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 SBMAs

2.3 $\mathbf{CTT_{uqe}}$

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 $x * \cdots * x$

* This research is supported by NSERC.

(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 $f_1 =_{\mathbb{Q}\rightarrow\mathbb{Q}} f_2$ 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 $\mathcal{L}' \rightarrow \mathcal{L}'$ 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) $\lambda\, x : \mathbb{Q} \,.\, e \simeq_{\mathbb{Q}\rightarrow\mathbb{Q}} \mathsf{normRatFun}(\lambda\, x : \mathbb{Q} \,.\, e)$. 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 $x^2 - 2$!

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 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 $\mathcal{L}$."

3.4 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

[1] 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 $T_0 = (L_0, \Gamma_0)$ for the field $\mathbb{Q}(x)$ of rational expressions in x over $\mathbb{Q}$. L_0 contains a base type κ that represents the elements of $\mathbb{Q}(x)$, a constant X_κ which represents the indeterminant symbol x of $\mathbb{Q}(x)$, and constants 0_κ, 1_κ, $+_{\kappa\to\kappa\to\kappa}$, $*_{\kappa\to\kappa\to\kappa}$, $-_{\kappa\to\kappa}$, and ${}^{-1_{\kappa\to\kappa}}$ representing the usual field elements and operators. Γ_0 contains axioms that say the type κ is a field.

The next step is to extend T_0 to a theory $T_1 = (L_1, \Gamma_1)$ by defining in T_0 the following constants:

1. $\mathbb{Q}_{\kappa\to o}$ is the predicate representing the subtype of κ that denotes $\mathbb{Q}$, the rational numbers. Thus $\mathbb{Q}_{\kappa\to o}\, 1_\kappa$ is valid in T_1.
2. $\mathbb{Q}[x]_{\kappa\to o}$ is the predicate representing the subtype of κ that denotes $\mathbb{Q}[x]$, the polynomials in x over $\mathbb{Q}$. Thus $\mathbb{Q}[x]_{\kappa\to o}\, X_\kappa$ is valid in T_1.
3. $\mathsf{RatExpr}_{\epsilon\to o}$ is the predicate representing the subtype of ϵ that denotes the expressions of type κ that have the form of rational expressions in x_κ (i.e., the expressions of type κ built from the variable x_κ and the field constants). Thus $\mathsf{RatExpr}_{\epsilon\to o}\ulcorner x_\kappa/x_\kappa\urcorner$ is valid in T_1.
4. $\mathsf{RatFun}_{\epsilon\to o}$ is the predicate representing the subtype of ϵ that denotes expressions of the form $\lambda\, x_\kappa\, .\, \mathbf{R}_\kappa$ where $\mathbf{R}_\kappa$ is a expression that has the form of a rational expression in x_κ. Thus $\mathsf{RatFun}_{\epsilon\to o}\ulcorner \lambda\, x_\kappa\, .\, x_\kappa/x_\kappa\urcorner$ is valid in T_1.
5. $\mathsf{val\text{-}in\text{-}}\kappa_{\epsilon\to\kappa}$ is a partial function that maps each member of the subtype $\mathsf{RatExpr}_{\epsilon\to o}$ to its denotation in κ. Thus $\mathsf{val\text{-}in\text{-}}\kappa_{\epsilon\to\kappa}\ulcorner 1_\kappa +_{\kappa\to\kappa\to\kappa} x_\kappa\urcorner = 1_\kappa +_{\kappa\to\kappa\to\kappa} X_\kappa$ and $\mathsf{val\text{-}in\text{-}}\kappa_{\epsilon\to\kappa}\ulcorner 1_\kappa/0_\kappa\urcorner\uparrow$ are valid in T_1. Notice that the function is partial since an expression like $1_\kappa/0_\kappa$ is undefined in κ.
6. $\mathsf{Norm}_{\epsilon\to o}$ 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 $\neg(\mathsf{Norm}_{\epsilon\to o}\ulcorner x_\kappa/x_\kappa\urcorner)$ and $\mathsf{Norm}_{\epsilon\to o}\ulcorner 1_\kappa\urcorner$ are valid in T_1.
7. $\mathsf{Quasinorm}_{\epsilon\to o}$ 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 $\mathsf{Quasinorm}_{\epsilon\to o}\ulcorner x_\kappa/x_\kappa\urcorner$ and $\neg(\mathsf{Quasinorm}_{\epsilon\to o}\ulcorner A_\kappa/A_\kappa\urcorner)$, where A_κ is $x_\kappa^2 +_{\kappa\to\kappa\to\kappa} 1_\kappa$, are valid in T_1.
8. $\mathsf{body}_{\epsilon\to\epsilon}$ is a partial function that maps each member of ϵ denoting an expression of 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_1 to a theory $T_2 = (L_2, \Gamma_2)$ in which L_2 has two additional constants $\mathsf{normRatExpr}_{\epsilon\to\epsilon}$ and $\mathsf{normRatFun}_{\epsilon\to\epsilon}$ and Γ_2 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

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

(3) says that if the input represents a rational expression then the output is a normal form. (4) says that if the input represents a rational expression then input and output are equal in the field of rational expressions. And (5) says that if in input does not represent a rational expression then the output is undefined.

$\mathsf{specNormRatFun}_o$ is the formula

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

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

4 Related Work

5 Conclusion

Todo list