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 $\mathbf{CTT_{uqe}}$

$\textsc{ctt}_{\mathrm{qe}}$ [?] 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 $\textsc{ctt}_{\mathrm{qe}}$ 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 $\textsc{ctt}_{\mathrm{qe}}$ is presented in [?] in great detail. A proof system for $\textsc{ctt}_{\mathrm{qe}}$ 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 [?]. By modifying HOL Light [?], we have produced an implementation of $\textsc{ctt}_{\mathrm{qe}}$ called HOL Light QE [?].

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

3 Factoring

3.1 Task

Here is a seemingly simple mathematical question: factor (over $\mathbb{N}$) the number 12. One common answer would be $12 = 2^2 \cdot 3$ — which is true, but not actually an answer in most systems! This is because, in any system with built-in β-reduction (as all CAS do, as do theorem provers based on dependent type theory), the above is *exactly* $12 = 12$, which is certainly not very informative.

3.2 The problem

So why is $2^2 \cdot 3$ not an answer? Because this mixes *syntax* and *semantics*. A better answer would be $\ulcorner 2^2 \cdot 3 \urcorner$, that would then make it clear that $\cdot$ *represents* multiplication rather than *being* multiplication. In other words, this is about intension and extension: we want to be able to both represent operations and perform operations. In Maple, one talks about inert forms, while in Mathematica, there are various related concepts such as `Hold`, `Inative` and `Unevaluated`. They both capture the same fundamental dichotomy about passive representations and active computations.

3.3 Solution

Coming back to integer factorization, interestingly both Maple and Mathematica choose a fairly similar option to represent the answer — a list of pairs, with the first component being the primes of the factorization, and the second component being the multiplicity (i.e. exponent); Maple furthermore gives a leading unit, so that one can also factor negative numbers. In other words, in Maple, the result of `ifactors(12)` is

$$[1, [2, 2], [3, 1]]$$

where lists are used (rather than proper pairs) as the host system is untyped (Mathematica does something similar).

3.4 Specification

Given the following Maple routine[1]

```
remult := proc(l :: [{-1,1}, list([prime,posint])])
  local f := proc(x, y) (x[1] ^ x[2]) * y end proc;
  l[1] * foldr(f, 1, op(l[2]))
end proc;
```

then the specification for `ifactors` is that for all $n \in \mathbb{Z}$,

$$\mathsf{remult}\,(\mathsf{ifactors}\,(n)) = n$$

and the primes in the decomposition are unique.

3.5 Discussion

Why do neither of the systems use their own means of representing intensional information? History! In both cases, the integer factorization routines predate the intensional features by more than *two decades*. And backward compatibility definitely prevents them from making that change.

Furthermore, factoring as an operation produces output in a very predicable *shape*: $u \cdot p_1^{i_1} \cdot p_2^{i_2} \cdot \ldots \cdot p_k^{i_k}$. Having to essentially parse such a term syntax to extract the information is tedious and error prone, at least in an untyped system. Such a shape could easily be coded up in a typed system using a very simple algebraic data type that would obviate the problem. But CAS are very good at manipulating lists[2], and thus this output *composes* well with other system features.

Nevertheless, the main lesson is that a simple mathematical question, such as factoring the number 12, which seems like a question about Peano Arithmetic, isn't. It is a question that can only be properly answered in a context with a significantly richer term language — either lists, or lists and pairs, or algebraic type, or a full syntactic term language.

4 Rational Expressions, Rational Functions

4.1 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 $x * \cdots * x$

[1] There are non-essential Maple-isms in this routine: because of how foldr is defined, op to transform a list to an expression sequence is needed; in other languages, this is unnecessary. Note however that it is possible to express the type extremely precisely.

[2] Unsurprisingly, since the builders of both Maple and Mathematica were well acquainted with Macsyma

(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!

4.2 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 $f_1 =_{\mathbb{Q}\rightarrow\mathbb{Q}} f_2$ 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, $\lambda\, x : \mathbb{Q}\ .\ (1/x - 1/x) \neq_{\mathbb{Q}\rightarrow\mathbb{Q}} \lambda\, x : \mathbb{Q}\ .\ 0$ 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 normRatFun be the name of the algorithm that implements this normalization function on $\mathcal{L}'$. Thus the signature of normRatFun is $\mathcal{L}' \to \mathcal{L}'$ and the specification of 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) $\lambda\, x : \mathbb{Q} \,.\, e \simeq_{\mathbb{Q}\to\mathbb{Q}} \mathsf{normRatFun}(\lambda\, x : \mathbb{Q} \,.\, e)$. 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 $x^2 - 2$!

4.3 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[3], 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 normRatExpr to e, but computer algebra 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 normRatExpr, 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 normRatExpr and 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 $\mathcal{L}$.”

4.4 The Formal Specification of normRatExpr and normRatFun

We will specify normRatExpr and normRatFun in $\textsc{ctt}_{\mathrm{uqe}}$. To do this we need to develop a theory $T = (L, \Gamma)$ of $\textsc{ctt}_{\mathrm{uqe}}$ in which normRatExpr and normRatFun are constants in L, the language L of T, and their specifications are formulas

[3] Mathematica has similar commands.

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, $+_{q \rightarrow q \rightarrow q}$, $*_{q \rightarrow q \rightarrow q}$, $-_{q \rightarrow q}$, and ${}^{-1}{}_{q \rightarrow q}$ 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, $+_{r \rightarrow r \rightarrow r}$, $*_{r \rightarrow r \rightarrow r}$, $-_{r \rightarrow r}$, and ${}^{-1}{}_{r \rightarrow r}$ 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 $q \rightarrow q$ that have the form of rational expressions and rational functions, respectively. T_2 is obtain by defining the following constants:

1. $\mathsf{RatExpr}_{\epsilon \rightarrow o}$ 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 $\mathsf{RatExpr}_{\epsilon \rightarrow o} \ulcorner x_q / x_q \urcorner$ is valid in T_2.
2. $\mathsf{RatFun}_{\epsilon \rightarrow o}$ is the predicate representing the subtype of ϵ that denotes the expressions of type $q \rightarrow q$ 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 $\mathsf{RatFun}_{\epsilon \rightarrow o} \ulcorner \lambda\, x_q \,.\, x_q / x_q \urcorner$ is valid in T_2.
3. $\mathsf{val\text{-}in\text{-}}r_{\epsilon \rightarrow r}$ is a partial function that maps each member of the subtype $\mathsf{RatExpr}_{\epsilon \rightarrow o}$ to its denotation in r. Thus $\mathsf{val\text{-}in\text{-}}r_{\epsilon \rightarrow r} \ulcorner x_q +_{q \rightarrow q \rightarrow q} 1_q \urcorner = X_r +_{r \rightarrow r \rightarrow r} 1_r$ and $(\mathsf{val\text{-}in\text{-}}r_{\epsilon \rightarrow r} \ulcorner 1_q / 0_q \urcorner) \uparrow$ 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. $\mathsf{Norm}_{\epsilon \rightarrow o}$ is the predicate representing the subtype of ϵ that denotes the members of the subtype $\mathsf{RatExpr}_{\epsilon \rightarrow o}$ that are normal forms. Thus $\neg(\mathsf{Norm}_{\epsilon \rightarrow o} \ulcorner x_q / x_q \urcorner)$ and $\mathsf{Norm}_{\epsilon \rightarrow o} \ulcorner 1_q \urcorner$ are valid in T_2.
5. $\mathsf{Quasinorm}_{\epsilon \rightarrow o}$ is the predicate representing the subtype of ϵ that denotes the members of the subtype $\mathsf{RatExpr}_{\epsilon \rightarrow o}$ that are quasinormal forms. Thus $\mathsf{Quasinorm}_{\epsilon \rightarrow o} \ulcorner x_q / x_q \urcorner$ and $\neg(\mathsf{Quasinorm}_{\epsilon \rightarrow o} \ulcorner A_q / A_q \urcorner)$, where A_q is $x_q^2 +_{q \rightarrow q \rightarrow q} 1_q$, are valid in T_2.
6. $\mathsf{body}_{\epsilon \rightarrow \epsilon}$ 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 $\mathsf{normRatExpr}_{\epsilon \rightarrow \epsilon}$ and $\mathsf{normRatFun}_{\epsilon \rightarrow \epsilon}$ and Γ has two additional axioms $\mathsf{specNormRatExpr}_o$ and $\mathsf{specNormRatFun}_o$ that specify $\mathsf{normRatExpr}_{\epsilon \rightarrow \epsilon}$ and $\mathsf{normRatFun}_{\epsilon \rightarrow \epsilon}$. $\mathsf{specNormRatExpr}_o$ is the formula

$$
\begin{array}{ll}
\forall\, u_\epsilon \;. & (1)\\
\quad \mathsf{if}\ (\mathsf{RatExpr}_{\epsilon\to o}\, u_\epsilon) & (2)\\
\qquad (\mathsf{Norm}_{\epsilon\to\epsilon}(\mathsf{normRatExpr}_{\epsilon\to o}\, u_\epsilon) \wedge & (3)\\
\qquad\quad \mathsf{val\text{-}in\text{-}}r_{\epsilon\to r}\, u_\epsilon \simeq \mathsf{val\text{-}in\text{-}}r_{\epsilon\to r}(\mathsf{normRatExpr}_{\epsilon\to o}\, u_\epsilon)) & (4)\\
\qquad (\mathsf{normRatExpr}_{\epsilon\to o}\, u_\epsilon)\!\uparrow & (5)
\end{array}
$$

(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

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

(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 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 CTT$_{\mathrm{uqe}}$.

5 Symbolic Differentiation of Rational Functions

6 Related Work

7 Conclusion

Todo list