in [5] that it is suitable for defining, applying, and reasoning about SBMAs.

To demonstrate that CTT_{qe} is indeed implementable, we have implemented CTT_{qe} by modifying HOL Light [8], a compact implementation of the HOL

logic [7]. The resulting version of HOL Light is called HOL Light QE. Here we present its design, implementation and the challenges to doing so.

The rest of the paper is organized as follows. Section 2 presents the key ideas underlying CTT_{qe} and explains how CTT_{qe} solves the three design problems given above. Section 3 offers a brief overview of HOL Light. The HOL Light QE implementation is described in section 4, and examples of how quotation and evaluation are used in it are discussed in section 5. Section 6 is devoted to related work. And the paper ends with some final remarks in section 7 including a brief discussion on future work.

2 CTT_{qe}

The syntax and semantics of CTT_{qe} as well as the proof system for CTT_{qe} are defined in [5]. In this section we will only introduce the definitions and results of CTT_{qe} that are key to understanding how HOL Light QE implements CTT_{qe}. The reader is encouraged to consult [5] when additional details are required.

2.1 Syntax

The syntax of CTT_{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_{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 $(\alpha \to \beta)$ is a type.

Let $\mathcal{T}$ denote the set of types of CTT_{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_{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_{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 $\mathsf{app}_{\epsilon\to\epsilon\to\epsilon}$, $\mathsf{abs}_{\epsilon\to\epsilon\to\epsilon}$, and $\mathsf{quo}_{\epsilon\to\epsilon}$.

An *expression of type* α of CTT_{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 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*: $(\mathbf{F}_{\alpha\to\beta}\, \mathbf{A}_\alpha)$ is an expression of type β.
4. *Function abstraction*: $(\lambda\, \mathbf{x}_\alpha\, .\, \mathbf{B}_\beta)$ is an expression of type $\alpha \to \beta$.

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 CTT_{qe} is an expression of type ϵ defined inductively as follows:

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 $\mathsf{app}_{\epsilon\to\epsilon\to\epsilon}\,\mathbf{A}_\epsilon\,\mathbf{B}_\epsilon$, $\mathsf{abs}_{\epsilon\to\epsilon\to\epsilon}\,\mathbf{A}_\epsilon\,\mathbf{B}_\epsilon$, and $\mathsf{quo}_{\epsilon\to\epsilon}\,\mathbf{A}_\epsilon$ are constructions.

The set of constructions is thus an inductive type whose base elements are quotations of variables and constants and whose constructors are $\mathsf{app}_{\epsilon\to\epsilon\to\epsilon}$, $\mathsf{abs}_{\epsilon\to\epsilon\to\epsilon}$, and $\mathsf{quo}_{\epsilon\to\epsilon}$. 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:

The definition below seems trivial, it just maps from one syntax to another. On the flip-side, if a reader is expecting a λ-calculus, then $\mathcal{E}$ in the app case is ill-defined as you can't pattern match on an application.

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

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, $\mathsf{app}_{\epsilon\to\epsilon\to\epsilon}\ulcorner \mathbf{x}_\alpha \urcorner\ulcorner \mathbf{x}_\alpha \urcorner$ represents the syntax tree of $(\mathbf{x}_\alpha\,\mathbf{x}_\alpha)$ which is not an expression of CTT_{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.

2.2 Semantics

The semantics of CTT_{qe} is based on Henkin-style general models [9]. 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 theorems hold:

Theorem 2.21 (Law of Quotation) $\ulcorner \boldsymbol{A}_\alpha \urcorner = \mathcal{E}(\boldsymbol{A}_\alpha)$ *is valid in* CTT_{qe}.

Theorem 2.22 (Law of Disquotation) $[\![\ulcorner \boldsymbol{A}_\alpha \urcorner]\!]_\alpha = \boldsymbol{A}_\alpha$ *is valid in* CTT_{qe}.

2.3 Proof System

The proof system for CTT_{qe} consists of the axioms for $\mathcal{Q}_0$, the single rule of inference for $\mathcal{Q}_0$, and additional axioms that extend the rules for beta-reduction, define the logical constants of CTT_{qe}, specify ϵ as an inductive type, and state the properties of quotation and evaluation. We prove in [5] that this proof system is sound for all formulas and complete for eval-free formulas.

Substitution is performed in the proof system for CTT_{qe} using the properties of beta-reduction as Andrews does in the proof system for $\mathcal{Q}_0$ [1, p. 213]. The syntactic notion of "a variable is free in an expression" is replaced in Andrews' beta-reduction axioms by the semantic notion of "a variable is effective in an expression" when the expression is not necessarily eval-free, and beta-reduction axioms for quotation and evaluation are added to Andrews' beta-reduction axioms. "$\mathbf{x}_\alpha$ 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 B_β" 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 $\mathbf{x}_\alpha = \mathbf{x}_\alpha$.

I think that some, if not most, of these rules should be shown. They either need to be here or in the implementation section, with a forward pointer.

Because substitution is so important, I think more will be needed – in the same vein as above.

2.4 Design Problems

CTT_{qe} solves the three design problems given in section 1. The Evaluation Problem is completely 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 as described above. The Double Substitution Problem is eluded by using a beta-reduction axiom for evaluations that excludes beta-reductions that embody a double substitution.

3 HOL Light

HOL Light [8] 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 [7] 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 Hale's proof of the Kepler conjecture [3].

HOL Light is very well suited to serve as a foundation on which to build an implementation of CTT_{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_{qe}, plus polymorphic type variables. The type variables in the implemented logic are not a hindrance; they actually facilitate the implementation of CTT_{qe}. And third, HOL Light supports the definition of inductive types so that ϵ can be straightforwardly defined.

4 Implementation

The implementation is available at `https://github.com/JacquesCarette/hol-light`.

5 Examples

6 Related Work

7 Conclusion

Todo list

The definition below seems trivial, it just maps from one syntax to another. On the flip-side, if a reader is expecting a λ-calculus, then $\mathcal{E}$ in the app case is ill-defined as you can't pattern match on an application. 4

I think that some, if not most, of these rules should be shown. They either need to be here or in the implementation section, with a forward pointer. 5

Because substitution is so important, I think more will be needed – in the same vein as above. 5

References

1. P. B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth through Proof, Second Edition*. Kluwer, 2002.
2. A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.
3. T. Hales et al. A formal proof of the Kepler conjecture. *Forum of Mathematics, Pi*, 5, 2017.
4. 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.
5. W. M. Farmer. Incorporating quotation and evaluation into Church's type theory. *Computing Research Repository (CoRR)*, abs/1612.02785 (72 pp.), 2016.
6. 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.
7. M. J. C. Gordon and T. F. Melham. *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*. Cambridge University Press, 1993.
8. 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.
9. L. Henkin. Completeness in the theory of types. *Journal of Symbolic Logic*, 15:81–91, 1950.
10. A. Tarski. The concept of truth in formalized languages. In J. Corcoran, editor, *Logic, Semantics, Meta-Mathematics*, pages 152–278. Hackett, second edition, 1983.