Some steps towards a theory of expressions

Paradox of substitution or *Error* of substitution ?
The use of x^H with H = -1 instead of fractions

Thomas Colignatus
March 6 2017

Abstract

The mathematical “theory of expressions” better be developed in a general fashion, so that it can be referred to in various applications. The paper discusses an example when there is a confusion between syntax (unevaluated) and semantics (evaluated), when substitution causes a contradiction. However, education should not wait till such a mathematical theory of expressions is fully developed. Computer algebra is sufficiently developed to support and clarify these issues. Fractions y / x can actually be abolised and replaced with $y\ x^H$ with H = -1 as a constant like e or i.

PM. Loading packages

Introduction

This text has been written as a notebook in *Mathematica,* and uses cells that have been generated by that program.

Mathematica distinguishes the presentation on-screen and presentation that is internal (FullForm). For example, what is presented on-screen in fraction form y / x, is represented internally in the power form $y * x^H$ for H = -1. This perfectly fits the suggestion by Pierre van Hiele (1909-2010) in 1973 to abolish fractions and use this multiplicative form.

```
FullForm[y / x]
```

```
Times[Power[x, -1], y]
```

The use of the value -1 might be confusing to students though, suggesting to them that they should subtract something. The formal parameter H with the value -1 should work. See Colignatus (2015) and the example in Colignatus (2017a).

The following will use an example of a numerator. When fractions are abolished then also the notions of numerator (term not under H) and denominator (term under H) would be somewhat superfluous. Yet, the real discussion below concerns the distinction between syntax and semantics.

Let us create the function Numerator[y / x] that generates y. We get Numerator[4 / 6] = 4 and Numerator[2 / 3] = 2. Can we use 4 / 6 = 2 / 3 for substitution and then find Numerator[4 / 6] = Numerator[2 / 3] too ? In that case 4 = 2, and thus we cannot do this substitution. Is this a *paradox* of substitution or an *error* of substitution ?

In common day situations (education, unsophisticated computer programs) an expression can be either *evaluated* (4 / 6 reduces to 2 / 3) or *treated as an expression itself* (4 / 6 remains as it is). Gray & Tall (1994) coined the term “procept” to identify these and other cases of ambiguity. A procept may be a strong point for flexibility in using notations in mathematics, but there is also the downside of confusion. This confusion is a problem for computer programming, since computers aren’t so flexible and then generate contradictions. Thus we need sophisticated computer programs that avoid the confusion.

To prevent confusion in *Mathematica* there are these rules on this issue:

- An expression is always evaluated unless one prevents this.
- When an expression is standing by itself, prevention of evaluation requires that the expression must be put into quotes and thus be presented as a String, or it must be contained within Hold or Unevaluated.
- When included within a larger expression with Head *h*, then *h* can have the attributes HoldAll, HoldFirst or HoldRest.

The following discusses how these rules resolve above example of the numerator of 4 / 6.

Discussion

The notion of a numerator (in fraction culture)

For 4 / 6, teachers must instruct students to factor the numerator and denominator in primes, such as (2 2) / (2 3), so that they can simplify to 2 / 3. Thus, it is important to be able to say that the numerator is 4 and that it can be factored in 2 2, and that the denominator is 6 and that it can be factored in 2 3. Thus the notions of numerator and denominator are notions of syntax (unevaluated) and not notions of semantics (reduce 4 / 6 to 2 /3).

Another example is what the numerator of (2 / 3) / 4 would be. This would evaluate to 1 / 6 and the numerator of the latter is 1. Yet, on syntax, the conventional answer would be 2 / 3. The answer to first reduce to 2 / 12 and then say 2 would be a third kind of confusion of evaluating halfway.

Curiously, though, (2 / 3) / 4 can also be written as 2 / 3 / 4, and for the latter the teacher might be inclined to hold that 2 is the numerator. Thus, brackets may not be just reading aids.

Then again, the numerator of (4 / 6) / 4 might be either 4 / 6 or 2 / 3.The syntactically correct answer would be 4 / 6 (before evaluation). Somewhat inconsistently, education tends to require that students give the simplified answer, and this would be 2 / 3. Obviously, both possible answers can be programmed.

The question "what a numerator is" depends upon choices in definitions. It would be a confusion to think that there would be some "pure" meaning. In education, (exam) questions focus on cases that are without ambiguity.

What is the numerator of 2 + 1/2 ? Should we first develop this into 5 / 2 so that the answer is 5 ? In this case, education would suggest that this is a mixed number, and hence there is no numerator. Observe that it is better to write 2 + 2^{H}.

Any *x* can also be written as *x* / 1. Should we declare that any expression like 71 or $\sqrt{2}$ without a slash or bar is also a numerator ? Or is the concept restricted to the situation that a fraction slash or bar is present ? It would be awkward to hold that having or being a numerator would be an absolute or unconditional notion.

It seems best to accept that the notion of a numerator is *conditional* to the context of a fractional expression. When *expr* has no fraction, then Numerator[*expr*] should give the answer *None*. However, *Mathematica* then generates Numerator[*expr*] = *expr*, and in itself one can accept this as a reasonable programming choice too.

The **Appendix** shows in more detail how such cases might be handled in *Mathematica*.

Dealing with expressions

In education, teachers tend to gloss over the distinction of the *"expression to be evaluated"* (semandics) and the *"expression to be treated as an expression itself"* (syntax). Apparently, students all by themselves learn to avoid such confusions. Those who don't get it disappear from class sooner.

For computer programs, this however needs close attention since they would all break down. When programs are not as sophisticated as *Mathematica* then such inconsistencies could arise indeed. Also in *Mathematica*, the programmer might deviate from the rules and create contradictions himself or herself.

Some (unspecified authors) call the above a "*paradox* of substitution". The word "paradox" means a "seeming contradiction". However, there is nothing "seeming" about this. Either there is a contradiction, as shown in above example, or there is no contradiction, as in the approach in *Mathematica*.

Thus the proper diagnosis is that it is an "*error* of substitution". The selection of the numerator of y / x conventionally looks at he unevaluated expression and not at the evaluated expression. In that case, substitution of $4 / 6 \to 2 / 3$ is a confusion and generates such contradiction.

Need for a mathematical theory of expressions

It may indeed be that the mathematical "theory of expressions" is problematic. Colignatus (2014) discusses the distinction between syntax and semantics for y / x for the case of the derivative (differential quotient). The definition of "dynamic division" or "dynamic quotient" uses the syntax of division with numerator and denominator, and it would be an error to rely on semantics and do a substitution like in the above. Colignatus (2017b) observes that the "theory of rational functions" refers to "expressions", but without much elaboration what these would be.

Conclusions

1. The notions of numerator and denominator are notions of syntax, and conditional to the presence of a fractional expression.
2. What some regard as a *paradox* of substitution actually derives from the use of unsophisticated contexts, and actually is an *error* of substitution.
3. Education better abolishes the use of fractions and mixed numbers $z + y / x$ and instead uses the format $z + y\, x^H$, with $H = -1$ that can be used as a formal parameter (like e or i).

4. The mathematical “theory of expressions” better be developed in a general fashion, so that it can be referred to in various other applications. Perhaps the team that developed *Mathematica* has found practical solutions that are not yet recognised at the theoretical level.
5. Education should not wait till such a mathematical theory of expressions is fully developed. A general approach is to use *Mathematica* in education, so that there is a natural context to discuss the distinctions on expressions as well. However, *Mathematica* is not designed for education, and thus one needs to pay attention. For students it can be enlightening however.

References

I already thanked Jan Bergstra in Colignatus (2014), and the example of substituting 4 / 6 by 2 / 3 has been given by him too.

Colignatus, Thomas (2014), “Education, division & derivative: Putting a Sky above a Field or a Meadow”, https://zenodo.org/record/292244

Colignatus, Thomas (2015), “A child wants nice and no mean numbers”, https://zeno-do.org/record/291979

Colignatus, Thomas (2017a), “Teaching Simpson’s paradox at elementary school – with H”, http-s://boycottholland.wordpress.com/2017/03/05/teaching-simpsons-paradox-at-elementary-school-with-h

Colignatus, Thomas (2017b), “A potential relation between the algebraic approach to calculus and rational functions”, https://zenodo.org/record/292247

Gray, E. & D.O. Tall (1994), “Duality, Ambiguity and Flexibility: A Proceptual View of Simple Arith-metic”, https://homepages.warwick.ac.uk/staff/David.Tall/pdfs/dot1994a-gray-jrme.pdf

Appendix

Numerator and Denominator in Mathematica

The standard functions Numerator and Denominator in *Mathematica* deviate from the convention, since they first evaluate the input before proceeding. Thus *Mathematica* actually generates numera-tor[Simplify[4 / 6]].

```
Numerator[4 / 6]
```

```
2
```

```
? Numerator
```

```
Numerator[expr] gives the numerator of expr. >>
```

While fractions and their notions of numerator and denominator can actualy be abolished, this will take some time, and it is useful to have the best tools available while it lasts. Thus the request to the makers of *Mathematica* is to create an Option EvaluateFirst -> False or Education -> True or a routine NumeratorEdu[4 / 6] that generates 4. Below we will look at some possibilities, but, in fact, the creation of such a routine is not self-evident.

The same holds for Rational:

```
Rational[4, 6]
```

$\frac{2}{3}$

```
Options[Rational]
```

{}

Screen versus FullForm

It is important to be aware of the distinction between the display of expressions on-screen and the internal FullForm that *Mathematica* uses.

```
FullForm[4 / 6]
```

Rational[2, 3]

```
FullForm["4 / 6"]
```

"4 / 6"

Interestingly, we cannot display the fraction 2 / 1 in this manner.

```
Rational[2, 1]
```

2

An expression standing by itself

There are the forms:

```
{4 / 6, "4 / 6", Hold[4 / 6], Unevaluated[4 / 6], Hold[Evaluate[4 / 6]]}
```

$\left\{\frac{2}{3}, 4 / 6, \text{Hold}\left[\frac{4}{6}\right], \text{Unevaluated}\left[\frac{4}{6}\right], \text{Hold}\left[\frac{2}{3}\right]\right\}$

```
? Hold
```

Hold[*expr*] maintains *expr* in an unevaluated form. >>

```
? Unevaluated
```

Unevaluated[*expr*] represents the unevaluated form of *expr* when it appears as the argument to a function. >>

```
? Evaluate
```

Evaluate[*expr*] causes *expr* to be evaluated even if it appears as the
argument of a function whose attributes specify that it should be held unevaluated. >>

```
? HoldForm
```

HoldForm[*expr*] prints as the expression *expr*, with *expr* maintained in an unevaluated form. >>

Expression and String

```
? ToExpression
```

ToExpression[*input*] gives the expression obtained by interpreting strings or boxes as Wolfram Language input.

ToExpression[*input*, *form*] uses interpretation rules corresponding to the specified form.

ToExpression[*input*, *form*, *h*] wraps the head *h* around the expression produced before evaluating it. >>

```
ToExpression["4 / 6"]
```

$\frac{2}{3}$

```
ToExpression[4 / 6]
```

ToExpression::esntx : Could not parse $\frac{2}{3}$ as input. >>

$Failed

The routine “FromExpression” is not needed and does not exist, since there is ToString:

```
ToString[4 / 6]
```

```
2
-
3
```

```
? ToString
```

ToString[*expr*] gives a string corresponding to the printed form of *expr* in OutputForm.

ToString[*expr*, *form*] gives the string corresponding to output in the specified form. >>

Notion of mixed number

Mathematica reduces a mixed number to a rational form:

```
2 + 1 / 2
```

$\frac{5}{2}$

The Economics Pack has this routine, and there should also be one with *H* = -1.

```
RationalHold[%]
```

$2 + \frac{1}{2}$

```
? RationalHold
```

RationalHold[expr] puts all Rational[x, y] in expr into
HoldForm[IntegerPart[expr] + RationalPart[expr]]. This is a better representation than ToFraction

An expression within a larger expression with a Head

Without HoldFirst

```
numeratorWithoutHoldFirst[x_] := Numerator[x]
```

Without HoldFIrst, the input 4 / 6 is first evaluated to 2 / 3, and only then submitted to the routine.

```
Attributes[numeratorWithoutHoldFirst]
```

{}

```
numeratorWithoutHoldFirst[4 / 6]
```

2

```
numeratorWithoutHoldFirst[4 / 6] // TracePrint
```

numeratorWithoutHoldFirst$\left(\frac{4}{6}\right)$

numeratorWithoutHoldFirst

$\frac{4}{6}$

Times

4

$\frac{1}{6}$

Power

6

-1

$\frac{1}{6}$

$\frac{4}{6}$

$\frac{2}{3}$

numeratorWithoutHoldFirst$\left(\frac{2}{3}\right)$

Numerator$\left[\frac{2}{3}\right]$

Numerator

$\frac{2}{3}$

2

2

Using String input and output

The following experimental routine has Strings as input and output. We might include a “ToExpression” in it, but let us not do so for now.

```
numeratorString::nofrac = "No fraction slash in: `1`";

numeratorString::usage = "numeratorString[\"y / x\"] puts out \"y\"";

? numeratorString
```

```
numeratorString["y / x"] puts out "y"
```

```
numeratorString[x_String] := Module[{pos},
  pos = StringPosition[x, "/"];
  If[pos === {}, Message[numeratorString::nofrac, x]; None,
   StringTake[x, pos[[1, 1]] - 1]]
 ] (* end numeratorString *)

numeratorString["4 / 6"]
```

```
4
```

```
% // FullForm
```

```
"4 "
```

The existence of a numerator is conditional to the presence of a fractional expression.

```
numeratorString["Mr Wilson"]
```

```
numeratorString::nofrac :  No fraction slash in: Mr Wilson

None
```

We seem to be saved by the bell since *Mathematica* cannot handle “(2”. However, we must adapt the routine since the output should be 2 / 3.

```
numeratorString["(2 / 3) / 4"]
```

```
(2
```

```
ToExpression[%]
```

```
ToExpression::sntxi: Incomplete expression; more input is needed .

$Failed
```

This is tricky .... but happens to be conventionally correct ...

```
numeratorString["2 / 3 / 4"]
```

```
2
```

And check the program code why it creates this funny answer.

```
numeratorString["2 + 1/2"]
```

```
2 + 1
```

```
ToExpression[%]
```

```
3
```

With HoldFirst and using FullForm

It is more convenient for the user to directly enter 4 / 6, and let the program generate the quotes. Do not program to others what you don’t want to be programmed to yourself.

```
numerator::nofrac = "No fraction slash in: `1`";

numerator::usage = "numerator[y / x] puts out y";

SetAttributes[numerator, HoldFirst];

? numerator
```

```
numerator[y / x] puts out y
```

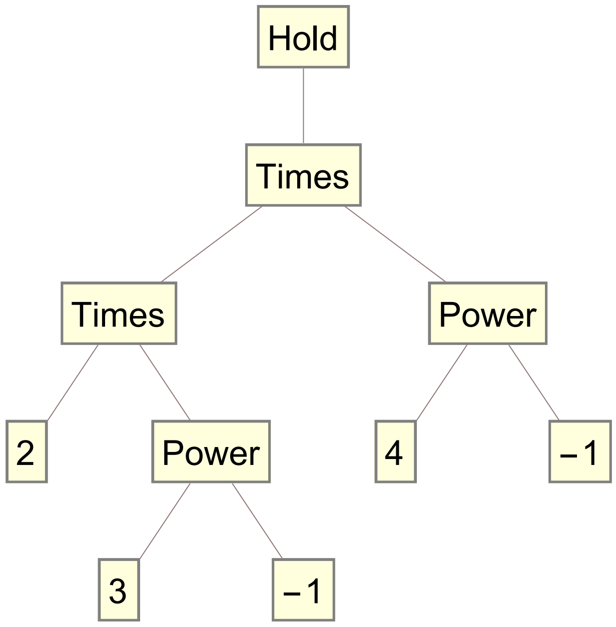
The FullForm approach seems most promising. We could locate all forms of Power[z__, -1] and use the Tree to see which is the dominant fraction.

```
numerator[x_] := Module[{expr, pos},
   expr = FullForm[Unevaluated[x]]
  ]

numerator[4 / 6]
```

```
Unevaluated[Times[4, Power[6, −1]]]
```

```
TreeForm[Hold[(2 / 3) / 4]]
```



I do not feel inclined at this moment of writing to further develop this.

With HoldFirst and returning to the internal use of Stings

Curiously, HoldFirst does actually change the expression a bit. When we return to the use of Strings, then we find the form used by *Mathematica* in 1993. The fraction bar is simulated by a separate line of dashes “-----“.

```
numerator[x_] := Module[{expr},
  expr = ToString[Unevaluated[x]]
 ]
```

```
numerator[4444 / 6666]
```

```
4444
----
6666
```

```
% // FullForm
```

```
"4444\n----\n6666"
```

To counter the change by HoldFirst, we must wrap the expression within InputForm.

```
numerator[x_] := Module[{expr},
  expr = ToString[InputForm[Unevaluated[x]]]
 ]
```

```
numerator[4 / 6]
```

```
Unevaluated[4/6]
```

This form may still be awkward to deal with, when we have more complex input.

```
numerator[Mr Wilson / (Mrs Wilson)]
```

```
Unevaluated[Mr*(Wilson/(Mrs*Wilson))]
```

```
numerator[2 / 3 / 4]
```

```
Unevaluated[2/3/4]
```

Anyway, when we wrap this around the earlier routine, then we can have non-string input and output.

```
numerator[x_] := Module[{expr},
  Results[numerator] =
   expr = ToString[InputForm[Unevaluated[x]]];
  expr = StringDrop[StringDrop[expr, 12], -1];
  ToExpression[numeratorString[expr]]
 ] (* end numerator *)
```

```
numerator[4 / 6]
```

```
4
```

```
numerator[Evaluate[Simplify[4 / 6]]]
```

```
2
```

This case gave an error above, but is now resolved by the elimination of brackets. A teacher may not be happy by this removal. (We can use the Results[numerator] to peek into what the expression actually was.)

```
numerator[(2 / 3) / 4]
```

```
2
```

```
Results[numerator]
```

```
Unevaluated[2/3/4]
```