What are the rules of “elementary algebra”?

James H. Davenport[1] and Christopher J. Sangwin[2]

[1] Department of Computer Science
University of Bath, Bath BA2 7AY, United Kingdom
`J.H.Davenport@bath.ac.uk`
[2] Maths Stats & OR Network, School of Mathematics
Birmingham, B15 2TT, United Kingdom
`C.J.Sangwin@bham.ac.uk`

Abstract. Many systems dealing with mathematics, including but not limited to computer-aided assessment (CAA) software, have to deal with “equality up to the usual rules of algebra”, but what this phrase means is often less clear. Even when they are clear in abstract, implementing them in a computer algebra system, which has to deal with the mathematics users have typed, complete with complications such as division (rather than multiplication by the inverse), binary subtraction etc., is far from clear. In this paper, we outline some pitfalls, and what we have learned about solving these problems.

Many systems dealing with mathematics, including but not limited to computer-aided assessment (CAA) software, have to deal with “equality up to the usual rules of algebra”, but what this phrase means is often less clear, and depends on context. Our focus, as in [3], is on that variety of computer-aided assessment in which we ask “has the user got ‘the right answer’?”. If not, how many marks should we give, and how can we provide effective feedback, for a near miss? In order to answer the first question, we need to define equality with ‘the right answer’. In this paper we identify various levels of equality, increasing in the number of objects they will consider equal (except between levels 3 and 4). To automatically provide feedback to students we often need to identify the levels between which equality fails to hold for the first time. We note that there is some ambiguity about reading what the user has entered, e.g. is `-x^b` really `-(x^b)` (normal) or `(-x)^b` (Excel)? The different decisions made during parser design mean that even experienced users may well get confused about mathematical precedence and those brought up using a variety of calculators may carry over calculator habits. Whatever the parsing precedences used, there is a lot to be said for showing the user a conventional 2-dimensional representation and asking “is this what you intended?”, as STACK [11] does.

To make this paper manageable, we will consider only *natural number* coefficients, as is usual in students’ work at this level. We therefore treat rational numbers as explicit quotients of integers and do not enter into questions such as “does $\frac{1}{2}$ equal `0.5`?”, important though they are. We say “natural numbers” because traditional mathematical notation is ambiguous about representations of negative integers: everyone agrees that -7 is as valid an integer as 7, and also accepts $x - 7$ as a polynomial, but, apparently inconsistently, rejects both $x7$ (which ought to be as valid as $x - 7$) and $x + -7$ (which ought to be as valid as $x + 7$). We note in passing the very subtle typographic difference in the spacing of the symbols which LaTeX uses here between $x - 7$ and $x{-7}$. The former is a binary minus, `x-7`, the latter juxtaposes x with the integer -7, `x{-7}`, which could be interpreted as implied multiplication.

We do not *explicitly* specify the domain of computation, as this is not usually done at this level. Where this matters it is, we claim, up to the teacher to ensure that the rules are relevant to the domain.

These "usual rules of algebra" would normally be described as 'underlying' in the terminology of [3], and therefore two expressions that differed by applications of these rules would not be regarded as 'different' from the point of view of Computer-Aided Assessment. Although we do not wish to push the analogy too far, we believe that "equality up to the usual rules of algebra" can be regarded as asking for equality at the level of some "deep structure" that underlies the "surface structure" of traditional computer-science parsing, which is the case of `+` and `*` only, is given by level 6 in the next section. As [3] points out, one *might* also wish to be more relaxed about what forms of expression are to be considered as equal, and we will illustrate some opportunities for this.

One example of this is given in [7], who define[3]

$$\begin{aligned} \mathit{Expr} = {} & \mathit{Nat\ Integer} \mid \mathit{Var\ String} \mid \mathit{Negate\ Expr} \\ & \mathit{Expr} \mathbin{\texttt{:+:}} \mathit{Expr} \mid \mathit{Expr} \mathbin{\texttt{:*:}} \mathit{Expr} \mid \mathit{Expr} \mathbin{\texttt{:-:}} \mathit{Expr} \mid \mathit{Expr} \mathbin{\texttt{:/:}} \mathit{Expr} \end{aligned}$$

Their context is slightly different, being focused on "interactive exercise assistants", rather than the 'marking' context of [3, 11]. We compare the two approaches in section 7.

If it were not for the *Negate*, `:-:` and `:/:` clauses, equality of terms "in the sense of elementary algebra", is precisely equality of terms modulo associativity and commutativity (but not distributivity) of `:+:` and `:*:`, and this a well-studied, and much implemented topic. Level 6 below corresponds to a canonical form for associative–commutative representations.

1 Addition and Multiplication

Let us first consider expressions built with just addition and multiplication. There are various levels of equality here. We start by considering raw strings, as typed by the student, then move to the parse trees these strings represent.

1. Textual identity. Here `x+ y` and `x+y` are considered different because of the space. Similarly `x+10` and `x+010` are different because of the leading zero[4].

[3] They use `:+:` etc. to denote the *syntactic* operators, as entered by the user.
[4] We ignore C's rule that a leading 0 implies octal!

2. Equality after lexical analysis. This would consider the above examples the same, but would distinguish `x+y*z` from `x+(y*z)` because of the extra parentheses.
3. Equality as binary parse trees. This would consider `x+y*z` and `x+(y*z)` as the same, both being the tree

$$\begin{tikzcd}
 & + \arrow[dl] \arrow[dr] & & \\
x & & * \arrow[dl] \arrow[dr] & \\
 & y & & z
\end{tikzcd} .$$

   However, this interpretation would distinguish `x+(y+z)` from `(x+y)+z`, being respectively the trees

$$\begin{tikzcd}
 & + \arrow[dl] \arrow[dr] & & & & & + \arrow[dl] \arrow[dr] & \\
x & & + \arrow[dl] \arrow[dr] & \text{and} & & + \arrow[dl] \arrow[dr] & & z \\
 & y & & z & x & & y &
\end{tikzcd} . \qquad (1)$$

   Being based on binary parse trees, it requires an operator precedence rule (is `+` left-associative or right-associative) to decide which of these becomes the representation of `x+y+z`.
4. We can avoid this if we allow n-ary trees, parsing `x+y+z` as

$$\begin{tikzcd}
 & + \arrow[dl] \arrow[d] \arrow[dr] & \\
x & y & z
\end{tikzcd} , \qquad (2)$$

   but this is now a third tree, different from the two trees in (1).
5. We can avoid this problem if we *flatten*[5] the associative operators `+` and `*`, which would transform both the trees in (1) into the tree in (2). While there are other ways of handling associativity, e.g. by *rotating* the second tree in (1) into the first, they do not differ fundamentally, and this is the one which generalises naturally.
6. It is customary to believe that "the rules of elementary algebra" include the facts that $+$ and $\times$ are commutative, as well as associative. In the formalism of level 5 above, this is easy to handle: we merely say that the children of a `+` (or `*`) node are *compared as (multi)sets*[6], rather than as lists, so now

[5] Such flattening is explicitly prohibited in some programming languages, such as Fortran [1], but then the `+` of floating point arithmetic is not associative.

[6] We say multisets, since `x+x+x` is different from `x+x`. In a pedagogical application, of course, we would expect to see `3x` etc., but nevertheless we must maintain the distinction. In the terminology of [3], writing `x+x+x` for `3x` would normally be considered as a *venial* error, whereas writing `x+x` for `3x` is a fundamental one.

the tree in (2) has the same children, admittedly in a different order, as the trees

$$\begin{tikzcd} & + \arrow[dl] \arrow[d] \arrow[dr] & & & & + \arrow[dl] \arrow[d] \arrow[dr] & \\ x & z & y & \text{and} & z & y & x \end{tikzcd} , \tag{3}$$

and three others. We note the necessity of doing this multiset-wise comparison recursively, to ensure that

$$\begin{tikzcd} & + \arrow[dl] \arrow[dr] & & & & & + \arrow[dl] \arrow[dr] & \\ x & & * \arrow[dl] \arrow[dr] & \text{and} & & * \arrow[dl] \arrow[dr] & & x \\ & y & & z & z & & y & \end{tikzcd} \tag{4}$$

are regarded as equal.

Beyond this we are in the realm of computer algebra, rather than syntactic manipulation. However, even for assessment of elementary expressions this move is necessary. Otherwise one is left with some kind of equality tester, often probabilistic such as [12, 13], as refined in [8], but possibly deterministic [9]. Examples of this include Maple's `testeq`. If we really want an equality test at this level, systems that act[7] *radically* in the sense of [10] compute a canonical form. Hence we define three further levels.

7. We remove identity elements: i.e. we remove 0 from the arguments of $+$ and 1 from the arguments of $\times$. Where necessary, we flatten to remove operations which end up with a single argument. For example

$$\begin{tikzcd} & + \arrow[ddl] \arrow[ddr] & & & + \arrow[dd] & \\ & & & \Rightarrow & & \Rightarrow x. \\ x & & 0 & & x & \end{tikzcd}$$

   Pairs of $\mathcal{I}$ symbols, defined in Section 3, are removed until at most one may appear among the arguments of $\times$, and pairs of $\mathcal{R}$ symbols are removed, as discussed after (13).
8. We combine all numerical terms in the arguments of $+$ and $\times$, and (for the sake of user presentation) order the result so that any number appears first for $\times$ and last for $+$.
9. We apply the distributive law (of multiplication over addition).

Depending on the precise application, and the definition being adopted of "obviously equal", somewhere at or between levels 6 and 9 lies a system that gets many examples "right", in the sense that the user is not upset that the system refuses to recognise as equal expressions in `+` and `*` that are "obviously equal".

However, the grammar of even elementary algebra is larger than this, and includes $-$ (in both unary and binary forms) and $/$. Practically any system for

[7] [10] talks about "radical systems", but most systems which can behave conservatively, such as Macsyma and Reduce, have options not to.

processing mathematics has to deal with these, and their impact on the problem of "equality modulo the usual rules of elementary algebra". It might be thought that `/` and (binary) `-` posed the same problems, being the inverse operations to `*` and `+` respectively, but in fact that is not quite the case, as the unary operators behave differently.

2 Subtraction

Here we have both unary and binary `-`. It would seem that the informal rules of understanding mathematics make the transformation `a-b`$\rightarrow$`a+(-b)` at a very early stage in our mental processing. Hence the expression `a-b-c` is thought of as `a+(-b)+(-c)`, and hence, in levels 4 and beyond, this would become the tree

$$\begin{tikzcd} & + \arrow[dl] \arrow[d] \arrow[dr] & \\ a & - \arrow[d] & - \arrow[d] \\ & b & c \end{tikzcd} . \tag{5}$$

Level 6 would then regard this as equal to the tree from `a-c-b`, which few would disagree with. It would, however, also regard this as equal to the expression `-b+a-c`. Logically, one cannot object to this, but nonetheless we "tend to prefer" one of the other forms. Convention urges minimality here, since the other forms require one fewer signs. While a notion of "simplification" can be based on minimality, as in [4], it is harder to make an argument based on "the laws of algebra".

3 Negation in Parse Trees

So far we have followed a relatively standard approach to parsing, regarding unary $-$ as a separate operator in our parse trees, even at level 9. In practice, this will not do, so we then have to decide whether `-x*y` represents

$$\begin{tikzcd} & \texttt{-(x*y)} & \\ & - \arrow[d] & \\ & * \arrow[dl] \arrow[dr] & \\ x & & y \end{tikzcd} \quad \text{or} \quad \begin{tikzcd} & \texttt{(-x)*y} & \\ & * \arrow[dl] \arrow[dr] & \\ - \arrow[d] & & y \\ x & & \end{tikzcd} . \tag{6}$$

However, the typical user of mathematics would say that these are equal, by "the usual rules of algebra", and would object to being forced to distinguish.

We therefore propose[8] replacing "unary minus" by multiplication by a special token, hereafter $\mathcal{I}$. Hence `-(x*y)`, `(-x)*y` and `x*(-y)` become the trees

$$\begin{tikzcd} & * \arrow[dl] \arrow[dr] & & \\ \mathcal{I} & & * \arrow[dl] \arrow[dr] & \\ & x & & y \end{tikzcd} , \begin{tikzcd} & & * \arrow[dl] \arrow[dr] & \\ & * \arrow[dl] \arrow[dr] & & y \\ \mathcal{I} & & x & \end{tikzcd} \text{ and } \begin{tikzcd} & * \arrow[dl] \arrow[dr] & & \\ x & & * \arrow[dl] \arrow[dr] & \\ & \mathcal{I} & & y \end{tikzcd} \tag{7}$$

respectively. In the presence of flattening and commutativity of `*`, these trees are regarded as equal. What about `(-x)*(-y)`? This corresponds to the tree

$$\begin{tikzcd} & & & * \arrow[dll] \arrow[drr] & & & \\ & * \arrow[dl] \arrow[dr] & & & & * \arrow[dl] \arrow[dr] & \\ \mathcal{I} & & x & & \mathcal{I} & & y \end{tikzcd} , \tag{8}$$

which is equivalent to the tree from `-(-(x*y))` and many other variants with (multi)set of children $\{\mathcal{I}, \mathcal{I}, x, y\}$, but *not* equivalent to the tree from `x*y` until we get to level 7.

Algebraically $\mathcal{I}$ behaves like -1, but is distinguished from it at this level so as not to be confused with an explicit -1 entered by the user, i.e. `-x*y` versus `x*(-1)*y`:

$$\begin{tikzcd} & * \arrow[dl] \arrow[dr] & & \\ \mathcal{I} & & * \arrow[dl] \arrow[dr] & \\ & x & & y \end{tikzcd} \textit{ versus } \begin{tikzcd} & * \arrow[dl] \arrow[dr] & & \\ x & & * \arrow[dl] \arrow[dr] & \\ & * \arrow[dl] \arrow[dr] & & y \\ \mathcal{I} & & 1 & \end{tikzcd} , \tag{9}$$

where the (multi)sets of children of a flattened multiplication operator are $\{\mathcal{I}, x, y\}$ and $\{x, \mathcal{I}, 1, y\}$ respectively.

This distinction *should*, we believe, be made at this level, where we claim that level 6 together with $\mathcal{I}$ as the interpretation of unary and binary `-` is the appropriate "deep structure". Equally though, the exercise author or equivalent may add further rules, such as levels 7, 8 and 9 to equate the two, whether with or without a penalty.

4 Division

Our treatment of division, reduction to the unary case, is similar to that of binary subtraction, even though the surface representation is different. We consider a

[8] And have actually implemented in STACK [11].

unary reciprocal operator $\mathcal{R}$, and regard `x/y` as generating the tree

$$\begin{tikzcd}
& * \arrow[dl] \arrow[dr] & \\
x & & \mathcal{R} \arrow[d] \\
& & y
\end{tikzcd} \quad . \tag{10}$$

We need one further rule, which we will express here in a simple form, and express more generally in section 5:

"$\mathcal{R}$ distributes over multiplication", i.e.

$$\begin{tikzcd}
& \mathcal{R} \arrow[d] & & & * \arrow[dl] \arrow[dr] & \\
& * \arrow[dl] \arrow[dr] & \Rightarrow & \mathcal{R} \arrow[d] & & \mathcal{R} \arrow[d] \\
x & & y & x & & y
\end{tikzcd} \quad . \tag{11}$$

With this addition, and the rules from level 6, all the following generate the tree

$$\begin{tikzcd}
& * \arrow[dl] \arrow[d] \arrow[dr] & \\
x & \mathcal{R} \arrow[d] & \mathcal{R} \arrow[d] \\
& y & z
\end{tikzcd} \quad : \tag{12}$$

`(x/y)/z`, `x/y/z` and `x/(y*z)`. `x/(y/z)` generates

$$\begin{tikzcd}
& * \arrow[dl] \arrow[d] \arrow[dr] & \\
x & \mathcal{R} \arrow[d] & \mathcal{R} \arrow[d] \\
& y & \mathcal{R} \arrow[d] \\
& & z
\end{tikzcd} \quad , \tag{13}$$

and it is a good question whether or not this should be simplified by means of $\mathcal{R}(\mathcal{R}(z)) \Rightarrow z$. Our suggestion is that it should not at level 6, but that this rule should be readily available in the toolkit of an exercise writer (e.g. at level 7).

There is one slight complication here, which is the absence of a general notation for $\mathcal{R}(z)$, at least until we allow powers and the `z^(-1)` notation. This therefore means that we should probably elide

$$\texttt{1/y} \to \mathcal{R}(y) \qquad \text{(rather than } 1 * \mathcal{R}(y)\text{)}. \tag{14}$$

5 Powers

Adding exponentiation, whether just raising to an integer power, or more generally, complicates matters, and we note that it is not present in [7]. We should first note that ^ is *not* associative, and that precedence issues are therefore trickier here (as indeed they are in computer languages in general). Let us first consider the case of integer exponents. This may be either explicit integers, or (worse from our points of view) a set of problems where the integrality is implicit. This implicitness would be achieved by putting in scope rules that are not vaid in a more general context.

The first challenge is that the distributive law for exponentiation over multiplication[9] —

$$(ab)^c = a^c b^c \tag{15}$$

— seems to be viewed differently from the distributive law for multiplication over addition: we do not regard $(xy)^2$ as better or worse than x^2y^2. This is easily implemented as a tree transformation:

$$\begin{tikzcd}
& & \hat{} \arrow[dl] \arrow[dr] & \\
& * \arrow[dl] \arrow[dr] & & z \\
x & & y &
\end{tikzcd}
\Rightarrow
\begin{tikzcd}
& & & * \arrow[dll] \arrow[drr] & & & \\
& \hat{} \arrow[dl] \arrow[dr] & & & & \hat{} \arrow[dl] \arrow[dr] & \\
x & & z & & y & & z
\end{tikzcd}
. \tag{16}$$

Equally, we do not regard $\left(\frac{x}{y}\right)^2$ as better or worse than $\frac{x^2}{y^2}$. The transformation (16) automatically leads to

$$\begin{tikzcd}
& & \hat{} \arrow[dl] \arrow[dr] & \\
& * \arrow[dl] \arrow[dr] & & z \\
x & & \mathcal{R} \arrow[d] & \\
& & y &
\end{tikzcd}
\Rightarrow
\begin{tikzcd}
& & & * \arrow[dll] \arrow[drr] & & & \\
& \hat{} \arrow[dl] \arrow[dr] & & & & \hat{} \arrow[dl] \arrow[dr] & \\
x & & z & & \mathcal{R} \arrow[d] & & z \\
& & & & y & &
\end{tikzcd}
, \tag{17}$$

but this is not quite what was desired. It is tempting to try to "fix" this by rules taking $\mathcal{R}(x)^y$ to $\mathcal{R}(x^y)$, but our experience is that it is better to "bite the bullet" and accept that $\mathcal{R}$ is really raising to the power -1, and introduce the rules

$$\begin{tikzcd}
& \hat{} \arrow[dl] \arrow[dr] & \\
\mathcal{R} \arrow[d] & & z \\
x & &
\end{tikzcd}
,
\begin{tikzcd}
& \mathcal{R} \arrow[d] & \\
& \hat{} \arrow[dl] \arrow[dr] & \\
x & & z
\end{tikzcd}
\Rightarrow
\begin{tikzcd}
& \hat{} \arrow[dl] \arrow[dr] & & \\
x & & * \arrow[dl] \arrow[dr] & \\
& \mathcal{I} & & z
\end{tikzcd}
. \tag{18}$$

[9] The other distributive law, $a^{b+c} = a^b a^c$, *does* seem to be in the same camp as the distributive law for multiplication over addition, and should probably not generally be applied.

`1/x^(-2)` is then represented as $x^{\mathcal{I}*\mathcal{I}*2}$. It is probably a matter of taste whether one regards `1/x^(-2)` as a clumsy expression which does not deserve to be simplified, or whether one wishes to work at level 7, in which case $\mathcal{I}*\mathcal{I}*z \Rightarrow z$ (or the pair $\mathcal{I}*\mathcal{I} \Rightarrow 1$ and $1*z \Rightarrow z$) need to be added to the ruleset.

6 Powers: non-integral exponents

We note that these re-write rules for exponentiation are subtle once we get to non-integral exponents and that even great thinkers have made mistakes, such as [5]

> §148. Moreover, as $\sqrt{a}$ multiplied by $\sqrt{b}$ makes $\sqrt{ab}$ we shall have $\sqrt{6}$ for the value of $\sqrt{-2}$ multiplied by $\sqrt{-3}$; and $\sqrt{4}$ or 2, for the value of the product of $\sqrt{-1}$ and $\sqrt{-4}$. Thus we see that two imaginary numbers, multiplied together, produce a real, or possible one.
> But, on the contrary, a possible number, multiplied by an impossible number, gives always an imaginary product: thus $\sqrt{-3}$ by $\sqrt{+5}$, gives $\sqrt{-15}$.
> §149. It is the same with regard to division; for $\sqrt{a}$ divided by $\sqrt{b}$ making $\sqrt{\frac{a}{b}}$, it is evident that $\sqrt{-4}$ divided by $\sqrt{-1}$ will make $\sqrt{+4}$ or 2; that $\sqrt{+3}$ divided by $\sqrt{-3}$ will give $\sqrt{-1}$; and that 1 divided by $\sqrt{-1}$ gives $\sqrt{\frac{+1}{-1}}$, or $\sqrt{-1}$; because 1 is equal to $\sqrt{+1}$.

A more detailed discussion of the mathematics underlying these issues is given by [6]. When it comes to expressions, rather than numbers, (15) is not in general valid: compare

$$\sqrt{1-z}\sqrt{1+z}\overset{?}{=}\sqrt{1-z^2} \tag{19}$$

with

$$\sqrt{z-1}\sqrt{z+1}\overset{?}{=}\sqrt{z^2-1}, \tag{20}$$

where (19) *is* universally valid, but (20) only on a halfplane. These difficulties are discussed in [2], where the proofs of validity are fundamentally not of a rewriting nature.

As an example, consider $\frac{1}{\sqrt{-4}}$: we can operate on this parse tree using (18) then (16) to give

$$\begin{tikzcd}
& & \mathcal{R} \arrow[d] & \\
& & \hat{} \arrow[dl] \arrow[dr] & \\
& * \arrow[dl] \arrow[dr] & & \mathcal{R} \arrow[d] \\
\mathcal{I} & & 4 & 2
\end{tikzcd}
\Rightarrow
\begin{tikzcd}
& & & \hat{} \arrow[dll] \arrow[drr] & & & \\
& * \arrow[dl] \arrow[dr] & & & & * \arrow[dl] \arrow[dr] & \\
\mathcal{I} & & 4 & & \mathcal{I} & & \mathcal{R} \arrow[d] \\
& & & & & & 2
\end{tikzcd}
\Rightarrow
\begin{tikzcd}
& & & \hat{} \arrow[dll] \arrow[drr] & & & \\
& * \arrow[dl] \arrow[dr] & & & & * \arrow[dl] \arrow[dr] & \\
\mathcal{I} & & H & & 4 & & H
\end{tikzcd}$$

(i.e. $(-)^{-\frac{1}{2}} \times 4^{-\frac{1}{2}}$) where $H = -\frac{1}{2}$ represents the tree

$$\begin{tikzcd} & * \arrow[dl] \arrow[dr] & \\ H := \mathcal{I} & & \mathcal{R}\,. \arrow[d] \\ & & 2 \end{tikzcd}$$

Further computer algebra rules are required to decide how $(-)^{-\frac{1}{2}} \times 4^{-\frac{1}{2}}$ should be dealt with.

7 Comparison with [7]

The paper [7] cites five basic assumptions, which we reproduce below (our layout), and comment on (marked **C**).

1a Associativity of operators is implicit, meaning that a user cannot and should not distinguish a+(b+c) from(a+b)+c. The system can thus minimize the use of parentheses in presenting terms.

C Level 5 and later deal with this.

1b Commutativity, on the other hand, should be used with care. We want to respect the order in which terms appear as much as possible for a better user experience.[10]

C This can be achieved by not aggressively re-ordering the multiset of children of a `+` or `*` node.

2 Constant terms are normalized aggressively: the skills to manipulate fractions and integers are assumed to be present.

C This contrasts with our approach, where we would normally prefer[11] the answer `4` over `2+2`. However, adding constant normalisation rules as "underlying" rather than "venial" would achieve this goal.

3a The distribution of multiplication over addition (law [M5]) is an explicit step in the derivation.

C This is essentially the difference between our level 6, which we claim to be the appropriate "deep structure", and level 9.

3b Laws to manipulate the sign of a term (laws [N1] up to [N6]) can be performed automatically.

[N1] $-(-a) = a$
[N2] $a - a = 0$
[N3] $a - b = a + (-b)$
[N4] $-(a + b) = (-a) + (-b)$
[N5] $-(a \cdot b) = (-a) \cdot b$
[N6] $-(a/b) = (-a)/b$

[10] Recall that they are *generating* terms, whereas we are not.
[11] In the sense that we would mark `2+2` down.

C [N3], [N5] and [N6] are consequences of our treatment of $\mathcal{I}$: the others could certainly be added as underlying rules. [N4], in particular, is the equivalent for $\mathcal{I}$ of (11) for $\mathcal{R}$.

It would therefore be reasonable to state that our level 6, augmented by $\mathcal{I}$ and $\mathcal{R}$, and (11), provides a subset of the functionality [7] need, and one which can be further enhanced to theirs by adding certain rules as 'underlying' — rules that we would probably wish to have as "venial' in any case.

8 Conclusions

We claim that "equality up to the usual rules of algebra", for the four operations of `+`, `-`, `*` and `/`, can be represented as equality of the deep structures of level 6 (associative–commutative equality for `+` and `*`) together with $\mathcal{I}$ and $\mathcal{R}$ for unary subtraction and multiplicative inverse. Further rules, such as those posited in [7], can then be added on top of these if required, but these rules alone suffice for the purposes of much computer-aided assessment.

If one wishes to add `^`, things become more complicated. They become more complicated pedagogically because they *are* more complicated mathematically, principally because fractional powers require multi-valued inverses. This moves us away from an *algebra* of simple expressions represented by parse trees. Restriction to explicit integer exponents can be dealt with by the mechanisms we have proposed.

References

1. American Standards Association. American Standard Fortran. *X3.9-1966*, 1966.
2. J.C. Beaumont, R.J. Bradford, J.H. Davenport, and N. Phisanbut. Testing Elementary Function Identities Using CAD. *AAECC*, 18:513–543, 2007.
3. R.J. Bradford, J.H. Davenport, and C.J. Sangwin. A Comparison of Equality in Computer Algebra and Correctness in Mathematical Pedagogy. In J. Carette *et al.*, editors, *Proceedings Intelligent Computer Mathematics* (Springer Lecture Notes in Artificial Intelligence 5625), pages 75–89, 2009.
4. J. Carette. Understanding Expression Simplification. In J. Gutierrez, editor, *Proceedings ISSAC 2004*, pages 72–79, 2004.
5. L. Euler. *Elements of Algebra.* Tarquin Publications, 2006.
6. A. Gardiner. *Infinite processes, background to analysis.* Springer-Verlag, 1982.
7. B. Heeren and J. Jeuring. Canonical Forms in Interactive Exercise Assistants. In J. Carette *et al.*, editors, *Proceedings Intelligent Computer Mathematics* (Springer Lecture Notes in Artificial Intelligence 5625), pages 325–340, 2009.
8. D. Lewin and S. Vadhan. Checking polynomial identities over any field: towards a derandomization? In *Proceedings 30th. ACM Symposium on Theory of Computing*, pages 438–447, 1998.
9. R. Lipton and N. Vishnoi. Deterministic identity testing for multivariate polynomials. In *Proceedings fourteenth annual ACM-SIAM Symposium on Discrete Algorithms*, pages 756–760, 2003.

10. J. Moses. Algebraic Simplification — A Guide for the Perplexed. *Comm. ACM*, 14:527–537, 1971.
11. C.J. Sangwin. STACK: making many fine judgements rapidly. `http://web.mat.bham.ac.uk/C.J.Sangwin/Publications/2007CAME_Sangwin.pdf`, 2007.
12. J.T. Schwartz. Fast Probabilistic Algorithms for Verification of Polynomial Identities. *J. ACM*, 27:701–717, 1980.
13. R.E. Zippel. Probabilistic Algorithms for Sparse Polynomials. In *Proceedings EUROSAM 79* (Springer Lecture Notes in Computer Science 72), pages 216–226, 1979.