Towards Specifying Symbolic Computation⋆

Jacques Carette and William M. Farmer

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

Abstract. Many interesting and useful symbolic computation algorithms manipulate mathematical expressions in mathematically meaningful ways. Although these algorithms are commonplace in computer algebra systems, they can be surprisingly difficult to specify in a formal logic since they involve an interplay of syntax and semantics. In this paper we discuss several examples of syntax-based mathematical algorithms, and we show how to specify them in a formal logic with undefinedness, quotation, and evaluation.

1 Introduction

2 Background

2.1 Definedness, Equality, and Quasi-Equality

Let e be a mathematical expression and D be a domain of mathematical values. We say e *is defined in* D if e denotes an element in D. When e is defined in D, the *value of* e *in* D is the element in D that e denotes. When e is undefined in D, the value of e in D is undefined. Two expressions e and e' are *equal in* D, written $e =_D e'$, if they are both are defined in D and they have the same values in D and are *quasi-equal in* D, written $e \simeq_D e'$, if either $e =_D e'$ or e and e' are both undefined in D.

2.2 $\mathbf{CTT_{uqe}}$

CTT_{qe} [5] is a version of Church's type theory with an inductive type of syntactic values that represent the expressions of the logic, a quotation operator that maps expressions to syntactic values, and an evaluation operator that maps syntactic values to the values of the expressions that they represent. These components provide CTT_{qe} with a *global reflection facility* that is well-suited for reasoning about the interplay of syntax and semantics and, in particular, for specifying, defining, applying, and reasoning about SBMAs. The syntax and semantics of CTT_{qe} is presented in [5] in great detail. A proof system for CTT_{qe} that is sound for all formulas and complete for formulas that do not contain evaluations is also

⋆ This research is supported by NSERC.

presented in [5]. By modifying HOL Light [6], we have produced an implementation of CTT_{qe} called HOL Light QE [1].

CTT_{uqe} [4] is a variant of CTT_{qe} that has built-in support for partial functions and undefinedness based on the traditional approach to undefinedness [2]. It is well-suited for specifying SBMAs that manipulate expressions that may be undefined. Its syntax and semantics are presented in [4]. A proof system for CTT_{uqe} is not given in [4], but a proof system can be straightforwardly derived by merging the proof systems for CTT_{qe} [5] and $\mathcal{Q}_0^{\text{u}}$ [3].

3 Rational Expressions, Rational Functions

3.1 Rational Expressions

Let e be an expression in the language $\mathcal{L}$ of the field $\mathbb{Q}(x)$, that is, a well-formed expression built from the symbols $x, 0, 1, +, *, -, {}^{-1}$, elements of $\mathbb{Q}$ and parentheses (as necessary). For greater readability, we will take the liberty of using fractional notation for ${}^{-1}$ and the exponential notation x^n for $x * \cdots * x$ (n times). e can be something simple like $\frac{x^4-1}{x^2-1}$ or something more complicated like

$$\frac{\frac{1-x}{3/2x^{18}+x+17}}{\frac{1}{9834*x^{19393874}-1/5}} + 3*x - \frac{12}{x}.$$

We assume that $\mathbb{Q} \subseteq \mathbb{Q}[x] \subseteq \mathbb{Q}(x)$ so that the field of rational numbers and the ring of polynomials in x are included in $\mathbb{Q}(x)$. The expressions in $\mathcal{L}$ are intended to denote elements in $\mathbb{Q}(x)$. Of course, expressions like $x/0$ are undefined in $\mathbb{Q}(x)$. We will call members of $\mathcal{L}$ *rational expressions (over $\mathbb{Q}$)*.

We are taught that, like for members of $\mathbb{Q}$ (such as $5/15$), there is a *normal form* for rational expressions. This is typically defined to be a rational expression p/q for two polynomials $p, q \in \mathbb{Q}[x]$ such that p and q are themselves in polynomial normal form and $\mathsf{gcd}(p,q) = 1$. The motivation for the latter property is that we usually want to write $\frac{x^4-1}{x^2-1}$ as x^2+1 just as we usually want to write $5/15$ as $1/3$. Thus, the normal forms of $\frac{x^4-1}{x^2-1}$ and $\frac{x}{x}$ are x^2+1 and 1, respectively. This definition of normal form is based on the characteristic that the elements of the *field of fractions* of a ring R can be written as quotients r/s of elements of R where $r_0/s_0 = r_1/s_1$ if and only if $r_0 * s_1 = r_1 * s_0$ in R.

Every computer algebra system implements a function that *normalizes* expressions that denote elements of $\mathbb{Q}(x)$ (including elements of $\mathbb{Q}$ and $\mathbb{Q}[x]$). Let $\mathsf{normRatExpr}$ be the name of the algorithm that implements this normalization function on $\mathcal{L}$. Thus the signature of $\mathsf{normRatExpr}$ is $\mathcal{L} \rightarrow \mathcal{L}$ and the specification of $\mathsf{normRatExpr}$ is that, for all $e \in \mathcal{L}$, (A) $\mathsf{normRatExpr}(e)$ is a normal form and (B) $e \simeq_{\mathbb{Q}(x)} \mathsf{normRatExpr}(e)$. $\mathsf{normRatExpr}$ is an example of an SBMA. (A) is the syntactic component of its specification, and (B) is the semantic component.

Unfortunately that statement is not quite right, because normalization in a CAS merely means that the result can checked to be 0 (or not) in O(1) time. This leads to different normalizations for all 3, implemented in 3 different functions. It turns out that, in the univariate case, they correspond, but already for 2 variables things are different.

I think you might be conflating what CAS people call normal and canonical. Normal just means O(1) zero-testing, while canonical means a = b iff C(a) = C(b) with the later = being O(1) because of hash-consing

in the above, you never actually define what a normal form is!

3.2 Rational Functions

Let $\mathcal{L}'$ be the set of expressions of the form $\lambda\, x : \mathbb{Q} \,.\, e$ where $e \in \mathcal{L}$. We will call members of $\mathcal{L}'$ *rational functions (over $\mathbb{Q}$)*. That is, a rational function is a lambda expression whose body is a rational expression.

If $f_i = \lambda\, x : \mathbb{Q} \,.\, e_i$ are rational functions for $i = 1, 2$, one might think that $f_1 =_{\mathbb{Q}\to\mathbb{Q}} f_2$ if $e_1 =_{\mathbb{Q}(x)} e_2$. But this is not the case. For example, the rational functions $\lambda\, x : \mathbb{Q} \,.\, x/x$ and $\lambda\, x : \mathbb{Q} \,.\, 1$ are not equal since $\lambda\, x : \mathbb{Q} \,.\, x/x$ is undefined at 0 while $\lambda\, x : \mathbb{Q} \,.\, 1$ is defined everywhere. But $x/x =_{\mathbb{Q}(x)} 1$! Similarly, $\lambda\, x : \mathbb{Q} \,.\, (1/x - 1/x) \neq_{\mathbb{Q}\to\mathbb{Q}} \lambda\, x : \mathbb{Q} \,.\, 0$ and $(1/x - 1/x) =_{\mathbb{Q}(x)} 0$. Note that, in some contexts, we might want to say that $\lambda\, x : \mathbb{Q} \,.\, x/x$ and $\lambda\, x : \mathbb{Q} \,.\, 1$ do indeed denote the same function by invoking the concept of *removable singularities*.

I don't see why this reasoning is less clear as a justification that $\lambda\, x : \mathbb{Q} \,.\, (1/x - 1/x)$ and $\lambda\, x : \mathbb{Q} \,.\, 0$ are equal.

As we have just seen, we cannot normalize a rational function by normalizing its body, but we can normalize rational functions if we are careful not to remove points of undefinedness. Let a *quasinormal form* be a rational expression p/q for two polynomials $p, q \in \mathbb{Q}[x]$ such that p and q are themselves in polynomial normal form and there is no irreducible polynomial $r \in \mathbb{Q}[x]$ of degree ≥ 2 that divides both p and q. We can then normalize a rational function by quasinormalizing its body. Let $\mathsf{normRatFun}$ be the name of the algorithm that implements this normalization function on $\mathcal{L}'$. Thus the signature of $\mathsf{normRatFun}$ is $\mathcal{L}' \to \mathcal{L}'$ and the specification of $\mathsf{normRatFun}$ is that, for all $\lambda\, x : \mathbb{Q} \,.\, e \in \mathcal{L}'$, (A) $\mathsf{normRatFun}(\lambda\, x : \mathbb{Q} \,.\, e) = \lambda\, x : \mathbb{Q} \,.\, e'$ where e' is a quasinormal form and (B) $\lambda\, x : \mathbb{Q} \,.\, e \simeq_{\mathbb{Q}\to\mathbb{Q}} \mathsf{normRatFun}(\lambda\, x : \mathbb{Q} \,.\, e)$. $\mathsf{normRatFun}$ is another example of an SBMA. (A) is the syntactic component of its specification, and (B) is the semantic component.

Why those conditions on r? It is ok, over $\mathbb{Q}(x)$, to remove a common factor of $x^2 + 1$. Or even $x^2 - 2$!

3.3 The Problem Here

So why are we concerned about rational expressions and rational functions? The reason is that computer algebra systems make little distinction between the two: a rational expression can be interpreted sometimes as a rational expression and sometimes as a rational function. For example, one can always *evaluate* an expression by assigning values to its free variables or even convert it to a function. In Maple[1], these are done respectively via `eval(e, x = 0)` and `unapply(e, x)`. We can exhibit the problematic behaviour as follows: In fact, there is an even more pervasive, one could even say *obnoxious*, way of doing this: as the underlying language is *imperative*, it is possible to do:

insert some Maple code with output here

```
e := (x^4-1)/(x^2-1);
# many, many more lines of 'code'
x := 1;
 try to use 'e'
```

Hence, if an expression e is interpreted as a function, then it is not valid to simplify the function by applying $\mathsf{normRatExpr}$ to e, but computer algebra

[1] Mathematica has similar commands.

systems let the user do exactly this because usually there is no distinction made between e as a rational expression and e as representing a rational function, as we have already mentioned.

To avoid unsound applications of $\mathsf{normRatExpr}$, $\mathsf{normRatFun}$, and other SBMAs in mathematical systems, we need to carefully, if not formally, specify what these algorithms are intended to do. This is not a straightforward task to do in a traditional logic since SBMAs involve an interplay of syntax and semantics and algorithms like $\mathsf{normRatExpr}$ and $\mathsf{normRatFun}$ are very sensitive to definedness considerations. In the next subsection we will show how these two algorithms can be specified in a version of formal logic with undefinedness, quotation, and evaluation.

I don't know why we need to say this: "Of course, given some symbol y, $f(y)$ is in $\mathcal{L}$."

3.4 The Formal Specification of $\mathsf{normRatExpr}$ and $\mathsf{normRatFun}$

We will specify $\mathsf{normRatExpr}$ and $\mathsf{normRatFun}$ in CTT_{uqe}. To do this we need to develop a theory $T = (L, \Gamma)$ of CTT_{uqe} in which $\mathsf{normRatExpr}$ and $\mathsf{normRatFun}$ are constants in L, the language L of T, and their specifications are formulas in Γ, the set of axioms of T. A complete development of T would be long and tedious, so we will only sketch the development of T.

The first step is to define a theory $T_0 = (L_0, \Gamma_0)$ that axiomatizes $\mathbb{Q}$, the field of rational numbers. L_0 contains a base type q and constants 0_q, 1_q, $+_{q \to q \to q}$, $*_{q \to q \to q}$, $-_{q \to q}$, and ${}^{-1}{}_{q \to q}$ representing the standard elements and operators of a field. Γ_0 contains axioms that say the type q is the field of rational numbers.

The second step is to extend T_0 to a theory $T_1 = (L_1, \Gamma_1)$ that axiomatizes $\mathbb{Q}(x)$, the field of rational expressions over $\mathbb{Q}$. L_0 contains a base type r; constants 0_r, 1_r, $+_{r \to r \to r}$, $*_{r \to r \to r}$, $-_{r \to r}$, and ${}^{-1}{}_{r \to r}$ representing the standard elements and operators of a field; and a constant X_r representing the indeterminant of $\mathbb{Q}(x)$. Γ_0 contains axioms that say the type r is the field of rational expressions over $\mathbb{Q}$. Notice that the types q and r are completely separate from each other since CTT_{uqe} does not admit subtypes as in [**?**].

The third step is to extend T_1 to a theory $T_2 = (L_2, \Gamma_2)$ that is equipped to express ideas about the expressions of type q and $q \to q$ that have the form of rational expressions and rational functions, respectively. T_2 is obtain by defining the following constants:

1. $\mathsf{RatExpr}_{\epsilon \to o}$ is the predicate representing the subtype of ϵ that denotes the expressions of type q that have the form of rational expressions in x_q (i.e., the expressions of type q built from the variable x_q and the constants representing the field elements and operators for q). Thus $\mathsf{RatExpr}_{\epsilon \to o} \ulcorner x_q / x_q \urcorner$ is valid in T_2.
2. $\mathsf{RatFun}_{\epsilon \to o}$ is the predicate representing the subtype of ϵ that denotes the expressions of type $q \to q$ that are rational functions in x_q (i.e., the expressions of the form $\lambda\, x_q \,.\, \mathbf{R}_q$ where $\mathbf{R}_q$ is an expression having the form of a rational expression in x_q). Thus $\mathsf{RatFun}_{\epsilon \to o} \ulcorner \lambda\, x_q \,.\, x_q / x_q \urcorner$ is valid in T_2.

3. $\mathsf{val\text{-}in\text{-}}r_{\epsilon\to r}$ is a partial function that maps each member of the subtype $\mathsf{RatExpr}_{\epsilon\to o}$ to its denotation in r. Thus $\mathsf{val\text{-}in\text{-}}r_{\epsilon\to r}\ulcorner x_q +_{q\to q\to q} 1_q\urcorner = X_r +_{r\to r\to r} 1_r$ and $(\mathsf{val\text{-}in\text{-}}r_{\epsilon\to r}\ulcorner 1_q/0_q\urcorner)\uparrow$ are valid in T_2. Notice that the function is partial since an expression like $1_q/0_q$ does not denote a member of either the field q or r.
4. $\mathsf{Norm}_{\epsilon\to o}$ is the predicate representing the subtype of ϵ that denotes the members of the subtype $\mathsf{RatExpr}_{\epsilon\to o}$ that are normal forms. Thus $\neg(\mathsf{Norm}_{\epsilon\to o}\ulcorner x_q/x_q\urcorner)$ and $\mathsf{Norm}_{\epsilon\to o}\ulcorner 1_q\urcorner$ are valid in T_2.
5. $\mathsf{Quasinorm}_{\epsilon\to o}$ is the predicate representing the subtype of ϵ that denotes the members of the subtype $\mathsf{RatExpr}_{\epsilon\to o}$ that are quasinormal forms. Thus $\mathsf{Quasinorm}_{\epsilon\to o}\ulcorner x_q/x_q\urcorner$ and $\neg(\mathsf{Quasinorm}_{\epsilon\to o}\ulcorner A_q/A_q\urcorner)$, where A_q is $x_q^2 +_{q\to q\to q} 1_q$, are valid in T_2.
6. $\mathsf{body}_{\epsilon\to\epsilon}$ is a partial function that maps each member of ϵ denoting an expression of the form $\lambda\, x_\alpha\,.\,B_\beta$ to the member of ϵ that denotes B_β and is undefined on the rest of ϵ.

The final step is to extend T_2 to a theory $T = (L, \Gamma)$ in which L has two additional constants $\mathsf{normRatExpr}_{\epsilon\to\epsilon}$ and $\mathsf{normRatFun}_{\epsilon\to\epsilon}$ and Γ has two additional axioms $\mathsf{specNormRatExpr}_o$ and $\mathsf{specNormRatFun}_o$ that specify $\mathsf{normRatExpr}_{\epsilon\to\epsilon}$ and $\mathsf{normRatFun}_{\epsilon\to\epsilon}$. $\mathsf{specNormRatExpr}_o$ is the formula

$$\forall\, u_\epsilon\,. \tag{1}$$
$$\quad \mathsf{if}\ (\mathsf{RatExpr}_{\epsilon\to o}\, u_\epsilon) \tag{2}$$
$$\quad\quad (\mathsf{Norm}_{\epsilon\to\epsilon}(\mathsf{normRatExpr}_{\epsilon\to o}\, u_\epsilon) \wedge \tag{3}$$
$$\quad\quad\quad \mathsf{val\text{-}in\text{-}}r_{\epsilon\to r}\, u_\epsilon \simeq \mathsf{val\text{-}in\text{-}}r_{\epsilon\to r}(\mathsf{normRatExpr}_{\epsilon\to o}\, u_\epsilon)) \tag{4}$$
$$\quad\quad (\mathsf{normRatExpr}_{\epsilon\to o}\, u_\epsilon)\uparrow \tag{5}$$

(3) says that, if the input represents a rational expression in x_q, then the output represents a rational expression in x_q in normal form. (4) says that, if the input represents a rational expression in x_q, then either the input and output denote the same member of r or they both do not denote any member of r. And (5) says that, if the input does not represent a rational expression in x_q, then the output is undefined.

$\mathsf{specNormRatFun}_o$ is the formula

$$\forall\, u_\epsilon\,. \tag{6}$$
$$\quad \mathsf{if}\ (\mathsf{RatFun}_{\epsilon\to o}\, u_\epsilon) \tag{7}$$
$$\quad\quad (\mathsf{RatFun}_{\epsilon\to o}\,(\mathsf{normRatFun}_{\epsilon\to o}\, u_\epsilon) \wedge \tag{8}$$
$$\quad\quad\quad \mathsf{Quasinorm}_{\epsilon\to\epsilon}(\mathsf{body}_{\epsilon\to\epsilon}(\mathsf{normRatExpr}_{\epsilon\to o}\, u_\epsilon)) \wedge \tag{9}$$
$$\quad\quad\quad [\![u_\epsilon]\!]_{r\to r} = [\![\mathsf{normRatExpr}_{\epsilon\to o}\, u_\epsilon]\!]_{r\to r}) \tag{10}$$
$$\quad\quad (\mathsf{normRatFun}_{\epsilon\to o}\, u_\epsilon)\uparrow \tag{11}$$

(8-9) say that, if the input represents a rational function in x_q, then the output represents a rational function in x_q whose body is in quasinormal form. (10) says that, if the input represents a rational function in x_q, then input and output denote the same (possibly partial) function on the rational numbers. And (11) says that, if in input does not represent a rational function in x_q, then the output is undefined.

Not only is it possible to specifying the algorithms normRatExpr and normRatFun in $\textsc{ctt}_{\mathrm{uqe}}$, it is also possible to define the functions that these algorithms implement. Then applications of these functions can be evaluated using a proof system for $\textsc{ctt}_{\mathrm{uqe}}$.

4 Symbolic Differentiation of Rational Functions

5 Related Work

6 Conclusion

References

1. Carette, J., Farmer, W.M., Laskowski, P.: HOL Light QE. In: Avigad, J., Mahboubi, A. (eds.) Interactive Theorem Proving. Lecture Notes in Computer Science, Springer (2018), forthcoming
2. Farmer, W.M.: Formalizing undefinedness arising in calculus. In: Basin, D., Rusinowitch, M. (eds.) Automated Reasoning—IJCAR 2004. Lecture Notes in Computer Science, vol. 3097, pp. 475–489. Springer (2004)
3. Farmer, W.M.: Andrews' type system with undefinedness. In: Benzmüller, C., Brown, C., Siekmann, J., Statman, R. (eds.) Reasoning in Simple Type Theory: Festschrift in Honor of Peter B. Andrews on his 70th Birthday, pp. 223–242. Studies in Logic, College Publications (2008)
4. Farmer, W.M.: Theory morphisms in Church's type theory with quotation and evaluation. In: Geuvers, H., England, M., Hasan, O., Rabe, F., Teschke, O. (eds.) Intelligent Computer Mathematics. Lecture Notes in Computer Science, vol. 10383, pp. 147–162. Springer (2017)
5. Farmer, W.M.: Incorporating quotation and evaluation into Church's type theory. Information and Computation **260**, 9–50 (2018)
6. Harrison, J.: HOL Light: An overview. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds.) Theorem Proving in Higher Order Logics. Lecture Notes in Computer Science, vol. 5674, pp. 60–66. Springer (2009)

Todo list