Message from the SIGSAM Chair

Ilias S. Kotsireas
Wilfrid Laurier University
Waterloo, ON, Canada

Dear SIGSAM Members,

I would like to take this opportunity to update you on the two ACM meetings that I recently attended on behalf of SIGSAM in New York City:

- September 30, 2013, SIG Leader Orientation meeting
- October 1, 2013 SGB[1] meeting.

1 SIG Leader Orientation meeting

During the SIG Leader Orientation meeting, the participants had the chance to hear introductory remarks from Erik Altman (SGB Chair) and John White, (ACM CEO). Subsequently, Erik Altman spoke on the topic of the Volunteer Structure of the ACM. He emphasized the SIG Video Repositories initiative and focused on the example of SIGSIM that has created a 1000+ simulation-related videos on their webpage `http://www.acm-sigsim-mskr.org/Videos/videos.htm` . Among the benefits of such a repository are that it represents a good value for members, could be used to entice more people to join a SIG. In addition, the repository could use automated video mining capability. I thought this is a very interesting initiative and would like to solicit opinions from SIGSAM Members on whether we could organize a similar initiative within SIGSAM.

During the second part of the SIG Leader Orientation meeting I attended four workshops: (1) Finance, (2) Marketing/Membership, (3) Publications, (4) Information Systems. Here is a summary of some interesting aspects of these workshops:

1. ACM Tech Pack initiative `http://techpack.acm.org/` . Production of annotated bibliographies on specific areas compiled by experts. Examples include the Cloud Computing, the Parallel Computing and the Security Tech Packs.
2. ACM Learning Webinars `http://learning.acm.org/webinar/` . Started in 2012, these video presentations on topics of interest to a wide range of computing professionals are non-vendor-specific and open to everyone.
3. ACM Distinguished Speakers Program `http://dsp.acm.org/` . Potential speakers can be nominated and ACM covers the cost of transportation for the speaker to travel an event.
4. ACM Books `http://books.acm.org/` . A new ACM book series has been created, in collaboration with Morgan & Claypool Publishers based in San Francisco.

[1] SIG Governing Board

2 SIG Governing Board meeting - ACM-W

During the SGB meeting there were several interesting presentations, but i would like to focus on the presentation on **ACM-W**, `http://women.acm.org/` the ACM organization whose mission statement reads:

> ACM-W supports, celebrates, and advocates internationally for the full engagement of women in all aspects of the computing field, providing a wide range of programs and services to ACM members and working in the larger community to advance the contributions of technical women.

On of the several interesting initiatives sponsored by ACM-W is the ACM-W Athena Lectures award. Athena Lectures celebrate outstanding women researchers who have made fundamental contributions to computer science. Each year ACM will honor a preeminent woman computer scientist as the Athena Lecturer. Speakers are nominated by SIG officers. The Athena Lecturer will give a one-hour invited talk at an ACM conference determined by the speaker and the SIG which nominated her. A video of the talk will appear on the ACM website. The award includes travel expenses to the meeting and a $ 10000 honorarium. Financial support for the 2008-2009 through 2014-2015 Athena Lecturers, is being provided by Google.

I believe that SIGSAM should try to nominate one of its members for the ACM-W Athena Lectures award and would like at this point to solicit suggestions for potential qualified candidates.

The minutes of the above two meetings as well as overheads of workshop presenters can be found on-line at `http://www.acm.org/sigs/sgb/minutes` and I would encourage you to browse these materials for more information on ACM initiatives and more details on how these are potentially relevant to SIGSAM.

Ilias S. Kotsireas
SIGSAM Chair `chair_SIGSAM@acm.org`
Wilfrid Laurier University
Waterloo, ON, Canada

ACM Communications in Computer Algebra, Vol. 47, No. 4, Issue 186, December 2013

A computer algebra user interface manifesto

David R. Stoutemyer*

Abstract

Many computer algebra systems have more than 1000 built-in functions, making expertise difficult. Using mock dialog boxes, this article describes a proposed interactive general-purpose wizard for organizing optional transformations and allowing easy fine grain control over the form of the result – even by amateurs. This wizard integrates ideas including:

• flexible subexpression selection;

• complete control over the ordering of variables and commutative operands, with well-chosen defaults;

• interleaving the choice of successively less main variables with applicable function choices to provide detailed control without incurring a combinatorial number of applicable alternatives at any one level;

• quick applicability tests to reduce the listing of inapplicable transformations;

• using an organizing principle to order the alternatives in a helpful manner;

• labeling quickly-computed alternatives in dialog boxes with a preview of their results, using ellipsis elisions if necessary or helpful;

• allowing the user to retreat from a sequence of choices to explore other branches of the tree of alternatives – or to return quickly to branches already visited;

• allowing the user to accumulate more than one of the alternative forms;

• integrating direct manipulation into the wizard; and

• supporting not only the usual input-result pair mode, but also the useful alternative derivational and *in situ* replacement modes in a unified window.

1 Introduction

"*Before the Lisp machine interface to Macsyma, computer algebra was like doing mathematics encumbered by boxing gloves.*"
– Bill Gosper

I am sorry Bill, but that user interface from 1988 [18] disappeared with the Lisp machine, and its best features regrettably have not yet been implemented in any of the current most powerful computer algebra systems. Even the much earlier 1972 article [6] discusses desirable features that are still missing from modern systems.

Many computer algebra systems have more than 1000 built-in functions. Besides standard *mathematical functions* such as cos(...) and classic higher transcendental functions, built-in functions often include numerous *optional transformation functions* such as expand(...), factor(...),

*dstout at hawaii dot edu

trigExpand(...), convert(...), and simplify(...) that supplement default simplification with various transformations. Besides the expression being transformed, these transformational functions often accept optional extra arguments such as a list of variables, and/or various keyword arguments that control details such as the amount of factoring. Moreover, most computer algebra systems have numerous global *control variables* whose values help control transformations done by these functions and/or by default.

Unlike cos(...), the names and semantic details of these transformation functions and control variables are not part of any standard mathematics curriculum. Therefore it requires a long time to fully exploit such systems well, and most users never do. Moreover, these names and behaviors vary greatly between systems, making it challenging to become skilled with more than one system in order to exploit their differing capabilities. Consequently many users are frustrated because they don't know how to make any system transform an expression to a desired form.

Worse yet, often there is *no* composition of functions and/or or combination of control-variable settings capable of producing a desired form. For example, users often want expansion with respect to a certain proper subset of the variables, with polynomial coefficients that are factored with respect to the other variables – or want partial fractions with respect to a certain proper subset of some variables, with factored numerators and denominators.

This article address the usefulness of computer algebra systems as productivity tools to help amateur users accomplish their tasks without necessarily being aware of the underlying algorithms, transformation functions and their nomenclature. This article presents some ideas for enhancing the kind of wizard implemented on the Lisp Machine by:

1. more flexible subexpression selection;
2. complete control over the ordering of variables and commutative operands, with well-chosen defaults;
3. interleaving the choice of successively less main variables with applicable function choices to provide detailed control without incurring a combinatorial number of applicable alternatives at any one level;
4. quick applicability tests to reduce the listing of inapplicable transformations;
5. using an organizing principle to order the alternatives in a helpful manner;
6. labeling quickly-computed alternatives in dialog boxes with a preview of the result, using ellipsis elisions if necessary or helpful;
7. allowing the user to retreat from a sequence of choices to explore other branches of the tree of alternatives – or to return quickly to branches already visited;
8. allowing the user to accumulate more than one of the alternative forms;
9. integrating direct manipulation into the wizard; and
10. supporting not only the usual input-result mode, but also the useful alternative derivational and *in situ* replacement modes in a unified window.

Section 2 describes important features to combine in a user interface. Section 3 presents some mock examples of using the proposed wizard. Section 4 addresses design issues and their resolution, with

a summary in Section 5. The Appendix summarizes some important rational-expression transformations that should be included in addition to those discussed in subsection 4.3. The wizard must also be aware of all of the many transformations specific to irrational expressions that are built in or should be, but that is an open-ended topic too large for discussion here. However, the wizard should be implemented in an extensible way that allows users to add components easily at run time for new transformations that they implement.

This article often uses "Float" as an abbreviation for "floating-point number".

2 Important features to combine in a user interface

Many of the ideas in this section have been implemented to some extent in various computer algebra systems, but integrating them into a uniformly designed user interface could greatly enhance the user experience over that of any one current system.

Computer algebra often generates large expressions, and with current RAM sizes measured in gigabytes, screen area is now the most precious resource for most user's tasks.[1] Many of the ideas discussed here are concerned with attempting to make the best use of that limited resource to help amateur and expert users arrive quickly at the most comprehensible alternative output forms.

2.1 Enhancing the Lisp Machine Macsyma precedent

Among the most helpful features of Lisp Machine Macsyma was that as you moved the mouse over an expression, the minimal rectangle containing a syntactically complete subexpression surrounding the mouse pointer would automatically be framed, which is perhaps the entire expression. A right click would open a drop-down menu of common transformations such as factor and expand, or you could enter a function name of your own. The selected transformation is applied to the framed subexpression.

This feature could be regarded as a *wizard* that helped users quickly locate appropriate subexpressions for desired transformations, then apply them to those subexpressions. This is important because without subexpression selection and shortcuts for applying a desired transformations to selected subexpressions, it is painful for even expert users to force large expressions into anything near the form they would prefer – death by a thousand cuts and pastes.

Part of the pain is carefully reassembling a final result from independently transformed subexpressions, then carefully deleting the distracting debris of all the intermediate steps.

Even merely ordering commutative operands as desired is difficult or impractical in most computer algebra systems. For example, it is a constant irritant to be unable to transform a result such as $E = c^2 m$ to $E = mc^2$ or, better yet, to prearrange that it will automatically be ordered as desired.

2.2 Including direct manipulation

Direct manipulation provides a complementary way that major computer algebra systems could make their user interfaces more helpful: With Milo [2] or Theorist [7] you could use the mouse to select a term or a factor, then drag and drop it to

- reorder terms and factors,

[1] The maximum number of legible characters simultaneously legible on multiple high-resolution screens or sheets of paper is unimprovable.

- distribute a term over a factor,
- factor out a common factor from a sum of terms,
- transpose factors or terms from one side of an equation to the other.

The selected subexpression could also be dropped into a variable in another expression to substitute the subexpression for every instance of the variable in that other expression. Also, expressions could optionally be compressed by automatic or mouse-driven temporary replacement of subexpressions with ellipses.

Here is a temporal sequence of Milo snapshots for dragging x successively further right in an equation [16]:

$$\frac{\boxed{\text{x}}\text{y}}{2}+\text{x}=1 \quad\rightarrow\quad \frac{\text{y}\boxed{\text{x}}}{2}+\text{x}=1 \quad\rightarrow\quad \frac{\text{y}}{2}\boxed{\text{x}}+\text{x}=1 \quad\rightarrow\quad \left(\frac{\text{y}}{2}+1\right)\boxed{\text{x}}=1 \quad\rightarrow\quad \frac{\text{y}}{2}+1=\frac{1}{\text{x}}$$

Milo evolved to The Plotting Calculator, which is still available and supported [3]. The most recent version of Theorist is named LiveMath$^{\text{tm}}$, which is also still available and supported [19]. Both are oriented toward mathematics through calculus, but direct manipulation should also be implemented in other computer algebra systems, integrated with the transformation wizard proposed here: After selecting subexpressions, one of the transformation options, if applicable, should be "drag and drop".

2.3 Collecting multiple alternatives

The wizard generates and displays the results of alternative transformations as the user explores a tree of successive applicable transformations. The user can accumulate any number of these alternative results into a list that is returned as the result if the user wants more than one. For example, as users interactively view alternative factored and partial fraction forms for a rational expression, they can indicate which ones they want included in a returned list of alternatives. This is inspired by Wolfram|Alpha [38], which automatically returns multiple alternative results. The difference here is that the user can participate in a more thorough exploration of the possible alternatives and select only those of interest.

2.4 Input-result pairs *versus* derivation steps *versus* replacement

Most industrial-strength computer algebra systems use only the *input-result* mode. This mode is particularly appropriate when the goal is to obtain good final results in as few steps as is practical.

Some mathematics education programs such as Mathpert$^{\text{tm}}$ and the SMG application for some TI computer algebra products use the *derivation* mode wherein the input is transformed to a result by selecting successive subexpressions and choosing transformations from menus, with the annotated result of each step displayed beginning on a separate line. This mode is also often used in theorem proving software [29, 37]. This mode is also good for expository use by professionals when they want to explain in a presentation or publication how a result is derived. For example, the multi-step derivational style is used several times in this article.

Some programs such as The Graphing Calculator offer the *replacement* mode wherein selected transformations replace the selected subexpressions *in situ*. This mode has the advantage of conserving screen space by minimizing the amount of debris – at the expense of not being able to view the input and result simultaneously.

An industrial-strength computer-algebra system should offer all three modes. The following observations can justify allowing mixtures of all three modes in a single session window and name-space context:

- An input-result pair can be regarded as a one-step instance of the derivation mode.
- A one step *in situ* replacement could be labeled with a two-button setter bar such as 

that toggles between the two.
- A multi-step *in situ* replacement could have a slider bar between these two endpoints, and perhaps also a Play button that does a slide show or an animation.
- A right click could offer the option of changing previous computations between these three modes, such as
  - collapsing a derivation sequence to an input-result pair or to an *in situ* replacement,
  - expanding an input-result pair to a refinable derivation sequence that was automatically used to create it.

The model-view-controller paradigm is a good way to achieve this multi-view software design [36].

2.5 {Undo^m, redo^n}

Anyone who has used software with a well designed essentially unlimited undo-redo capability knows how aggravating it is to return to software that offers only one step of undo – perhaps with no redo. With current RAM capacities measured in gigabytes there is no excuse for this. Internet browsing has familiarized users with using the "←" and "→" buttons together with a drop-down browse history list to revisit easily throughout the tree of past web page visitations. The wizard can use the same techniques and temporarily save all recent closed dialog boxes for quick regeneration.

2.6 Dynamically created dialog boxes specialized for the example

The variety of mathematics examples is so great that general-purpose dialog boxes created when a computer algebra system is built would be unpleasantly cumbersome to use:

- They would have numerous distracting grayed-out controls.
- They would entail numerous subsidiary dialog boxes to accommodate all of the inapplicable entries without making each dialog box unreasonably large.
- They would contain lengthy or awkward wording such as "variable or variables" or "variable(s)" to correctly accommodate both singular and plural cases without distracting grammatical errors.

Thus custom dialog boxes specialized to the framed subexpression must be created at run time.

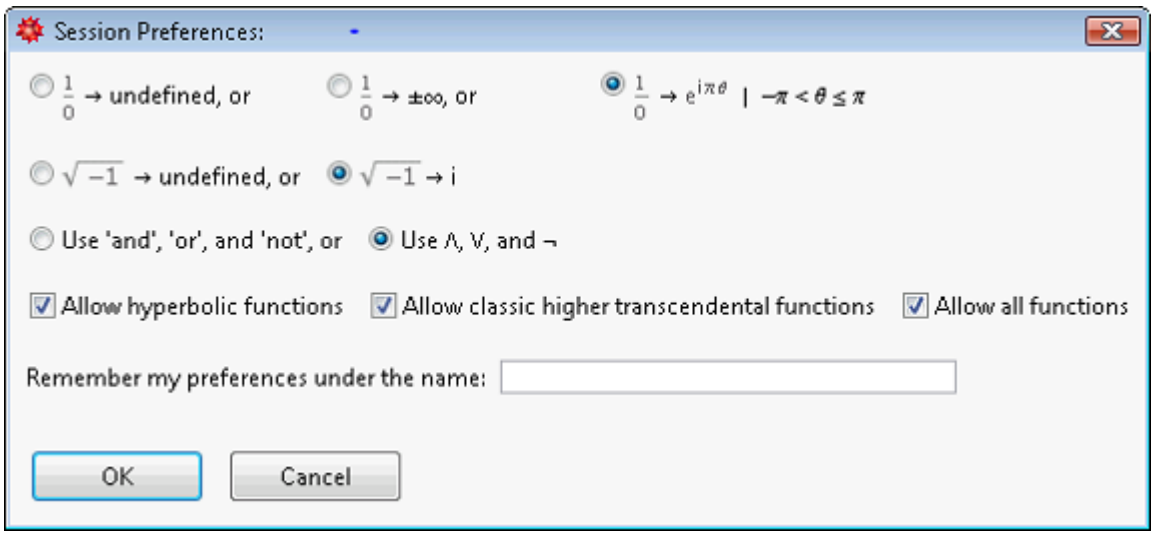
2.7 Adapt to the user's level and goals

Computer algebra is being used by students from beginning secondary school algebra through graduate-level mathematics – and by professional mathematicians, scientists, engineers, economists etc. at many different levels of sophistication. The number of potential users declines rapidly with increasing mathematics level. However, most computer algebra systems are designed for the higher levels of this spectrum. Consequently the most powerful general-purpose systems are quite daunting to most potential users. For example, in many courses from secondary school algebra through university real-variable calculus:

1. Many students know nothing about hyperbolic functions, higher transcendental functions and hypergeometric functions.
2. Most students have not encountered many standard mathematics symbols and notations such as $\exists$, $\forall$, $\neg$, $\vee$, $\wedge$, $\Re$, $\Im$, $\mathbb{Z}$ and $\mathbb{Q}$.
3. Most students know nothing about terminology such as algebraic groups, rings, fields, ideals, varieties, and square-free factorization.
4. The expression $1/0$ is usually or always regarded as undefined rather than as $\pm\infty$ or a circle of infinite radius in the complex plane.
5. The expression $\sqrt{-1}$ is usually or always regarded as undefined rather than i.
6. The expression $(-1)^{1/3}$ is usually taken to mean -1 rather than $1/2 + i\sqrt{3}/2$.

The mathematically weakest students and professionals who could most benefit from computer algebra are most intimidated by the appearance of such unknown function names, symbols and nomenclature in their dialog boxes and results. Often this intimidation and the consequent loss of self esteem terminates receptivity to learning effective use of the computer algebra system.

In computer aided instruction there are efforts to automatically infer the level and overall goals of users, then adapt the interface accordingly. Those techniques are not explored in this article, and the termination of receptivity might occur before enough input occurs to make an accurate inference. However, one easy way to accomplish many of the benefits of such customization is for the first dialog box of a session to have a button labeled "Session preferences" that opens a dialog such as the following if pressed:



The defaults should be those of the computer algebra system, but the more elementary alternatives should appear first in each row of alternatives to minimize alarming elementary users.

Another complementary alternative to matching student mathematical level is to enable the easy creation of named shell programs and their icons that launch the computer algebra system then immediately set appropriate preferences automatically. An instructor can then create such shells named, for example, MapleForAlgebra1 or MathematicaForCalculus1.

3 Examples of using the wizard

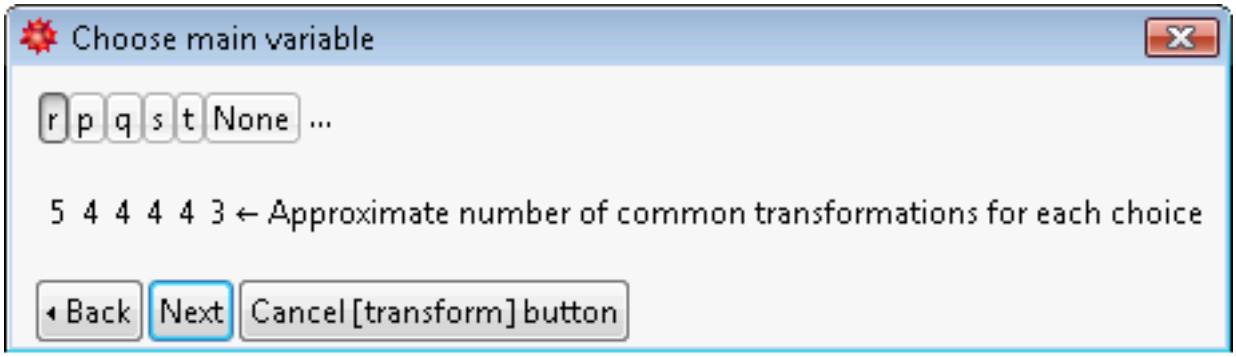
This section contains examples of using the proposed wizard.

3.1 Simplifying an ugly expression

The *Mathematica* CreateDialog[...] function is a convenient way to create new dialog boxes at run time. Thus I used CreateDialog[...] to create dialog boxes that are appropriate for specific examples, without bothering to attach these boxes to each other or to any *Mathematica* transformation functions. The mock intermediate and final results are not that of any particular current computer algebra system, but rather what I wish they would produce – especially with regard to ordering of factors and terms. For example, suppose that *either an input or a result* of previous steps is

$$\begin{gathered}(p^2qr^3+p^2qr^2s+p^2qr^2t+p^2qrst+p^2r^4+p^2r^3s+p^2r^3t+p^2r^2st+pqr^4+pqr^3s+pqr^3t+pqr^2st+\\ 2pqrs+2pqrt+4pqr+4pqst+2pqs+2pqt+pr^5+pr^4s+pr^4t+pr^3st+pr^2s+2pqrs+\\ 2pqrt+4pqr+4pqst+2pqs+2pqt+pr^5+pr^4s+pr^4t+pr^3st+pr^2s+pr^2t+2pr^2+\\ 2prst+2prs+2prt+2pr+2pst+ps+pt+qr^2s+qr^2t+2qr^2+2qrst+2qrs+2qrt+\\ 2qr+2qst+qs+qt+2r^2s+2r^2t+4r^2+4rst+2rs+2rt)/\\ (pqr^2+pqrs+pqrt+pqst+pr^3+pr^2s+pr^2t+prst+qr^3+qr^2s+qr^2t+qrst+r^4+r^3s+r^3t+r^2st).\end{gathered} \tag{1}$$

I have no *particular* goal form in mind, but I would like a result that is more concise and comprehensible – and more efficient for substitution of numbers. I position the mouse pointer between the left margin and the expression, thus framing the entire expression, then right click and choose "transform".[2] This opens the following dialog box courteously positioned just above the subexpression if the subexpression is low on the screen, or just below the subexpression otherwise[3]:

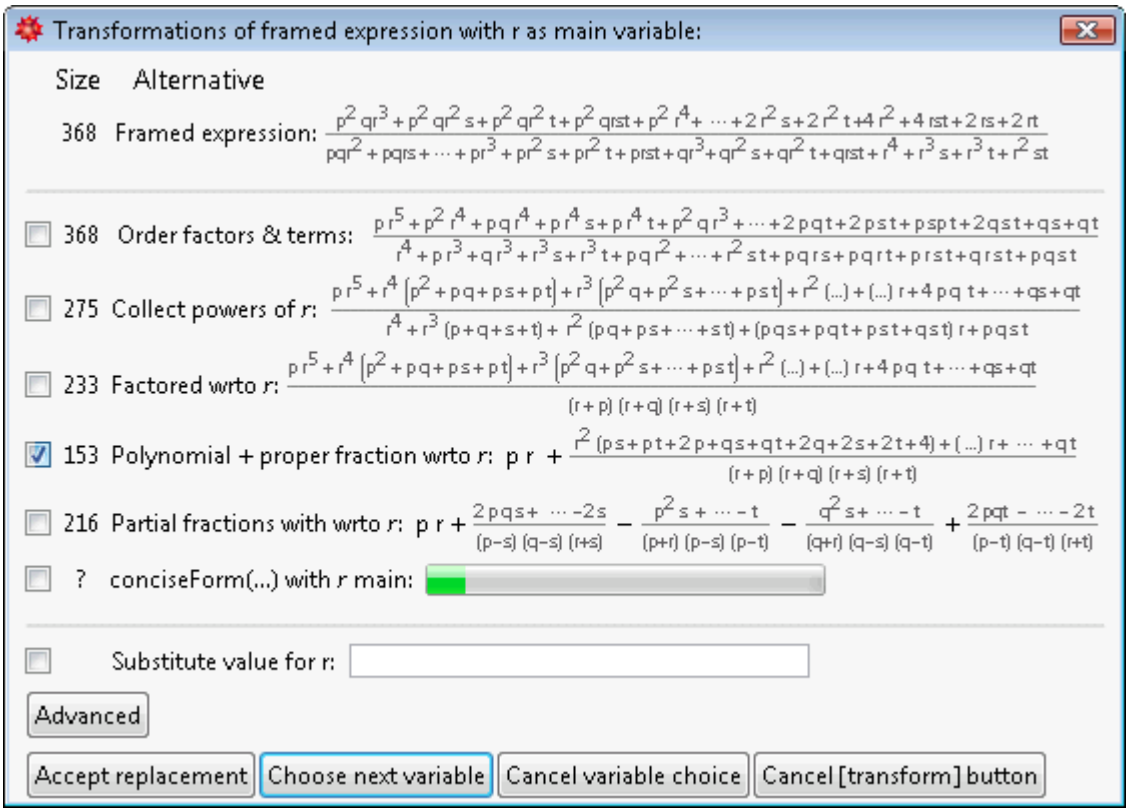


[2] Alternatively, I can choose "Transform ..." from the main menu bar or click Transform on the main toolbar.

[3] I hate it when a dialog box initially covers the information I need to respond to it!

Some transformations such as expansion of an improper ratio to a polynomial plus a proper ratio require a designated variable. The None button considers only transformations that do not require such a variable, such as factoring or *polynomial* expansion with respect to *all* variables. With that choice, the variables would be ordered according to the analysis in [20].

The variables are listed in non-increasing order of an estimated number of applicable common transformations because the choice of main variable tends to have the greatest influence on the overall form of the result. Choosing a main variable for which there are few alternative forms tends to narrow the choices more than otherwise. For example, if we chose p, q, s or t as the main variable, then there would probably be fewer than 5 common alternatives for r thereafter. Thus button r was initialized to *pressed* to encourage lazy users such as me to accept it, which I do. This opens the following dialog box:



"*A lot of times, people don't know what they want until you show it to them.*"
– Steve Jobs

The Advanced button lists alternatives that would interest most users only occasionally for this example – alternatives such as continued fractions, Hornerization, expression in terms of Chebyshev polynomials, or series approximations.

The displayed *sizes* are some easily computed measure that correlates approximately with the relative area that would be required to display the entire alternative results. The initially checked boxes are those having the smallest size.

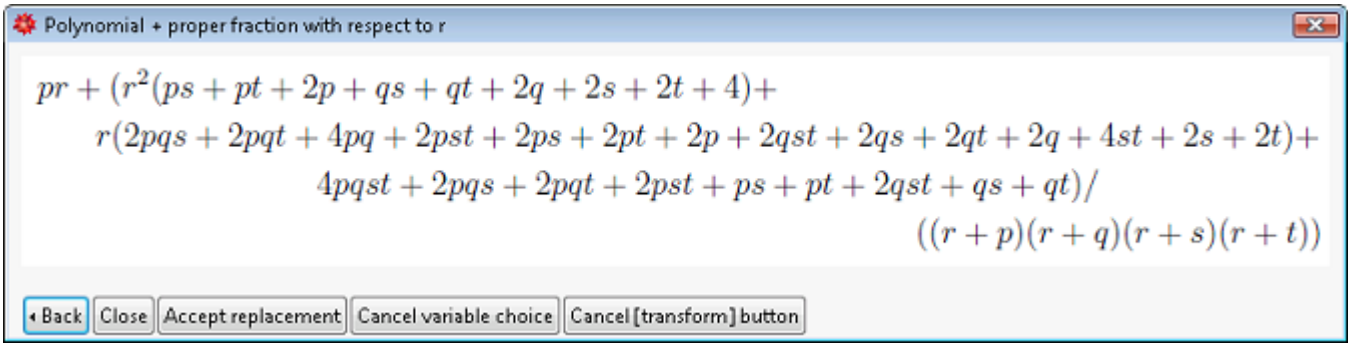
The dialog box shows alternative results for all the alternatives that can be computed in a *total* of at most 0.1 seconds – with elisions if necessary to avoid scroll bars or using the entire screen. User interface designers [23] feel that maximum acceptable response times are:

- about 0.1 seconds for responses to a mouse click, key press, or anything involving hand-eye coordination;
- about 1 second for opening a progress indicator, closing a dialog box or reformatting a table;
- about 10 seconds for everything else, including displaying a graph or completing an understandably time-consuming task. This "mind begins to wander" threshold is not always achievable with computer algebra.

The *Mathematica* Collect[. . . , r], Factor[. . .], PolynomialQuotient[. . . , . . . , r], PolynomialRemainder[. . . , . . . , r] and Apart[. . . , r] functions require a total of only 0.04 seconds on a dual-core 1.6 gigahertz computer to compute *Mathematica*-ordered versions of the five initially-displayed results in this dialog box. Therefore this dialog box could be created and displayed in an acceptable amount of time.

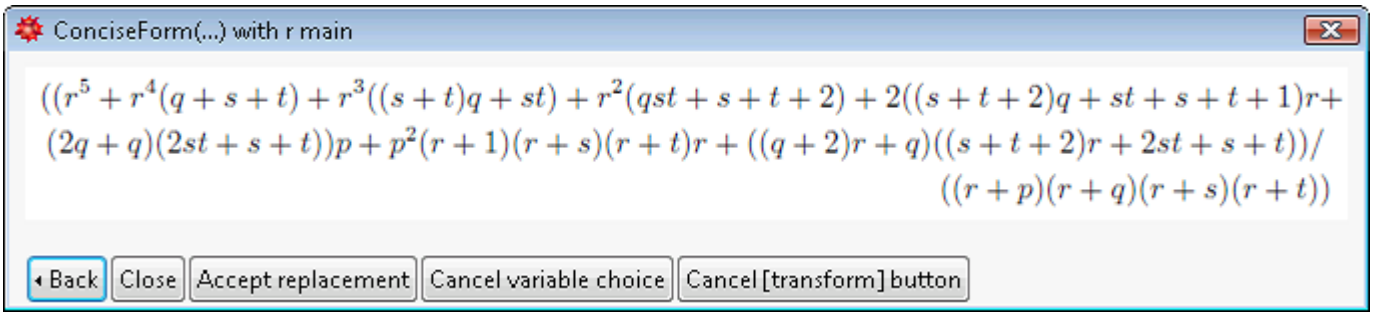
The progress bar for the conciseForm(. . .) alternative appears within one second, indicating that it is still being computed after the initial display of the dialog box. If that function doesn't post progress messages, then a slug cyclically moves from left to right to let the user know that the computer is working rather than merely awaiting user input.[4] The conciseForm(. . .) alternative represents the system's most powerful general-purpose simplification function such as the *Mathematica* FullSimplify[. . .] function, which requires 2.4 seconds for this example. This is way less than the time it requires for me to compare the five alternatives above conciseForm(. . .) with the Framed expression. As such continuing computations complete, their results are displayed in the dialog box – elided if necessary. They are listed below initially presented results to reduce displacement distraction when they complete. The conciseForm(. . .) alternative completes with a size of 125, which is the smallest, so that check box *also* becomes checked if I haven't already pushed a button.

Every time a check box is checked that has not been checked before, if the corresponding result is elided, then another dialog box opens that merely displays the un-elided alternative result, with scroll bars if necessary. For example, the initial "Polynomial + proper fraction wrt r" choice opens the dialog box



and when conciseForm(. . .) completes, it opens the dialog box

[4]It is not yet customary to have computer algebra functions post progress messages, but there are obvious candidate events for some algorithms. For example, many algorithms for degree n or for n variables, terms, factors, equations or columns process them one at a time, permitting messages of the form "$1/n$ % done", "$2/n$ % done", ... even if the time spent for each such step is likely to be rather uneven. People are comforted by progress bars even when they are inaccurate.



Both alternatives are significant improvements over the original framed expression (1), but both numerators are still lengthy with no easily discerned pattern. The reasons for the factored denominator in "Polynomial + proper fraction" are:

- The factored denominator was already computed for the alternative "Factored wrt r".
- The factored denominator is much more compact and informative than the fully expanded original denominator.
- There is nothing in the phrase "Polynomial + partial fraction" that promises displaying the denominator expanded with respect to r that was used to compute the polynomial part and the numerator.
- It is easy to frame the factored denominator then expand it if desired.

Although the ConciseForm result is slightly more compact, perhaps I could improve the numerator of the proper fraction result because I requested no more than a polynomial plus a proper fraction, and the quickest path to that goal was to expend no extra effort on the numerator beyond the collection with respect to the main variable r that was already done. Therefore in either the dialog box that contains the elided or the complete version of the proper fraction, I frame the entire numerator, right click, then choose "transform". This recursively opens up a new dialog box to choose a main variable, for which I again choose r for consistency. The resulting displayed alternatives include the factored numerator

$$(r(p + q + 2) + 2pq + p + q)\,(r(s + t + 2) + 2st + s + t)\,.$$

This is much more compact, with insightful symmetries $p \leftrightarrow q$, $s \leftrightarrow t$ and $[p, q] \leftrightarrow [s, t]$ that are also true of the denominator. Therefore I accept this replacement in this *sub-problem* that is an alteration of the "Polynomial + proper fraction" alternative, thus transforming the expression in that dialog box to

$$pr + \frac{(r(p + q + 2) + 2pq + p + q)\,(r(s + t + 2) + 2st + s + t)}{(r + p)(r + q)(r + s)(r + t)}. \tag{2}$$

This is the nicest overall result so far, but before accepting it, I notice that although every factor contains r, the first numerator factor and the first two denominator factors are free of s and t, whereas the second numerator factor and the last two denominator factors are free of p and q. From experience I know that for two ratios having disjoint variable sets or nearly so, common denominators almost always increase bulk because there can be very little cancellation in the resulting numerator.

Thus conversely, partitioning the ratio in (2) into a ratio containing $\{r, p, q\}$ and a ratio containing $\{r, s, t\}$ then transforming each ratio to partial fractions *might* reduce bulk. Consequently, I drag the first numerator factor left of the ratio giving

$$pr + (r(p+q+2) + 2pq + p + q)\,\frac{(r(s+t+2) + 2st + s + t)}{(r+p)(r+q)(r+s)(r+t)}.$$

Then I drag the first two denominator factors under the former numerator factor giving

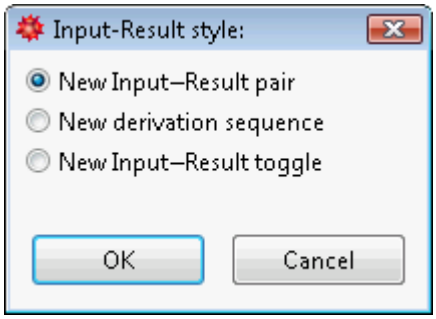
$$pr + \left(\frac{r(p+q+2) + 2pq + p + q}{(r+p)(r+q)}\right)\left(\frac{r(s+t+2) + 2st + s + t}{(r+s)(r+t)}\right). \tag{3}$$

(With the ability to select several non-adjacent subexpressions, I could instead select in (2) the first factor of the numerator and the first two factors of the denominator, then right click, then choose an action named something such as "collect", "group" or "isolate" – or perhaps directly choose partial fractions.)

Next I highlight the left ratio in (3), choose r as the main variable, then accept partial fraction expansion with respect to r, then do similarly for the right factor, giving

$$pr + \left(\frac{p+1}{r+p} + \frac{q+1}{r+q}\right)\left(\frac{s+1}{r+s} + \frac{t+1}{r+t}\right). \tag{4}$$

This is a very gratifying result compared to the equivalent input (1), and I am happy with the ordering of the terms and factors. Therefore I press the Accept result button, which opens the dialog



If I choose "New Input-Result pair", then the main computer algebra session window would be updated with a numbered input such as $Input_4$: Transform($Result_3, \ldots$) followed by expression (4) preceded by a label such as $Result_4$. The purpose of the Input line is to capture a programmatic way to transform $Result_3$ to $Result_4$ for purposes such as scripting. Most users will not want to view the ugly details, but if I click on the ellipsis in Transform($Result_3, \ldots$), then it expands to a procedure that generates $Result_4$, such as

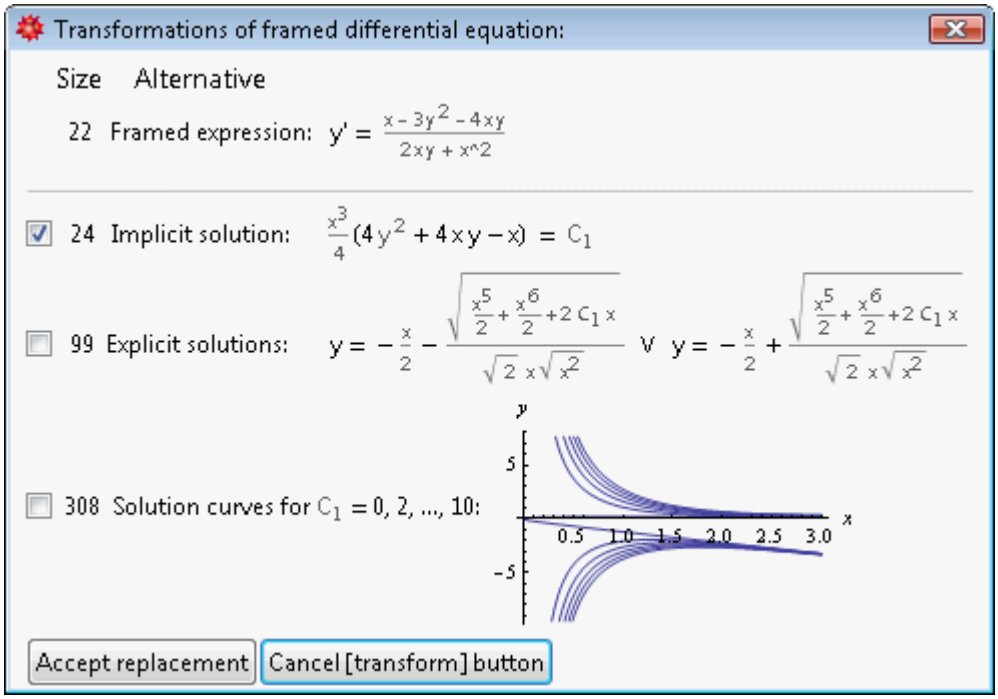
```
Block(Local(temp1,temp2,temp3,temp4),
  temp1 := PolynomialPlusProperFraction(Result3, r),
  temp2 := Factor(Numerator(Second(temp1)), r),
  temp3 := Factor(Denominator(Second(temp1)), r),
  First(temp1) +
    PartialFractions(First(temp2)/(First(temp3)*Second(temp3)), r)*
    PartialFractions(Second(temp2)/(Third(temp3)*Fourth(temp3)), r));
```

The "New derivation sequence" choice is similar, except it generates a collapsible sequence of such pairs using labels such as $\text{Intermediate}_{4,1}$, $\text{Intermediate}_{4,2}$, ...

The "New Input-Result toggle" is similar to the "New derivation sequence", except listing a single expression labeled with a two-button setter bar.

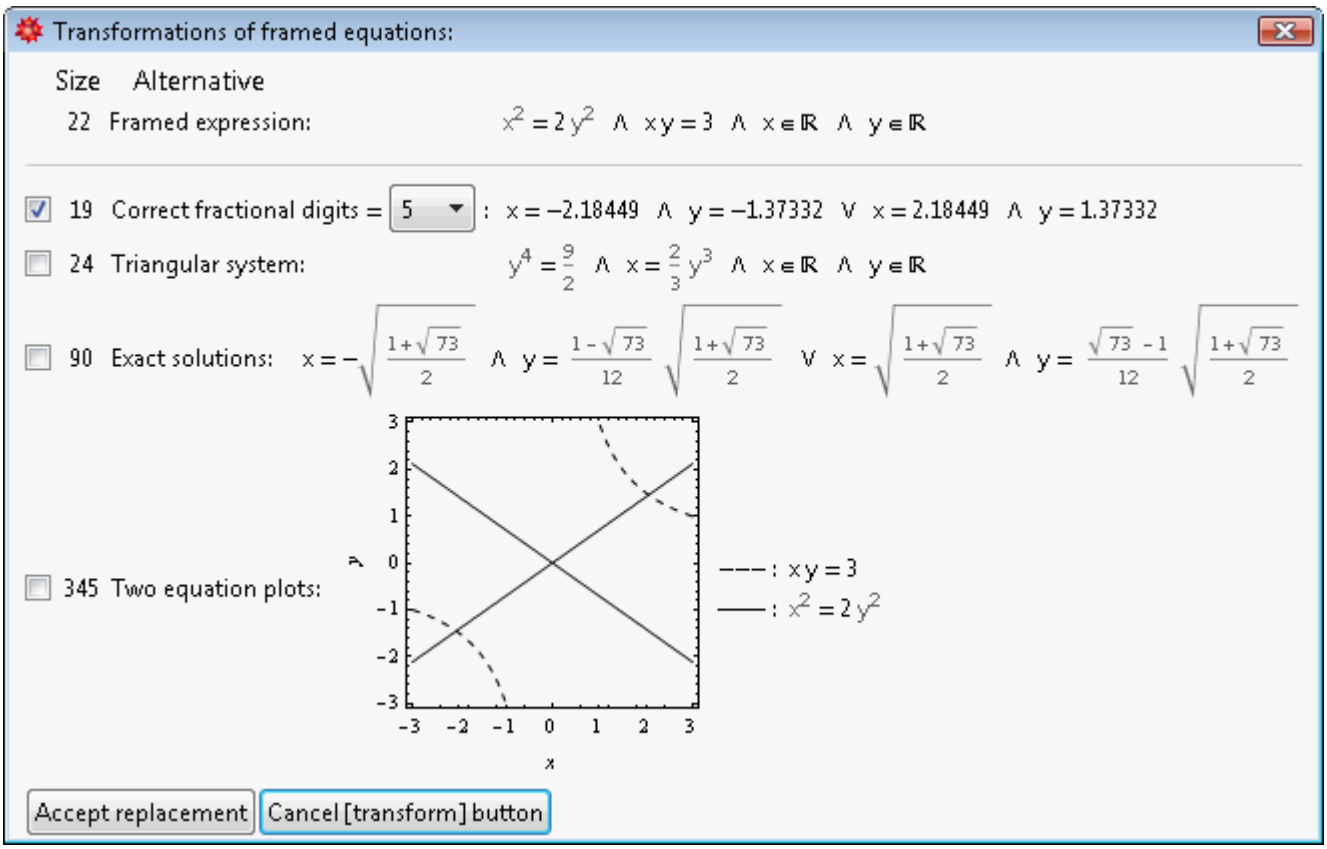
3.2 Transforming equations, inequalities and Boolean expressions

The primary activity that users want to do with equations, inequalities and systems thereof is to transform them into explicit solutions, so why force users to learn numerous different function names with different parameter semantics for solving different kinds of equations? If the user clicks the Transform button when an entire equation, inequality or system thereof is framed, then the wizard tries to return solutions. Here is an example for a differential equation:



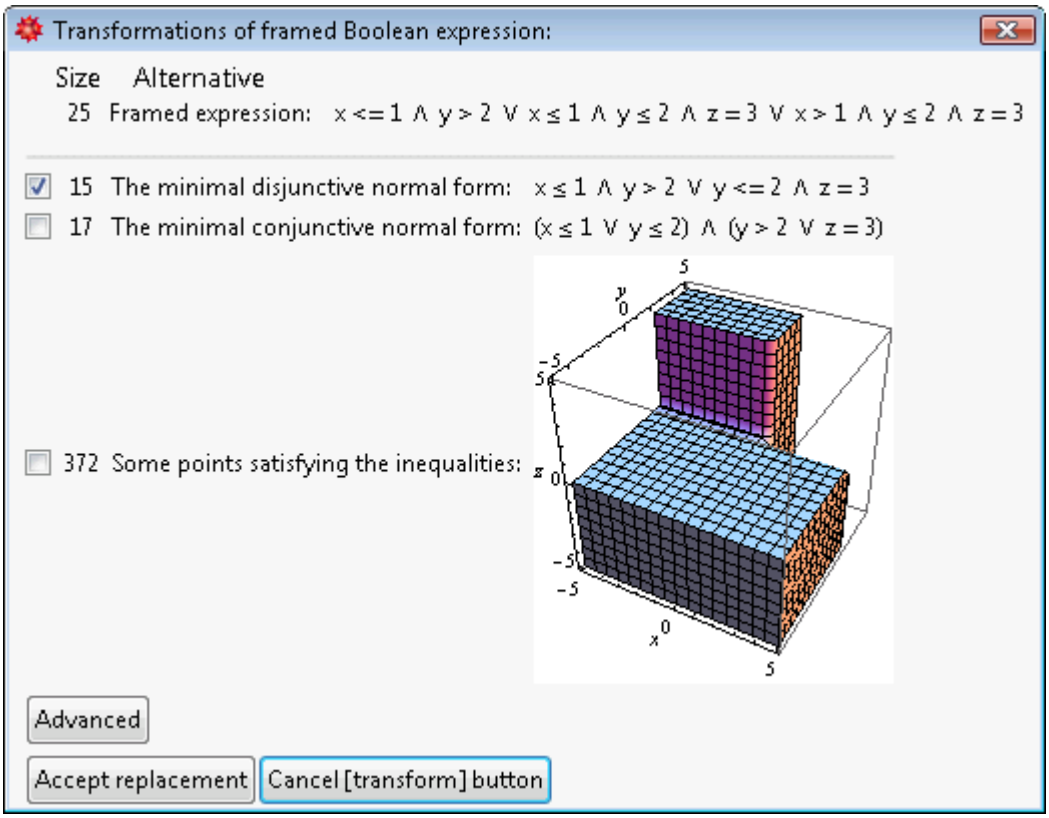
Notice the synergy of multiple views of the solutions. When the mouse pointer is over a curve, it is attached to a call-out displaying the corresponding explicit solution containing the associated numeric value for C_1. The plot range for x and the set of values for C_1 were chosen to insure that y is real. Checking this alternative opens a dialog with a magnified plot that gives the user control over the plot ranges for x and y together with the set of numeric values for C_1. The listed size of 308 is correlated with the area of the plot in the session window if accepted.

Here is an example for a system of two nonlinear algebraic equations:



- The approximate solution was computed with *adaptive precision interval arithmetic* to deliver guaranteed requested accuracy initially set to correspond to six significant digits for nonzero components. If interval arithmetic is inapplicable or is taking too long for the specified accuracy, then the wizard tries *adaptive significance arithmetic*. If that is also taking too long, the wizard switches the popup digits setting to "IEEE" double and uses that. The corresponding displayed phrases are "Estimated correct fractional digits" or "Approximate solutions of unknown accuracy".
- The triangularized system is a reduced lexicographic Gröbner basis, which might be the preferred alternative for parametrized systems having exact explicit solutions that are messy or require unendurable time to complete. If the system included inequalities, then there would be a cylindrical algebraic decomposition instead.
- The x and y ranges in the plot were automatically set to include a margin around the convex hull of all the isolated finite solutions – a margin small enough to resolve detail near the solutions but large enough to provide useful context.

Here is an example of transforming a Boolean expression:



The Advanced button lists alternatives expressed in terms of nands, nors, etc.

3.3 The wizard is helpful even for mere numbers

Common useful internal representations for exact numbers are rational numbers and irrational constant expressions such as $\sqrt{\pi} + \ln 2$. Common representations for approximate numbers are:

1. software variable-precision Floats – preferably with adaptive significance tracking,
2. IEEE double Floats to take advantage of fast hardware instructions;
3. intervals whose endpoints are each independently an exact rational number or a Float – preferably adaptive precision for floating-point endpoints, and allowing a disjoint union of such intervals with either open or closed endpoints.

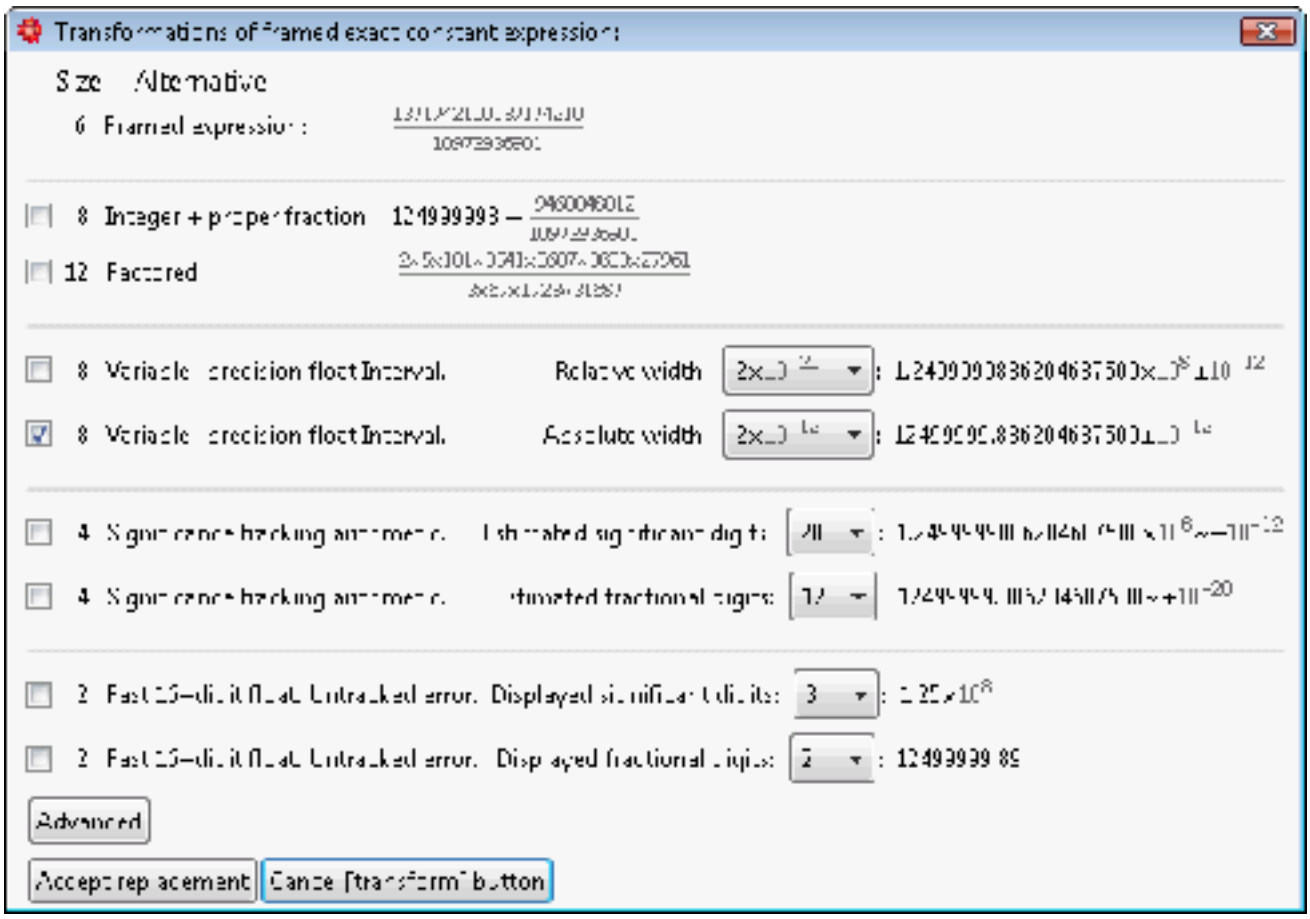
Before commencing a computation a user might want to transform from an exact to an approximate representation to make the computation faster – or from an approximate to an exact representation to avoid rounding errors. *After* a computation a user might want to convert from an approximate to an exact representation to attempt recovering an exact result – or from an exact to an approximate representation to make lengthy exact numbers more comprehensible or faster for purposes such as plotting. Also, at the end of a computation a user might want to simply alter the *display* of a number, such as displaying an exact rational number factored or as an integer plus a proper fraction or as a decimal fraction or in scientific form with a particular number of fractional or significant digits. The wizard makes it easy to do these transformations.

3.3.1 Alternate forms for rational numbers

If the framed subexpression is

$$\boxed{\frac{1371742100137174210}{10973936901}},$$

then the wizard could offer the dialog box



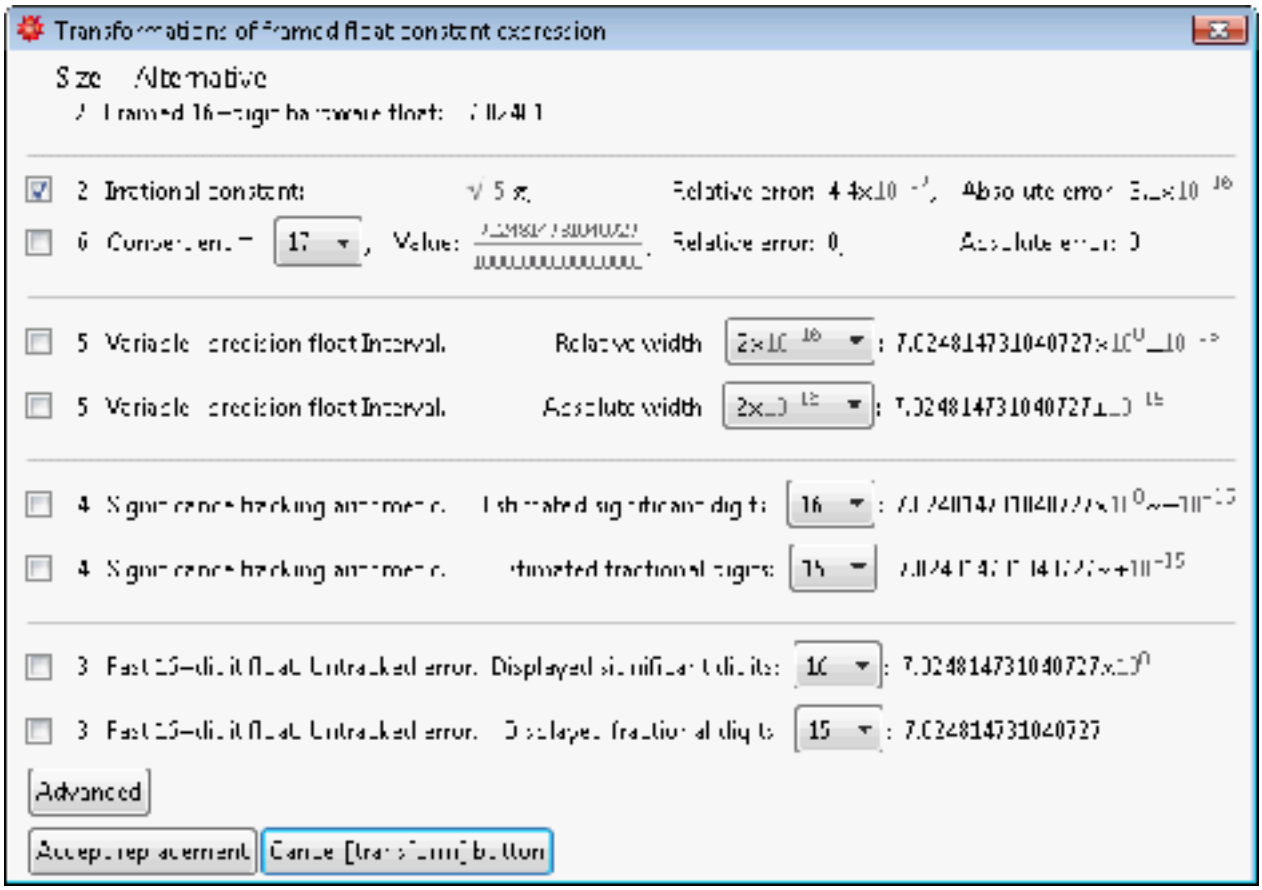
In accordance with the recommendations of [27], the approximate alternatives are ordered from intervals through bare IEEE Floats to encourage more use of arithmetic that is closer in spirit to exact arithmetic. The Advanced button could offer p-adic, continued fraction, and different radix representations.[5]

Notice that this dialog provides useful supplementary information about the framed number even if the user never intended to replace the framed number: The user now has a good estimate for its magnitude, can see that it is well approximated by 1.25×10^8, and that both the numerator and denominator are composite but square free.

[5]In most systems, default simplification would immediately transform a factored or continued fraction or integer plus proper fraction form of a rational number back to a reduced ratio. Thus special passivity is required to make such volatile forms appear in results, and such non-idempotent forms are quite likely to disappear when such results are used in subsequent inputs. For this reason, many systems return a factored rational number as a list of pairs of bases and exponents, etc. The wizard must correct for such impediments to direct substitution of transformed subexpressions into the expression from which it came, suppressing default simplification where necessary to preserve the displayed overall result in standard mathematical form.

3.3.2 Alternate forms for intervals and Floats

If the framed subexpression is an interval, an IEEE double or a variable-precision Float, then the wizard could offer alternatives including approximating the number with an exact rational or irrational constant. For example, if the framed number was the IEEE double 7.024814731040727 or a reasonably close approximation to it, albeit perhaps *displayed* with fewer significant digits, then the wizard could offer



Details matter. For example:

- The alternate expressions are aligned, if practical, to make it easier to compare them.
- If the framed Float displayed few digits, the initial displayed digits for alternatives displays all or many digits – and *vice versa*.

The delightful alternative $\sqrt{5}\,\pi$ was computed quickly by the Maple identify(...) function, for which there is a more powerful free stand-alone version on the internet [5].

3.3.3 Alternate forms for non-real numbers

For non-real numbers, which of *rectangular*, *unit polar* and *exponential polar* form is most attractive depends on the particular number. For example, compare

Rectangular		*Unit polar*		*Exponential polar*
$\mathbf{7+5i}$	=	$(-1)^{\arctan(5/7)/\pi}$	=	$\sqrt{74}\,e^{\arctan(5/7)\,i}$,
$-2\sin\left(\frac{3\pi}{14}\right)+2\cos\left(\frac{3\pi}{14}\right)i$	=	$\mathbf{2(-1)^{5/7}}$	=	$2e^{5\pi i/7}$,
$2\cos(1)+2\sin(1)\,i$	=	$2\,(-1)^{1/\pi}$	=	$\mathbf{2e^{i}}$.

Default simplification would ideally *display* the form that is most concise for each non-real number in a result even if a different form is used internally. However, optional transformations can conveniently offer all three alternatives.

4 Design issues and their resolutions

Challenging design issues include deciding:

1. What set of transformations should the wizard consider?
2. How can the wizard quickly estimate the number of applicable transformations without knowing the next variable choice?
3. When the next variable choice is known or None, how can the wizard quickly determine what subset of transformations are applicable and schedule them so that a worthwhile number are completed quickly?

This section addresses these issues and a few others. However, there are so many transformations that the wizard should know about that this section concentrates on those that help resolve issues 2 and 3. For more completeness, the Appendix discusses additional transformations that are relevant to the rational aspects of expressions. Transformations of the *irrational* aspects of expressions is too large a topic for treatment in this article.

First, three simple definitions:

Definition. *Default simplification* is the result of pressing the Enter key or else perhaps Shift Enter with the factory-default mode settings and no transformational or simplification functions anywhere in the input expression.

Definition. A *functional form* is an expression of the form

$$f\,(\mathit{expression}_1,\ \mathit{expression}_2,\ \ldots)$$

where f is any function name.

Definition. *Generalized variables* are the smallest subtrees of an expression tree that are not a sum, difference, product, ratio, rational number, Float, or reasonably regardable as an integer power.

For example, z, π, i, $\cos(x+f(2))$, $z^{1/5}$, and $3^{1/5}$ are generalized variables. In contrast, $3/4$, x/y, and $x-3$ are not. Also, $z^{2/5}$ and $3^{2/5}$ are not generalized variables, because they can be regarded as $(z^{1/5})^2$ and $(3^{1/5})^2$.

The importance of generalized variables is that transformations that are applicable with respect to variables in an expression can also be applicable with respect to generalized variables in an expression. For example, a user might want ordering, expansion or factoring with respect to to π and or $\cos(x+f(2))$. Additional transformations might be applicable with respect to generalized variables that are not merely indeterminates, such as $\cos(x+f(2)) \to \sin(\pi/2-x-f(2))$, $\cos(x+f(2))^2 \to 1-\sin(x+f(2))^2$, or $\pi \to 3.14159\,$.

As a prerequisite to discussing the wizard, it is helpful to organize the most important optional transformations offered by most computer algebra systems into categories of related transformations. For simplicity, the discussion addresses only constant ground domains that are common scalar numeric domains of characteristic 0. However, much of the discussion is relevant to other ground domains such as $\mathbb{Z}_m$ or {true, false}.

4.1 Different transformations for different generalized variables

"*To each his own.*"
– Cicero

"*I got different strokes for different folks.*"
– Muhammad Ali

Often users want certain transformations such as expansion or factoring only with respect to certain variables. For example:

- To compute the integral of $\left(x^5+(c+1)^{999}\,x+1\right)^2$ with respect to x, it is helpful to expand with respect to x, but foolish to expand with respect to c.
- To solve
$$\left(c^{999}-1\right)\left(z^2-1\right)=0 \quad | \quad c^{999}\neq 1$$
it is helpful to factor with respect to z, but foolish to factor with respect to c.

In these cases we would prefer either concise or mere default simplification with respect to c.

When the user requests successive transformation for successive variables, we do not want to destroy transformations done for prior variables. Consequently, requested transformations are automatically mapped into the largest subexpressions that do not contain variables that have already been treated. For example:

1. If the alternative for *expanding* a framed expression with respect to the chosen main variable x is
$$\left(y^2+2y+1\right)x^2-y^2+2y-1,$$
then *factoring* this alternative with respect to y gives $(y+1)^2x^2-(y-1)^2$ rather than
$$((y+1)x+y-1)\,((y+1)x-y+1)\,.$$

2. If the alternative for *factoring* a framed expression with respect to the chosen main variable x is
$$(y-1)(y+1)\left((y-2)(y+2)x+(z+1)^2\right)\left(x+(2y+1)(2y-1)\right),$$
then *expanding* this alternative with respect to y gives
$$(y^2-1)\left((y^2-4)x+(z+1)^2\right)\left(x+4y^2-1\right).$$

3. If the alternative for *factoring* a framed expression with respect to the chosen main variable x is
$$\left(y^2-1\right)\left((y^2-4)x+(z+1)^2\right)\left(x+4y^2-1\right),$$
then *factoring* this alternative with respect to y gives
$$(y-1)(y+1)\left((y-2)(y+2)x+(z+1)^2\right)\left(x+(2y-1)(2y+1)\right).$$
(Notice that the wizard factored not only the *coefficients* of powers of x with respect to y, including the coefficient of the zeroth power of x, but also the top-level *content* y^2-1, because none of these contain x.)

4. If the alternative for *expanding* a framed expression with respect to the chosen main variable x is
$$\left((z+1)^9y^2+y+3\right)x^2+(z+1)^9x+(y+1)(y-1),$$
then *expanding* this alternative with respect to y gives two distinct alternatives: *distributed* form
$$(z+1)^9x^2y^2+yx^2+3x^2+(z+1)^9x+y^2-1,$$
and the often more concise *recursive* form
$$\left((z+1)^9y^2+y+3\right)x^2+(z+1)^9x+y^2-1,$$
both of which are offered to the user. Expansion of $(z+1)^9$ will be offered if the user proceeds to that last remaining variable rather than balking or accepting an alternative already displayed.

Now consider the input $\sin(x)\left(\cos(2y)+1\right)\cos(x)$. If the user is allowed to choose trigonometric expansion of multiple angles for the generalized variable $\cos(2y)$ but choose the opposite transformation for $\sin(x)$ and $\cos(x)$, then this product can transform to the particularly concise equivalent $\sin(2x)\cos(y)^2$ because $\cos(2y)\equiv 2(\cos(y)^2-1)$ and $\sin(x)\cos(x)\rightarrow\sin(2x)/2$.

Thus for maximum flexibility:

1. The user should be able to choose the order of generalized variables.
2. The user should be able to choose separate transformations for each generalized variable.
3. The choices for each variable should include conciseForm(...) and mere reordering with any associated default simplification when such results differ from the framed subexpression.
4. Where there is expansion with respect to two or more successive variables, both distributed and recursive forms should be offered if they are not identical.

4.2 Control over the order of generalized variables

"*Order is the shape upon which beauty depends.*"
– Pearl S. Buck

Subsection 4.1 discussed how collection of similar powers of a generalized variable can be recursively applied to the resulting collected coefficients to perform transformations for successive generalized variables in any order. However, current computer algebra systems would nonetheless impose their built-in ordering rules to the resulting factors and terms. Therefore the displayed ordering of factors in terms and of terms in multinomials might not correspond to the recursive most main to least main order in which the user has treated successive variables. For example after requesting expansion with respect to y with coefficients that are factored with respect to all other variables a user might obtain a result such as

$$x^3 + y^3(z+1)^2 + y^2 z$$

rather than the more appropriate result

$$(z+1)^2 y^3 + z y^2 + x^3.$$

Also, regardless of the requested order, Newton's definition of force might be displayed as

$$f = am \tag{5}$$

which is visually quite disturbing despite its compliance with the usual alphabetical ordering convention for variables in a monomial. Consequently:

Optional transformations should include control over the displayed ordering of factors and terms.

The few systems that give such control tend to do so indirectly and incompletely via control over the ordering of generalized variables. For example:

- In the Maxima computer algebra system the desired traditional order displayed in result equation (5) can be accomplished by a declaration such as ordergreat(a, m) or orderless(m, a). If the user has issued an ordergreat(...) and a non-conflicting orderless(...) declaration, then all other variables order between the least of the great and the greatest of the least, alphabetically. The effect is global from the time of a declaration until all declared orders are deleted with an unorder() declaration, which *must* be used between any two ordergreat(...) declarations or between any two orderless(...) declarations. Maxima also provides another mechanism for overriding default alphabetical ordering:

  declare (*variable*$_1$, *property*$_1$, *variable*$_2$, *property*$_2$, ...)

  gives each variable the corresponding property. Possible properties include *constant*, *scalar*, and *mainvar*. The command remprop(*variable*, *property*) can be used to remove such a declaration. Using $\prec$ to represent "less main",

  $$constants \prec scalars \prec undeclared \prec mainvars.$$

  By default, alphabetical order is used within each of these categories.

- The Reduce computer algebra system has an order declaration that is similar to ordergreat(...), except that more than one cumulative order declaration is allowed before a declaration

  order nil;

  which clears all such ordering declarations. The Reduce order declaration also accepts functional forms and built-in literal constants such as π, which is important.

- These and some other systems provide some control over the ordering of special distributed polynomials for Gröbner bases, but that is not very helpful for controlling the ordering of factors and terms in *general* expressions.

4.3 Common alternate forms for the *rational* aspect of expressions

Many computer algebra systems have separate functions for common denominators, various factorization levels, and various levels of polynomial or partial fractions expansions. This subsection describes how, for any particular ordering of generalized variables, these traditionally disparate concepts can be organized into a single topologically sorted list of partially-ordered alternatives varying from the most complete commonly-named factorization through the most complete commonly-named expansion offered by many computer algebra systems. This *organizing principle* greatly simplifies the transformation wizard by preventing selection of a set of contradictory transformations and by making the trade-off consequences in this list more obvious.

This subsection concerns only addition, subtraction, multiplication, division and integer powers, but most of the ideas also apply recursively to rational compositions of generalized variables and to fractional powers. Moreover, this subsection discusses only factoring, common denominators and expansion because they are most relevant to estimating quickly how many transformations are applicable for each variable and for quickly determining exactly which common transformations are applicable for a particular variable. The Appendix discusses additional rational transformations.

4.3.1 Reasons for common denominators, factoring and expansion

Definition. A **candid expression** is one that is not equivalent to an expression that visibly manifests a simpler expression class [24] .

As counterexamples:

- The expression $x(y+1) - xy$ is *not* candid because it contains the superfluous variable y.
- The expression $(x+1)^2 - x^2$ is *not* candid because it appears to be quadratic but is actually linear.
- The expression $(x+1)/(x^2+2x+1)$ is *not* candid because it is equivalent to $1/(x+1)$, which has lower numerator and denominator degrees.

For most computer algebra systems, any amount of factoring *includes* reduction over a common denominator. Reduction over a common denominator yields a candid form for rational expressions, because the resulting form has no superfluous variables and has maximum possible cancellation of poles with coincident zeros.

4.3.2 A *univariate* partially-ordered set of factorization and expansion levels

1. For univariate factoring there are names for certain amounts of exact or approximate factoring based on multiplicities and the desired numeric coefficient domain of the factors:
   (a) *term primitive*,[6]
   (b) *square free*,
   (c) *over the integers* $\mathbb{Z}$,
   (d) *over the Gaussian integers* $\mathbb{Z}[i]$,
   (e) *over particular algebraic extensions*[7]
   (f) *exact reasonably absolute*,[8]
   (g) *exact absolute*,[9]
   (h) *approximate absolute over the floating-point real numbers* $\tilde{\mathbb{R}}$,
   (i) *approximate absolute over the floating-point complex numbers* $\tilde{\mathbb{C}}$.[10]

2. For systems that support variable precision Floats, users can choose the precision level for alternatives (h) and (i). For systems that offer significance and/or interval arithmetic, those alternatives immediately before (h) for real numbers and before (i) for non-real numbers.

---

[6]The *term content* of a univariate polynomial is the gcd of the numeric coefficients times the smallest occurring power of the variable. Factorization into the term content times the *term primitive part* forces a common denominator if any coefficient has a denominator, because with polynomials A, B, C and D,

$$\frac{A}{B}+\frac{C}{D} \equiv AB^{-1}+CD^{-1} \quad \rightarrow \quad (AD+BC)B^{-1}D^{-1} \equiv \frac{AD+BC}{BD} \quad \rightarrow \quad \frac{(AD+BC)/G}{(BD)/G}$$

where $G \leftarrow \gcd(AD+BC,\, BD)$.

[7]As a convenience in *Mathematica,* Factor[*expression*, Extension → Automatic] automatically uses extensions implied by the complex unit i and/or any radicals *present* in *expression*. For example,

$$\text{Factor}[x^2+x-2+\sqrt{2},\ \text{Extension} \rightarrow \text{Automatic}] \quad \rightarrow \quad -\left(-x-1+\sqrt{2}\right)\cdot\left(x+\sqrt{2}\right)$$

However, Factor$[x^2+2\cdot\sqrt{2}\cdot x-1,$ Extension → Automatic$] \not\rightarrow \left(x+\sqrt{2}-\sqrt{3}\right)\cdot\left(x+\sqrt{2}+\sqrt{3}\right)$ because $\sqrt{3}$ is not in the given polynomial. We must instead use Factor$[x^2+2\cdot\sqrt{2}\cdot x-1,$ Extension → $\{\sqrt{2},\ \sqrt{3}\}]$ to obtain this factorization, but how many users would know to include $\sqrt{3}$?

[8]This is what is usually expected of algebra through calculus students for purposes such as solving equations or integrating rational functions: Algebraic extensions implied by radicals in the input together with use of the quadratic formula and n^{th} roots to factor binomials. *Derive* offers this option but also includes cubic and quartic formulas, which tends to generate unreasonably messy factorizations.

[9]This means whatever algebraic extension is necessary to factor the polynomial as much as possible, without the extension being provided by the user. Reference [9] discusses some algorithms for this. Some systems appear to use absolute factorization in their functions that solve systems of polynomial equations and integrate, but unfortunately appear not to offer it as a built-in factorization option. Therefore many computer algebra systems cannot directly factor x^2+2x-1 into $(x+1+\sqrt{2})(x+1-\sqrt{2})$ without assistance, which any beginning algebra student can do!.

Attempted exact absolute factorization might consume an intolerable amount of computing time, or resulting factors might entail intolerably messy nested radicals or intolerably messy subexpressions containing functional forms named something such as Root. This is why I list the *exact reasonably absolute* level of factorization.

[10]Alternatives (h) and (i) are *approximations* rather than equivalence transformations. Some methods for exact absolute factorization begin from an approximate absolute factorization that is often preferable to the resulting messy exact factorization!

3. With $F_1 \succeq F_2$ denoting the fact that for a given example, a factorization at level F_1 is either identical to a factorization at level F_2 or is a further splitting of the factorization at level F_2, we have

$$\mathbb{Z}[i] \succeq \mathbb{Z} \succeq \text{square free} \succeq \text{term primitive},$$
$$\text{exact absolute} \succeq \text{specific algebraic extensions} \succeq \mathbb{Z},$$
$$\text{exact absolute} \succeq \text{reasonably exact absolute} \succeq \mathbb{Z},$$
$$\tilde{\mathbb{C}} \succeq \tilde{\mathbb{R}} \succeq \mathbb{Z},$$
$$\tilde{\mathbb{C}} \succeq \mathbb{Z}[i].$$

   Thus these factorization levels form a *directed acyclic graph*. that we can topologically sort into one of several alternative lists, such as order 1(a) through 1(i) above.[11]

4. If we fully expand the product of the numerator factors and the product of the denominator factors of a reduced ratio, then we have the reduced ratio of two fully expanded polynomials. Despite the common denominator, the result is an expanded polynomial when this reduced denominator is 1 or when both the numerator and denominator are numeric. Therefore this form is on the borderline between factored and expanded.

5. The computer algebra built-into Texas Instruments hand held, Windows and Macintosh products has a function propFrac(*expression*, *variable*) that expands *expresssion* into a expanded polynomial with respect to *variable* plus a reduced ratio of two polynomials that is *proper* with respect to *variable*. The propFrac(...) function can easily be implemented using a polynomial quotient and remainder function, and the resulting form is an appropriate next node in our partial ordering from most factored to most expanded. This form is often more concise than either a common denominator or a partial fraction expansion. For example, this form was a key intermediate step in the example of subsection 3.1. For canonicality:

   (a) The coefficients of the resulting expanded univariate polynomial part that are not complex Floats can be normalized to Gaussian rationals $\mathbb{Q}[i]$ or rationalized algebraic numbers.

   (b) The denominator of the proper ratio can be made unit normal as described in [25].

   (c) The numeric coefficients in the resulting proper ratio that are not complex Floats can be normalized to Gaussian integers or algebraic integers.

   (d) If all of the denominator numeric coefficients are complex Floats, then we can normalize their magnitudes – such as making the largest of the real and imaginary magnitudes in the denominator coefficients be 1.0.

6. Polynomial expansion can be regarded as a special case of propFrac(...) for when the denominator of the given reduced ratio is numeric – perhaps 1.

7. If the reduced ratio of two polynomials has a non-numeric denominator, then the relevant adjective phrases 1(a) through 1(i) above can be used to label successive nodes corresponding to the amount of denominator factorization for corresponding partial fraction expansions.

[11] Actually, different specific algebraic extensions generally form a directed acyclic subgraph because, for example, we could have any one, two or all three of the extensions $\sqrt{2}$, $\sqrt{3}$ and $\sqrt{5}$, giving more than one path to $\{\sqrt{2}, \sqrt{3}, \sqrt{5}\}$. These directed acyclic subgraphs are the field extension lattices of Galois theory.

8. However, the square-free aspect of partial fraction expansion has two variants in the partial ordering. In non-decreasing order of the amount of expansion, adjective phrases applicable to the square free aspect are:
   (a) *incomplete*, meaning multiples of all powers of the same square-free denominator factor are combined over a common denominator, and
   (b) *complete*, meaning instead that for each resulting *very proper* ratio $N(x)/D(x)^m$ with expansion variable x, $\deg_x(N(x)) < \deg_x(D(x))$.[12]

Whenever a resulting numerator has more than one term, we can distribute an associated denominator over the numerator terms. It is generally unwise to distribute a multinomial denominator over the numerator terms for purposes such as integration, and it almost always increases bulk. However, it is helpful to do such a distribution for purposes such as fragmenting a ratio into the greatest number of simplest possible pieces for angle sum expansion.[13] The wizard can display both alternatives when they are not identical.

Table 1 shows the named alternative forms for a univariate example of the reduced ratio of two expanded polynomials.

1. The first two rows and last two rows are approximations to all of the other rows, which are equivalent to each other.
2. The double line separating "ratio of expanded polynomials" and "polynomial + proper ratio" separates factored from expanded forms.
3. Factors that differ from those of the preceding row are boldface.
4. Wherever there is a sum in a numerator, the corresponding denominator can optionally be distributed over the terms of the numerator.
5. For each of these named levels an example can be constructed where it is more concise than all of the other named levels. Therefore all of the levels are important.[14]
6. If a system doesn't offer built-in support for all of these named levels and a wizard implementer is not inclined to add such support, then:
   (a) Missing intermediate factorization levels can be provided by over-factoring then expanding appropriate subsets of factors.
   (b) Missing expansion levels can be provided by over expanding then combining appropriate subsets of summands.
7. The input could be any of these expressions or any rational expression that is equivalent to one of these expressions. If an input contains Floats, then float-free alternatives can be obtained by using a function such as the Maple identify(...) function to determine close rational or irrational constants [5].

[12] In contrast, for the *incomplete* square-free partial fraction expansion we can guarantee only that $\deg_x(N(x)) < \deg_x(D(x)^m)$. Many systems offer only complete expansions, but incomplete expansions are adequate for most purposes and are often more concise!

[13] If you must distribute multinomial denominators over numerator terms, it is most efficient to wait until after the expansion is complete in other respects.

[14] There are also unnamed intermediate levels such as splitting some but not all of the square-free factors over $\mathbb{Z}$.

Table 1: A univariate expression partially ordered from most factored to expanded

Amount of factor or expand	Boldface parts are different from the alternative above them
$\tilde{\mathbb{C}}$	$\frac{1.5(z-1.37)(z+5.7)(z-1.07+0.76i)(z-1.07+0.76i)\cdots(z+1.09+0.18i)(z+1.09-0.18i)}{z(z-1)^2(z+i)(z-i)(z+1.414)(z-1.414)(z+1.13)(z-1.04+0.82i)(z-1.04-0.82i)\cdots}$
$\tilde{\mathbb{R}}$	$\frac{1.5\cdots\mathbf{(z^2-2.13z+1.71)(z^2-1.13z+0.9)(z^2-0.05z+0.24)\cdots(z^2+2.19z+1.23)}}{z(z-1)^2\mathbf{(z^2+1)}(z+1.414)(z-1.414)(z+1.13)\mathbf{(z^2-2.08z+1.76)(z^2+0.95z+1.5)}}$
reasonably absolute ($\mathbb{Z}\left[i,\sqrt{2}\right]$)	$\frac{\mathbf{3(z^{12}+4z^{11}-9z^{10}+4z^9+z^8+13z^6-29z^5+7z^4+13z^3-25z^2+6z-6)}}{\mathbf{2}z(z-1)^2\mathbf{(z+i)(z-i)(z+\sqrt{2})(z-\sqrt{2})(z^5+z+3)}}$
$\mathbb{Z}[i]$	$\frac{3(z^{12}+4z^{11}-9z^{10}+4z^9+z^8+13z^6-29z^5+7z^4+13z^3-25z^2+6z-6)}{2z(z-1)^2(z+i)(z-i)\mathbf{(z^2-2)}(z^5+z+3)}$
$\mathbb{Z}$	$\frac{3(z^{12}+4z^{11}-9z^{10}+4z^9+z^8+13z^6-29z^5+7z^4+13z^3-25z^2+6z-6)}{2z(z-1)^2\mathbf{(z^2+1)}(z^2-2)(z^5+z+3)}$
square free	$\frac{3(z^{12}+4z^{11}-9z^{10}+4z^9+z^8+13z^6-29z^5+7z^4+13z^3-25z^2+6z-6)}{2z(z-1)^2\mathbf{(z^9-z^7-z^5+3z^4-z^3-3z^2-2z-6)}}$
term primitive	$\frac{3(z^{12}+4z^{11}-9z^{10}+4z^9+z^8+13z^6-29z^5+7z^4+13z^3-25z^2+6z-6)}{2z\mathbf{(z^{11}-2z^{10}+2z^8-2z^7+5z^6-8z^5+2z^4+3z^3-5z^2+10z-6)}}$
ratio of expanded polynomials	$\frac{\mathbf{3z^{12}+12z^{11}-27z^{10}+12z^9+3z^8+39z^6-87z^5+21z^4+39z^3-75z^2+18z-18}}{\mathbf{2z^{12}-4z^{11}+4z^9-4z^8+10z^7-16z^6+4z^5+6z^4-10z^3+20z^2-12z}}$
polynomial + proper ratio	$\frac{3}{2}+\frac{\mathbf{18z^{11}-27z^{10}+6z^9+9z^8-15z^7+63z^6-93z^5+12z^4+54z^3-105z^2+36z-18}}{2z^{12}-4z^{11}+4z^9-4z^8+10z^7-16z^6+4z^5+6z^4-10z^3+20z^2-12z}$
term primitive partial fraction	$\frac{3}{2}+\mathbf{\frac{3}{2z}}+\mathbf{\frac{15z^{10}-21z^9+6z^8+3z^7-9z^6+48z^5-69z^4+6z^3+45z^2-90z+6}{2z^{11}-4z^{10}+4z^8-4z^7+10z^6-16z^5+44^5+6z^3-10z^2+20z-12}}$
incomplete square free part frac	$\frac{3}{2}+\frac{3}{2z}+\mathbf{\frac{3z+3}{(z-1)^2}}+\mathbf{\frac{12z^8-3z^6-3z^4+36z^3-18z+24}{2z^9-2z^7-2z^5+6z^4-2z^3-6z^2-4z-12}}$
complete square free part frac	$\frac{3}{2}+\frac{3}{2z}+\mathbf{\frac{3}{(z-1)^2}}+\mathbf{\frac{3}{2z-2}}+\frac{12z^8-3z^6-3z^4+36z^3-18z+24}{2z^9-2z^7-2z^5+6z^4-2z^3-6z^2-4z-12}$
partial fractions over $\mathbb{Z}$	$\frac{3}{2}+\frac{3}{2z}+\frac{3}{(z-1)^2}+\frac{3}{2z-2}+\mathbf{\frac{3z}{z^2-2}}+\mathbf{\frac{3z}{z^2+1}}+\mathbf{\frac{3z^2-12}{2z^5+2z+6}}$
partial fractions over $\mathbb{Z}[i]$	$\frac{3}{2}+\frac{3}{2z}+\frac{3}{(z-1)^2}+\frac{3}{2z-2}+\frac{3z}{z^2-2}+\mathbf{\frac{3z}{2z+2i}}+\mathbf{\frac{3}{2z-2i}}+\frac{3z^2-12}{2z^5+2z+6}$
absolute partial fractions	$\frac{3}{2}+\frac{3}{2z}+\frac{3}{(z-1)^2}+\frac{3}{2z-2}+\mathbf{\frac{3}{2z+2\sqrt{2}}}+\mathbf{\frac{3}{2z-2\sqrt{2}}}+\frac{3z}{2z+2i}+\frac{3}{2z-2i}+\frac{3z^2-12}{2z^5+2z+6}$
partial fraction over $\tilde{\mathbb{R}}$	$\cdots+\mathbf{\frac{1.5}{z+1.41}}+\mathbf{\frac{1.5}{z-1.41}}+\mathbf{\frac{3.0z}{z^2+1}}+\mathbf{\frac{0.162}{z+1.1}}-\mathbf{\frac{0.18z-0.27}{z^2-2.1z+1.8}}+\mathbf{\frac{0.018z+0.31}{z^2+0.95z+1.5}}$
partial fraction over $\tilde{\mathbb{C}}$	$1.5+\cdots-\mathbf{\frac{0.089+0.049i}{z-1.04+0.82i}}-\mathbf{\frac{0.089-0.049i}{z-1.04-0.82i}}+\mathbf{\frac{0.0088+0.13i}{z+0.48+1.13i}}+\mathbf{\frac{0.0088-0.13i}{z+0.48-1.13i}}$

4.3.3 *Multivariate* partially-ordered sets of factorization and expansion levels

Reference [26] describes how to do multivariate partial fraction expansion with respect to two or more successive generalized variables. As a degenerate case, the expansion is *polynomial* expansion with respect to variables that don't occur in the reduced common denominator of the given expression.

As shown that article, the number of terms in a multivariate partial fraction expansion can depend on the ordering of the expansion variables. Table 2 shows eight alternatives for factoring and or expanding a bivariate example over $\mathbb{Z}$. Notice how the partial fraction expansion with respect

to y in the last two rows introduces poles at $x = \pm 1$ into some individual ratios, forcing the use of a piecewise result to avoid contracting the domain of definition. Making a ratio proper can also cause this. (Unfortunately, most current computer algebra systems quietly do such domain reductions.) Common denominators remove these additively removable singularities.

Table 2: Some alternative recursive form factorizations and expansions over $\mathbb{Z}$.
f = **f**actored. e = **e**xpanded to incomplete partial fractions.

1st	2nd	Result
f_x	f_y	$\frac{(y-1)(y+1)(y^2+3)x^3-y(y+1)(y^4-y^3+y^2-3y-4)x-2y^2(y-1)(y^2+2)}{(x-y)(x+y)(y-1)(y+1)(y^2+2)}$
f_x	e_y	$\frac{(y^4+2y^2-3)x^3-(y^6-2y^3-7y^2-4y)x-(2y^5-2y^4+4y^3-4y^2)}{(x-y)(x+y)(y^4+y^2-2)}$
f_y	f_x	$\frac{xy^6+2y^5-(x^3+2)y^4-2(x-2)y^3-(2x^3+7x+4)y^2-4xy+3x^2}{(y-x)(y+x)(y-1)(y+1)(y^2+2)}$
f_y	e_x	$\frac{xy^6+2y^5-(x^3+2)y^4-(2x-4)y^3-(2x^3+7x+4)y^2-4xy+3x^2}{(y-x)(y+x)(y-1)(y+1)(y^2+2)}$
e_x	f_y	$\frac{y^3+3}{y^2+2}x + \frac{2y^2}{(x+y)(y-1)(y+1)} + \frac{2y}{(x-y)(y-1)(y+1)}$
e_x	e_y	$x + \frac{x}{y^2+2} + \frac{2}{x+y} + \frac{1}{xy-x+y^2-y} + \frac{1}{xy+x+y^2+y} + \frac{1}{xy-x-y^2+y} + \frac{1}{xy+x-y^2-y}$
e_y	f_x	$\begin{cases} 1 - \frac{y+8}{6(y-1)^2} - \frac{11y+8}{6(y+1)^2} + \frac{1}{3(y^2+2)}, & \text{if } x = -1, \\ -1 + \frac{1}{(y-1)^2} - \frac{2y+1}{(y+1)^2} - \frac{1}{y^2+2}, & \text{if } x = 1, \\ x + \frac{x}{y^2+2} + \frac{2x}{(x-1)(x+1)(x-y)} - \frac{2x^2}{(x-1)(x+1)(y+x)} + \frac{2x}{(x-1)(x+1)(y-1)} - \frac{2}{(x-1)(x+1)(y+1)}, & \text{otherwise} \end{cases}$
e_y	e_x	$\begin{cases} 1 - \frac{y+8}{6(y-1)^2} - \frac{11y+8}{6(y+1)^2} + \frac{1}{3(y^2+2)}, & \text{if } x = -1, \\ -1 + \frac{1}{(y-1)^2} - \frac{2y+1}{(y+1)^2} - \frac{1}{y^2+2}, & \text{if } x = 1, \\ \cdots + \frac{2}{y+x} - \frac{1}{xy+y+x^2+x} + \frac{1}{xy-y+x^2-x} - \frac{1}{xy+y-x^2-x} - \frac{1}{xy-y-x^2+x} + \cdots - \frac{1}{xy-y+x-1} + \frac{1}{xy-y-x+1}, & \text{otherwise} \end{cases}$

4.4 Series and other approximations

The discussion so far has been about transformations to alternatives that are *equivalent* to the input wherever the input is defined – except perhaps approximating exact numbers in the input with approximate Floats or approximating Floats in the input with nearby rational numbers. This subsection instead addresses the equally important alternatives of transformations that *approximate* the input with *simpler expressions*.

Closed-form exact results aren't always obtainable. Even when they are, the results might be too bulky to convey needed insight or to permit fast enough well-conditioned evaluation for numerous floating-point values of the variables therein. Therefore, various kinds of approximation are useful transformations. Also it is important to realize that the ultimate destiny of many exact expressions is to substitute Floats into them, in which case the resulting rounding errors might exceed those caused by an approximate expression. Here are some examples of appropriate optional approximate transformations for a wizard to offer:

- Quadrature can often be used to determine a single approximate number for a definite integral.

- Approximate equation solving is often preferable even when compact explicit exact solutions are obtainable.
- Generalized infinite or truncated Laurent-Puiseux series (allowing, for example, logarithmic factors) can concisely approximate lengthy expressions. The wizard can initialize expansion points to ones that are most likely of interest, such as 0, infinities, and poles. If selected, the user can adjust the expansion points and the requested order.
- Padé approximations often have a larger region of convergence and greater computational efficiency than power series.
- Truncated Fourier or wavelet series are often more appropriate than expansions about a point.

A well-chosen approximation can be simpler than any obtainable exact result and yet retain all of the important characteristics of an exact result.

4.5 Generating the list of generalized variables

If a variable v in the framed subexpression has an assigned value, then it would be misleading to list that irrelevant v. However, we do want to consider listing some or all of the generalized variables in the *assigned value*, if any. We can use default simplification for this purpose, because it replaces all assigned variables with their values.

On many systems *default* simplification can easily produce results containing generalized variables that candid simplification would eliminate. For example, the default simplification of most systems merely reorders the factors and/or terms in the input $((x+1)x - x^2 - x + 2)\,/\,(y^2-1)$, but all of the factoring and expansion transformations described in sub-subsections 4.3.2 and 4.3.3 transform the expression to $2/(y^2-1)^2$ or $2/\left((y-1)(y+1)\right.$ or $1/(y-1) + 1/(y+1)$, which all depend on y alone.

It is helpful for the wizard to recognize such *superfluous* generalized variables. For polynomial expressions, expansion to recursive form always eliminates superfluous variables. For other rational expressions, reduction over a common denominator always does so.[15] Thus the first thing that the wizard can do is compute such a form to identify generalized variables that thereby disappear. One of these transformations can be initially checked to help encourage the user to eliminate superfluous generalized variables. However, the user might prefer to eliminate them in a way that alters cherished structure less drastically. To do this, the wizard could also offer the alternative, for example, "merely eliminate superfluous x and $\ln(y)$". This can be accomplished by substituting simple exact constants such as 0, 1 or -1 for those generalized variables, then applying default simplification. If the substitution value is at a removable singularity and thereby causes division by 0, then the wizard can backtrack and try another simple constant.

A generalized variable wouldn't be offered if doesn't affect ordering and no optional transformation is applicable to it. For example, it is pointless to list x if it only occurs as the argument in $\sin(x)$. However, it might be worthwhile to list the generalized variable $\sin(x)$, depending on how it occurs in the default-simplified result. As another example, *none* of the transformations or ordering choices being discussed here are applicable to the subexpression $3x^2$.

There will usually be an initial "Choose main variable" dialog if there is more than one applicable generalized variable. If so, and after choosing the transformation with respect to the selected main

[15] If this reduction produces $0/0$, then the expression is undefined for all values of its variables, so the one and only offered alternative can be the result of "0/0".

variable there is still more than one applicable generalized variable for some maximal subexpression that doesn't contain the main variable, then there will be another dialog labeled instead "Choose next most main variable", and so on until the user aborts the investigation or accepts replacements.

4.6 Estimating the number of common transformations for each generalized variable choice

Quick syntactic checks can identify some opportunities for transforming a framed subexpression with respect to a generalized variable. For example,

- Expansion is applicable to any variable that occurs in a multinomial raised to an integer power or a multinomial multiplied by any subexpression.
- A common denominator is applicable to any variable that occurs to a negative power in a sum.
- Angle sum expansion is applicable if the argument of a sinusoid is a sum.
- The transformations $\sin(u)^2 \to 1 - \cos(u)^2$ or $\sin(u)^n \to (\sin(nu) + \cdots)/2^n$ are applicable if $\sin(u)$ is raised to any integer power exceeding 1.

Quick syntactic checks can also *preclude* some opportunities for transforming a framed subexpression with respect to a generalized variable. For example, expansion is precluded if the framed subexpression is monomial or linear in the variable.

Another ordering heuristic is that the number of applicable transformations is likely to increase with the degree of a variable in a numerator or denominator, and more transformations are associated with high degree denominators than high degree numerators, because denominator factorizations can also enable partial fraction expansions.

If time remains in the 0.1 second acceptable delay for generating and displaying the first dialog box, then we can compute more accurate estimates for the number of common applicable transformations for each variable: For the rational aspects of a framed subexpression, minimal-effort reduction over a common denominator reveals the numerator and denominator degrees of every top-level generalized variable, which are all of those that can be affected by top-level factoring or expansion. Minimal effort here means using partially factored form – preferably recursive – as described in [24]. If this reduced partially factored form differs from the framed subexpression, then it is already one applicable transformation. Moreover, expansion to a polynomial plus a proper fraction is applicable to every such generalized variable whose degree in the resulting numerator is at least as large as the degree in the denominator; and it is easy to compute these degrees for partially factored forms.

If time still remains in the 0.1 second acceptable delay, then we can compute more accurate estimates by factoring the common denominator with respect to all of its generalized variables. From this it is easy to determine which successively lesser factorizations would combine two or more factors containing that main variable and thus determine the number of distinct named partial fraction expansions with respect to that variable and a lower bound on the number of distinct named factorizations over a common denominator.

If time still remains, then the wizard can factor the numerator to better estimate the number of named factorizations over a common denominator.

To reduce the chance of exceeding 0.1 second before *any* factorization is achieved, it could be done in levels, such as term content with respect to every variable, then square free, etc. through, say, exact reasonably absolute factorization followed by absolute factoring over the complex Floats. If the allotted 0.1 seconds runs out before completing this agenda, then we simply use the best estimates that we have at that time. Moreover, we can continue to refine and update the estimates after the initial display, without changing the initial order of the generalized variable buttons.

4.7 Recognizing applicable transformations

The exact and approximate factorizations with respect to all variables is a useful point of departure for computing alternative results regardless of what variable the user chooses. Therefore it is helpful to proceed computing those factorizations in the background while users ponder their choice of variable.

After a user chooses a variable, the wizard can complete the distinct factorization levels by appropriately expanding pairs of factors, which is fast. To convey the strongest possible information, when labeling the displayed alternatives or elided versions thereof, they are labeled with the most thorough applicable factoring level. For example, if the only distinct factorizations are factorizations over the Gaussian integers and over the integers, then the later would not be labeled "square-free" even though it is that as well. If there is only one factored alternative, then it could be labeled merely "factored" for brevity, but with a more detailed phrase displayed upon mouse-over. This provides a learning opportunity for mathematically unsophisticated users.

If there is a denominator and it contains the chosen variable, the wizard can then proceed to compute the polynomial part and proper fraction, followed by any distinct partial fraction levels with respect to that variable.

5 If I want this interface, why haven't I implemented it?

Good question. The reasons are:

- I am not an interface programmer.
- It is best done by a team.
- Some aspects will require access to proprietary internals that are inaccessible to outsiders for systems that aren't open source.
- It will require testing feedback from numerous users. Surely some of the ideas presented here won't work out well in practice, and better ideas will occur.
- If it isn't the default interface distributed with a computer algebra system, then it is unlikely to be known to most users, and the system is likely to evolve in a way so that the alternative interface no longer works.
- For various reasons, corporations are often unwilling to adopt and maintain packages written by outsiders as first class parts of their system – thoroughly and seamlessly integrated into the system and the documentation with no need for building or loading.

Therefore, although I would be delighted to help, the purpose of this manifesto is to stimulate computer algebra users to request better interfaces and stimulate decision makers to build them.

"*If you build it, they will come.*"
– an apt misquote from *Field of Dreams*

Computer algebra users of the world: the squeaky wheels get the grease!

Acknowledgments

I thank Bill Gosper for information about Lisp Machine Macsyma and Norbert Kajler for helpful suggestions. Jacques Carette provided so many extraordinarily good suggestions that he should be a co-author – except for conflict of interest that he is an unmasked referee!

Appendix: More transformations for rational expressions

It is worthwhile to list in one place most of the many known transformations that might be of general interest. Subsection 4.3 discussed factoring, common denominators and expansion. Here are some additional transformations for the rational aspect of expressions:

A.1: Basic – of interest to most users

The wizard should automatically try the following transformations in the background because they can dramatically decrease the bulk of an expression and reveal important structure:

Polynomial shifts
Most of the factorization and expansion levels leave polynomial sub-expressions that often have more than two terms. Sometimes merely re-expressing such a multinomial in terms of optimally shifted variables can greatly reduce the number of terms and/or the size of coefficients. For example,

$$\left(y^2-4y+4\right)x^3+\left(3y^2-12y+12\right)x^2+\left(3y^2-12y+12\right)x+y^2-4y+1 \;\;\rightarrow\;\; (y-2)^2(x+1)^3+8,$$

$$531441\,x^6+2834352\,x^5+6298560\,x^4+7464960\,x^3+4976640\,x^2+1769472\,x+262151 \rightarrow (9x+8)^6+7\,.$$

Articles [13, 14], give algorithms for computing optimal shifts. As a quick preclusion test, it is not worth shifting a polynomial with respect to a variable if the polynomial is monomial or binomial with respect to that variable.

Polynomial decomposition
Complementary to such shift decompositions, Kozen and Landau [17] describe an efficient algorithm for completely decomposing a univariate polynomial $p(x)$ into nested polynomials

$$p_1\left(p_2\left(\ldots\left(p_m\left(x\right)\right)\ldots\right)\right)$$

with each $p_k\left(t\right)$ of degree at least 2 in t. For example, the irreducible polynomial

$$\begin{aligned} P(x) &:= x^{12}+4x^{10}+x^9+6x^8+3x^7+4x^6+3x^5+x^4+x^2+7 \\ &\rightarrow \left(x^3+x\right)^4+\left(x^3+x\right)^3+7. \end{aligned}$$

Their algorithm also applies recursively to multivariate polynomials represented recursively. As a quick preclusion test, such decompositions are inapplicable with respect to a variable of degree less

than 4 or having few terms. There are also algorithms for other kinds of univariate and multivariate polynomial decompositions, as described, for example, in [12, 34, 33, 32, 35].

Polynomials rewritten in these ways can reveal significant structure, help precondition an expression for efficient repeated numeric evaluation or reduced rounding error, and facilitate solutions of higher-degree polynomial equations or systems of equations. For example, with the above decomposition, masochists could apply the quartic formula then the cubic formula to express the zeros of $P(x)$ in terms of radicals.

Linear combinations of powers

At least since Pythagoras, people have been interested in representing numbers and non-numeric expressions as sums, differences, or general linear combinations of powers of other expressions. For example, if an expression can be rewritten as a sum of even powers of real subexpressions, then the expression is thereby proven to be nonnegative for all real values of these subexpressions. Algorithms for such transformations can be found in, for example, [21, 22, 31].

A.2: Advanced – of interest only to experts

Here are some transformations that should be tried only in response to the Advanced button because they would probably intimidate and distract most users without any compensating benefit to them.

Expression in terms of orthogonal polynomials

Important optional transformations include a change of basis from monomials to orthogonal polynomials, which can

- yield more concise results,
- yield results less subject to magnified rounding errors,
- reveal patterns that otherwise wouldn't be apparent, or
- suggest efficient accurate min-max truncated approximations.

Many computer algebra systems provide functions that return one of various classic orthogonal polynomials of a specified degree in a specified variable using the monomial basis. However, most of the systems provide little or no automation for:

- converting ordinary polynomials to linear combinations of orthogonal polynomials specified by symbols such as the Chebyshev polynomials of the first kind $T_0(z)$, $T_1(z)$, ... ;
- propagating linear combinations of such polynomials into other linear combinations thereof *exactly* through rules such as

$$\begin{aligned} T_m(z)\,T_n(z) &\rightarrow \frac{1}{2}\,T_{|n-m|}(z) + \frac{1}{2}\,T_{m+n}(z), \\ T_m\left(T_n(z)\right) &\rightarrow T_{mn}(z), \\ \int T_n(z)\,dz &\rightarrow \begin{cases} \frac{1}{4}\left(T_0(x) + T_2(x)\right), & \text{if } n = 1, \\ \frac{1}{2-2n}\,T_{n-1}(z) + \frac{1}{2+2n}\,T_{n+1}(z), & \text{otherwise,} \end{cases} \end{aligned}$$

  and *approximately* through infinite or truncated series expansions;

- efficiently and accurately substituting Floats into expressions involving combinations of such functions: Rather than substituting a number z_0 for z in the monomial-basis representations of the symbols $T_0(z)$, $T_1(z)$, ... in a combination thereof, it is much faster and more accurate to compute the successive $T_k(z_0)$ from the recurrence $T_n(z_0) \leftarrow 2z_0\,T_{n-1}(z_0) - T_{n-2}(z_0)$.

Of the various named orthogonal polynomials, $T_0(z)$, $T_1(z)$, ... are probably most important. Therefore, Trefethen and others [30] developed a powerful MATLAB package that automates effective use of these polynomials for many applications, using IEEE double Floats to represent the coefficients. Fateman [11] implemented some of these capabilities in Maxima to take advantage of its variable precision floating point and exact rational arithmetic. Such transformations and analogous ones for other orthogonal polynomials would be a welcome addition to many computer algebra systems.

Expression in terms of symmetric polynomials

Analogous transformations for the most common kinds of *symmetric polynomials* would be another welcome addition. Such polynomials can help reduce the curse of dimensionality for problems that exhibit symmetries when pairs of variables are interchanged. As a start toward this, the *Mathematica* function SymmetricReduction [*expression*, $\{v_1, v_2, \ldots v_n\}$, $\{s_1, s_2, , \ldots, s_n\}$] returns the pair $\{p, r\}$ where *expression* depends on variables $\{v_1, v_2, \ldots v_n\}$, p is the symmetric component of *expression* in terms of symbols $\{s_1, s_2, , \ldots, s_n\}$ representing the *elementary symmetric polynomials* through degree n, and r is the residual of *expression* that can't be so represented. To convert a symmetric result back to the original variables, function SymmetricPolynomial[$m, \{v_1, v_2, \ldots v_n\}$], returns the m^{th} elementary symmetric polynomial using variables $\{v_1, v_2, \ldots v_n\}$. Sturmfels [28] contains algorithms for transforming to and from symmetric polynomials.

Rational Decomposition

We can attempt separate polynomial decompositions on every numerator and denominator polynomial in an expression. However, there are also algorithms for decomposing *ratios* of polynomials into *nested ratios*, as discussed in [4, 15, 39]. As an example from the first of these articles, but using recursive form and primitive normalization, the ratio

$$\frac{(y^2 + 2z^2y + z^4 - 81)\,x^2 - 2y\cdot(y^5 + z^2y^4 + 225\,z)\,x + y^2\,(y^8 - 625\,z^2)}{(y^2 + 2z^2y + z^4 - 162)\,x^2 - 2y\cdot(y^5 + z^2y^4 + 450\,z)\,x + y^2\,(y^8 - 1250\,z^2)}$$

$$\rightarrow \begin{cases} 1, & \text{if } 9x + 25zy = 0, \\ \dfrac{\left(\dfrac{(y+z^2)\,x - y^5}{9x + 25\,zy}\right)^2 - 1}{\left(\dfrac{(y+z^2)\,x - y^5}{9x + 25\,zy}\right)^2 - 2}, & \text{otherwise.} \end{cases}$$

This example also illustrates that rational decomposition can introduce new removable singularities in the nested form, such as on the manifold $9x + 25zy = 0$ for this example. We can avoid this by clearing the nested denominators but preserving the nested polynomial components thereof to obtain *correlated* polynomial decompositions of the numerator and denominator:

$$\frac{\left(\dfrac{(y+z^2)\,x - y^5}{9x + 25zy}\right)^2 - 1}{\left(\dfrac{(y+z^2)\,x - y^5}{9x + 25zy}\right)^2 - 2} \;\rightarrow\; \frac{((y+z^2)\,x - y^5)^2 - (9x + 25\,zy)^2}{((y+z^2)\,x - y^5)^2 - 2\,(9x + 25\,zy)^2}.$$

If desired, for this example we can then further factor the difference in two squares in the numerator and in the denominator to obtain the factorization over $Z[\sqrt{2}\,]$.

$$\frac{\left(\left(y+z^2+9\right)x-y^5+25\,zy\right)\left(\left(y+z^2-9\right)x-y^5-25\,zy\right)}{\left(\left(y+z^2+9\sqrt{2}\right)x-y^5+25\sqrt{2}\,zy\right)\left(\left(y+z^2-9\sqrt{2}\right)x-y^5-25\sqrt{2}\,zy\right)}.$$

The numerator factorization could easily have been computed from the original numerator. However, the required $\sqrt{2}$ algebraic extension necessary to factor the multivariate denominator would be more difficult to determine without the intervening rational decomposition.

Continued fractions

Continued fractions are another type of compound-fraction representation for rational expressions. Acton [1] lists three different variants of continued fractions together with algorithms for converting between them and a reduced ratio. Cuyt and Verdonk [10] review methods for multivariate continued fractions. As an example of a continued fraction expansion that reveals a simple pattern:

$$\frac{\left(z^4-105\,z^2+945\right)z}{15\left(z^4-28\,z^2+63\right)}\;|\;z^2\notin\left\{63,\;\frac{45}{2},\;\frac{3}{2}\left(35\pm\sqrt{805}\right)\right\}\rightarrow\cfrac{z}{1-\cfrac{z^2}{3-\cfrac{z^2}{5-\cfrac{z^2}{7-\cfrac{z^2}{9}}}}}.$$

Here a constraint was appended to the input to avoid the *appearance* of contracting the domain of definition because of removable singularities introduced by the continued fraction. If instead we used a piecewise result, then it would have 9 pieces, 8 of which are constants that can be determined by substituting the two square roots of each element in the constraint set into the original expression.

Actually, if the computer algebra system automatically handles unsigned zeros and infinities correctly, then with exact computation the continued fraction form evaluates to the correct finite values even at these removable singularities. However, unlike the reduced ratio, the continued fraction form *might* be less accurate due to catastrophic cancellation near those introduced singularities. Therefore it is worth alerting the user to these singularities by either appending an input constraint or producing a piecewise result.[16]

Hornerized forms

In its simplest form, Horner's rule is the factoring out of the least power of a variable from successively lower-degree terms. For example,

$$-1234321\,x^7-1234321\,x^5+2468642\,x^4+7x^3+14x-21$$
$$\rightarrow\left(\left(\left(\left(-1234321\,x^2-1234321\right)x+2458642\right)x+7\right)x^2+14\right)x-21.$$

[16]With correct handling of infinities, a continued fraction can be defined at infinity where a reduced common denominator is *not*. For example, $1/(1+1/z)\rightarrow 1$ at $z=\infty$, where the corresponding reduced ratio $z/(z+1)\rightarrow\infty/\infty$ is indeterminate. Most people would prefer having a result defined at all *finite* values of variables to being defined at *infinite* values, but a mere division can turn 0 into an infinite value.

It is also worth partially factoring out units and numeric content to the extent that it reduces bulk or the number of operations. For example,

$$-1234321\,x^7 - 1234321\,x^5 + 2468642\,x^4 + 7x^3 + 14x - 21$$
$$\rightarrow ((-1234321((x^2+1)\,x-2)\,x+7)\,x^2+14)\,x-21.$$

Horner's rule often leads to faster evaluation when substituting numbers for variables, which is particularly important in situations such as plotting, where substitution is done many times for different values. Horner's rule also often improves accuracy for approximate arithmetic, because the operands of a catastrophic cancellation are closer to the input numbers, hence less contaminated with rounding error.

Horner's rule can be viewed as factoring out term content term by term, starting with the highest-degree terms at each level. With this viewpoint, we can apply it to multinomials throughout an expression in all of the above special forms. Ceberio and Kreinovich [8] discuss greedy algorithms for computing efficient multivariate Hornerized forms.

Another transformation that can enable faster evaluation when substituting numbers for variables is to factor out an integer common divisor of the exponents from a product of powers. For example, if the powers are done with the help of repeated squaring, then

$$(y+3)^6x^4 \rightarrow \left((y+3)^3\,x^2\right)^2$$

uses only five multiplications rather than six.[17]

References

[1] Acton, F.S., *Numerical Methods that Work*, Chapter 11, Harper and Row, 1970 or The Mathematical Association of America, 1990.

[2] Avitzur, R., Milo (a Macintosh computer program), Paracomp Inc., 1988.

[3] Avitzur, R., The Graphing Calculator Story, `http://www.pacifict.com/Story/`

[4] Ayad, M. and Fleischmann, P., On the decomposition of rational functions, *Journal of Symbolic Computation* 43 (4), pp. 259-274, 2008.

[5] Bailey, D. and Borwein, J., Inverse Symbolic Calculator, `http://isc.carma.newcastle.edu.au/advanced`

[6] Barton, D. and Fitch, J.P., A review of algebraic manipulative programs and their application, *The Computer Journal* 15(4), pp. 362-381, 1972.

[7] Bonadio, A., Theorist (a computer program), Prescience Corporation, 1989.

[8] Ceberio, M. and Kreinovich, V., Greedy algorithms for optimizing multivariate Horner schemes, *ACM SIGSAM Bulletin* 38(1), pp. 8-15, 2004.

[17] However, for most systems default simplification automatically distributes the outer exponent 2 over the two factors unless something special is done to prevent it.

[9] Cheze, C. and Galligo, A., Four lectures on polynomial absolute factorization, *Algorithms and Computation in Mathematics* 14, pp. 339-392, 2005.

[10] Cuyt, A.A.M. and Verdonk, B.M., A review of branched continued fraction theory for the construction of multivariate rational approximants, *Applied Numerical Mathematics* 4 (2-4), pp. 263-271, 2005.

[11] Fateman, R.J., Notes on Chebyshev series and computer algebra, 2011, `http://www.cs.berkeley.edu/~fateman/papers/cheby.pdf`

[12] Faugère, J.C. and Perret, L.. High order derivatives and decomposition of multivariate polynomials. *Proceedings of ISSAC 2009*, pp. 207-214.

[13] Giesbrecht, M., Kaltofen, E. and Lee, Wen-shin., Algorithms for computing the sparsest shifts of polynomials via the Berlekamp/Massey algorithm, *Proceedings of ISSAC 2002*, pp. 101-108.

[14] Grigoriev, D.Y., Lakshman, Y.N., Algorithms for computing sparse shifts for multivariate polynomials, *Proceedings of ISSAC 1995*, pp. 96-103.

[15] Gutierrez, J., Rubio, R. and Sevella, D., On multivariate rational function decomposition, *Journal of Symbolic Computation* 33 (5), pp. 545-562, 2002.

[16] Kajler, N. and Soiffer, N., A survey of user interfaces for computer algebra systems, *Journal of Symbolic Computation* 25 (2), pp. 127-159, 1998.

[17] Kozen, D. and Landau, S., Polynomial decomposition algorithms, *Journal of Symbolic Computation* 7 (5), pp. 445-456, 1989.

[18] Krausz, F., A better user interface for Symbolics Lisp Machine Macsyma, Macsyma Newsletter 5(3), pp. 3-5, 1988.

[19] MathMonkeys, LLC, LiveMath, formerly known as Theorist, MathView, MathPlus, and Live Math Maker., `http://www.livemath.com/`

[20] Moses, J, Algebraic simplification: a guide for the perplexed. *Proceedings of the second ACM symposium on symbolic and algebraic manipulation*, pp. 282-304, 1971.

[21] Powers, V. and Wörmann, T., An algorithm for sums of squares of real polynomials, `http://www.mathcs.emory.edu/~vicki/pub/sos.pdf`

[22] Reznick, B.E., *Sums of Even Powers of Real Linear Forms*, Memoirs of the American Mathematical Society, 1992, and `http://www.math.uiuc.edu/~reznick/memoir.html`

[23] Shneiderman, B. and Plaisant, C., *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, 4th edition, Pearson Addison Wesley, p. 367, 2004.

[24] Stoutemyer, D.R., Ten commandments for good default expression simplification, *Journal of Symbolic Computation*, 46(7), pp. 859-887, 2011.

[25] Stoutemyer, D.R., Unit normalization of multinomials over Gaussian integers, *ACM Communications in Computer Algebra* 43 (3/4), pp. 73-76, 2009.

[26] Stoutemyer, D.R, Multivariate partial fraction expansion. *ACM Communications in Computer Algebra* 42 (4), pp. 206-210, 2008.

[27] Stoutemyer, D.R., Useful computations need useful numbers, *ACM Communications in Computer Algebra* 41 (3), pp. 75-99, 2007.

[28] Sturmfels, B., *Algorithms in invariant theory*, 2nd edition, Springer 2008.

[29] Théry, L., Bertot, Y. and Kahn, G., Real theorem provers deserve real user-interfaces, *SDE 5 Proceedings of the fifth ACM SIGSOFT symposium on Software development environments* 17(5), pp. 120-129, 1992.
Preprint at `http://hal.inria.fr/docs/00/07/69/07/PDF/RR-1684.pdf`

[30] Trefethen, L.N. and others, Chebfun Version 4.2, The Chebfun Development Team, 2011, `http://www.maths.ox.ac.uk/chebfun/`

[31] Vandenberghe, L. and Boyd, S., "Semidefinite Programming", *SIAM Review* 38, pp. 49-95, March 1996.

[32] von zur Gathen, J., Functional Decomposition of Polynomials: The Wild Case., *Journal of Symbolic Computation* 10(5): pp. 437-452, 1990.

[33] von zur Gathen, J., Gutierrez, J., Rubio, R., Multivariate Polynomial Decomposition, *Applicable Algebra in Engineering, Communication and Computing* 14(1), pp. 11-31, 2003.

[34] von zer Gathen, J. and Weiss, J., Homogeneous bivariate decompositions, *Journal of Symbolic Computation* 19, pp. 409-434, 1992.

[35] Watt, S.M., Functional Decomposition of Symbolic Polynomials, *Proceedings of the International Conference on Computational Sciences and its Applications*, IEEE Computer Society, pp. 353-362, 2008.

[36] Wikipedia, Model-view-controller,
`http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller#References`

[37] UITP, User Interfaces for Theorem Provers, `http://www.informatik.uni-bremen.de/uitp/`

[38] Wolfram|alpha, `http://www.wolframalpha.com/`

[39] Zippel, R., Rational function decomposition, *Proceedings of ISSAC-91*, pp. 1-6, 1991.

ACM Communications in Computer Algebra, Vol. 47, No. 4, Issue 186, December 2013

ISSAC 2013 Software Presentations

Communicated by Joris van der Hoeven

Arb: a C library for ball arithmetic

Fredrik Johansson *
RISC
Johannes Kepler University
4040 Linz, Austria
fjohanss@risc.jku.at

1 Introduction

Arb [1] is a new open source C library for provably correct arbitrary-precision numerics, extending FLINT [3] (which provides fast arithmetic over various exact rings) to the real and complex numbers. Following the example of iRRAM [7] and Mathemagix [13], Arb performs automatic error propagation using ball arithmetic [12] (not to be confused with heuristic significance arithmetic as used e.g. in Mathematica [9]). This gives performance close to floating-point arithmetic such as provided by MPFR [2] while avoiding the cost at high precision of endpoint-based interval arithmetic as provided for instance by MPFI [8].

One of our motivations for developing a new library has been to provide a low-level, low-overhead interface, and our implementation differs from others in some technical aspects. Arb also provides fast polynomial arithmetic, to our knowledge only available in Mathemagix and without error control in MPFRCX [1], as well as matrix arithmetic. Finally, Arb implements some special functions that have been absent from arbitrary-precision interval software, with performance that compares favorably to available nonrigorous implementations. The presentation covers implementation details and shows some benchmarks.

2 Feature overview

Arb provides the following types:

- `fmpr_t`: floating-point real numbers $\mathbb{R}_D = \mathbb{Z} \times 2^{\mathbb{Z}} \cup \{-\infty, +\infty, \mathrm{NaN}\}$
- `fmprb_t`: real numbers implemented as balls $\mathbb{R}_B = \{[m-r, m+r] : m, r \in \mathbb{R}_D, r \geq 0\}$
- `fmpcb_t`: complex numbers in rectangular form $\mathbb{C}_B = \mathbb{R}_B[i]$
- `fmprb_poly_t`, `fmpcb_poly_t`: polynomials (and truncated power series) over $\mathbb{R}_B$, $\mathbb{C}_B$
- `fmprb_mat_t`, `fmpcb_mat_t`: matrices over $\mathbb{R}_B$, $\mathbb{C}_B$

Each type has a set of associated methods for memory management, conversions, arithmetic and special functions, with an interface resembling that of FLINT. For example, the power series multiplication $a \leftarrow b \times c \bmod x^n$ with rounding to *prec* bits, where a, b, c are of type `fmprb_poly_t`, is written as:

```
fmprb_poly_mullow(a, b, c, n, prec)
```

Polynomial methods have corresponding "underscore" versions that act directly on coefficient arrays, reducing overhead and giving more control over memory allocation and copying (like the `mpn` layer of GMP):

```
_fmprb_poly_mullow(a->coeffs, b->coeffs, b->length, c->coeffs, c->length, n, prec)
```

*Supported by the Austrian Science Fund (FWF) grant Y464-N18.

[1] `http://fredrikj.net/arb/` (Arb is licensed GNU GPL version 2 or later)

3 Representation of numbers

Arb does not directly base its arithmetic on MPFR (but does call MPFR for a few operations, and the test suite extensively verifies correctness against MPFR). MPFR attaches a precision to each variable, and allocates memory for a full-precision number even if only a few bits are used. In Arb, the precision is always passed as an argument to each function; the components of an `fmpr_t` are FLINT integers, and can grow dynamically. A mantissa or exponent with at most 62 bits (30 bits on a 32-bit system) is particularly efficient, as it takes up a single word in the `fmpr_t` struct without allocating memory on the heap.

We have found this approach convenient for mixed-precision algorithms and particularly valuable for computations involving integer coefficients of variable size (such as binary splitting and various polynomial operations). The same type also works well for low-precision arithmetic such as error bound calculations. The drawback is some overhead at precisions up to a few hundred digits, although experiments suggest that this overhead could be reduced with further implementation effort.

Bits	`mpfr_mul`	`fmpr_mul`	`fmprb_mul`
32	1.0	0.6	2.3
128	1.0	1.4	2.7
512	1.0	0.9	1.4
2048	1.0	1.1	1.2
8192	1.0	1.2	1.2
32768	1.0	1.2	1.2
131072	1.0	1.1	1.1
524288	1.0	1.0	1.0

Table 1: Time relative to MPFR of floating-point and ball multiplication. The difference below approximately 1000 bits results from implementation overhead, and the $10\% - 20\%$ difference around 10^3 – 10^5 bits is due to MPFR using the mulhigh algorithm.

An `fmprb_t` consists of a midpoint and a radius, both of type `fmpr_t`. Radius operations use a predefined precision (30 bits). Midpoint arithmetic is always carried out at the requested working precision. It would be more efficient to round midpoints to the accuracy indicated by the radius, though such a normalization naturally can be performed explicitly, and the present convention is sometimes useful for detecting when the computed error bound greatly overshoots the actual numerical error.

Complex numbers are represented as pairs of real balls. This seems preferable to a complex midpoint with a single radius, for reasons of convenience, and it is frequently useful to track whether either the real or imaginary part is exact. Similarly, polynomials and matrices are represented as arrays of coefficients to maximize flexibility. Where a different data order is required, temporary copies are relatively cheap since the base `fmpr_t` type takes up only two words and usually only needs to be copied shallowly.

4 Special functions

Except for some special cases, the elementary functions in Arb call the MPFR implementations of exp, log, sin, cos and atan (our future plan is to develop faster implementations for precisions up to a few thousand digits), using function derivatives to bound propagated errors. Care has been taken to ensure numerically satisfactory behavior on the whole complex plane, for example when evaluating $\tan(x+yi)$ for large $|y|$. Extremely large numbers are handled specially: we allow arbitrary-precision exponents, and we restrict the internal working precision allowed for argument reduction to a small multiple of the requested precision. For example, an attempt to evaluate $\cos(2^{10^{10}})$ quickly returns a crude bounding interval (e.g. $[-1, 1]$) unless the precision is set in the hundreds of millions of digits. This makes worst-case evaluation

time at a given precision predictable and avoids unnecessary stalls caused by tiny terms that might not even contribute to the final result, particularly aiding "black-box" use in computer algebra settings.

Interval software has historically been limited to the elementary functions and some special functions of a real variable, while software with good support for special functions (e.g. [10], [5]) has not guaranteed correctness. We wish to improve this situation. As of the current version, Arb provides Bernoulli numbers, the Hurwitz zeta function $\zeta(s,a)$ and its derivatives with respect to s for complex s and a, and the gamma and digamma functions for real and complex arguments. The implementations are tuned for different sizes and precisions, incorporating many optimizations. Arb also contains code for binary splitting evaluation of generic rational hypergeometric series with automatic error bounding, used for evaluation of mathematical constants, as well as code for rigorous polynomial root refinement, used for some algebraic numbers.

Evaluation	Digits	MPFR 3.1.1	Pari/GP 2.5.3	Mathematica 8.0	Arb
A: γ (Euler's constant)	10^6	93 s	> 1 h	30 s	18 s
B: $\cos(\pi/31)$	10^5	6.1 s	42	12 s	0.48 s
C: ${}_3F_2(\frac{1}{2},\frac{1}{3};\frac{1}{4},\frac{1}{5},\frac{1}{6};\frac{1}{7})$	10^5	n/a	n/a	1396 s	0.45 s
D: $\Gamma(\sqrt{2})$	10^4	60 s	1.9 (233) s	13 s	0.21 (1.3) s
E: $\Gamma(\sqrt{2}+i\sqrt{3})$	10^4	n/a	2.9 (235) s	5.8 (44) s	0.67 (1.7) s
F: $\zeta(1/2+1000i)$	10^4	n/a	24 (1571) s	672 s	22 (25) s
G: $\zeta(1+2i,3+4i)$	10^3	n/a	n/a	2.4 s	0.38 s

Table 2: Special function timings, measuring repeated calls with the initial call inside parentheses. Algorithms in Arb: A) binary splitting B) minimal polynomial root refinement C) generic binary splitting D-E) Stirling's series F-G) Euler-Maclaurin summation.

5 Polynomials and power series

Polynomial operations are implemented in an asymptotically fast way by reducing to multiplication using standard techniques such as Newton iteration for division, series logarithm and series exponential, divide-and-conquer for composition [4], rectangular splitting for power series composition, and product trees for fast multipoint evaluation and interpolation. We have implemented three algorithms for multiplication in $\mathbb{R}[x]$: classical, sloppy, and blockwise. The latter two translate to $\mathbb{Z}[x]$ and call FLINT (which uses classical, Karatsuba, Kronecker substitution, and Schönhage-Strassen FFT multiplication).

The sloppy algorithm cuts off the coefficients of each input polynomial *prec* bits below the top bit of the polynomial as a whole, performs a single multiplication over $\mathbb{Z}[x]$, and bounds errors using max norms. This is fast, and numerically satisfactory if all coefficients have the same magnitude, but not used by default due to the poor numerical stability for polynomials with coefficients of varying magnitude.

The blockwise algorithm splits the input polynomials into blocks of similarly-sized coefficients and multiplies each pair of blocks exactly in $\mathbb{Z}[x]$. In the worst case, this degenerates to multiplication of 1×1 blocks equivalent to classical multiplication. In the typical case, it only performs slightly worse than the sloppy multiplication. Accurate per-coefficient error bounds are computed using an $O(n^2)$ loop running over exponents. The algorithm could be improved further using scaling and by discarding parts of the inputs that do not contribute to the result, as discussed in [11].

We illustrate the importance of polynomial arithmetic that is both fast and numerically stable. Letting $\xi(s) = (s-1)\pi^{-s/2}\Gamma\left(1+\frac{1}{2}s\right)\zeta(s)$, Li's criterion [6] states that the Riemann hypothesis is equivalent to the positivity for all $n > 0$ of the coefficients λ_n defined by $\log \xi\left(z/(z-1)\right) = \sum_{n=0}^{\infty} \lambda_n z^n$. We prove positivity of the first 10,000 coefficients by evaluation. This requires derivatives of $\zeta(s)$, a series logarithm, derivatives of $\log\Gamma(s)$, and a series composition with $z/(z-1)$. In this example, the final composition catastrophically

magnifies the error bounds if sloppy multiplication is used, making a precision of nearly $10n$ bits necessary. With classical multiplication, about $1.3n$ bits suffice, speeding up the the zeta function evaluation, but the subsequent power series operations now dominate. Blockwise multiplication allows using the same precision as with classical multiplication, and the power series operations only take a fraction of the time.

	Sloppy	Classical	Blockwise
Working precision	100000 bits	13000 bits	13000 bits
Zeta	147180 s	1242 s	1272 s
Logarithm	56 s	2760 s	8.3 s
Gamma	781 s	3.4 s	3.4 s
Composition	1994 s	7971 s	185 s
Total	150011 s	11976 s	1469 s

Table 3: Precisions and timings for computing the Li coefficients λ_n up to $n = 10000$ with correctly determined signs, using three different multiplication algorithms.

References

[1] A. Enge. MPFRCX: a library for univariate polynomials over arbitrary precision real or complex numbers, 2012. `http://www.multiprecision.org/index.php?prog=mpfrcx`.

[2] L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier, and P. Zimmermann. MPFR: A multiple-precision binary floating-point library with correct rounding. *ACM Transactions on Mathematical Software*, 33(2):13:1–13:15, June 2007. `http://mpfr.org`.

[3] W. B. Hart. Fast Library for Number Theory: An Introduction. In *Proceedings of the Third international congress conference on Mathematical software*, ICMS'10, pages 88–91, Berlin, Heidelberg, 2010. Springer-Verlag. `http://flintlib.org`.

[4] W. B. Hart and A. Novocin. Practical divide-and-conquer algorithms for polynomial arithmetic. In *Computer Algebra in Scientific Computing*, pages 200–214. Springer, 2011.

[5] F. Johansson et al. *mpmath: a Python library for arbitrary-precision floating-point arithmetic, version 0.17*, 2011. `http://mpmath.org`.

[6] Xian-Jin Li. The positivity of a sequence of numbers and the Riemann Hypothesis. *Journal of Number Theory*, 65(2):325–333, 1997.

[7] N. Müller. The iRRAM: Exact arithmetic in C++. In *Computability and Complexity in Analysis*, pages 222–252. Springer, 2001. `http://irram.uni-trier.de`.

[8] N. Revol and F. Rouillier. Motivations for an arbitrary precision interval arithmetic library and the MPFI library. *Reliable Computing*, 11(4):275–290, 2005. `http://perso.ens-lyon.fr/nathalie.revol/software.html`.

[9] M. Sofroniou and G. Spaletta. Precise numerical computation. *Journal of Logic and Algebraic Programming*, 64(1):113–134, 2005.

[10] The PARI Group, Bordeaux. *PARI/GP, version 2.5.3*, 2012. `http://pari.math.u-bordeaux.fr`.

[11] J. van der Hoeven. Making fast multiplication of polynomials numerically stable. Technical Report 2008-02, Université Paris-Sud, Orsay, France, 2008.

[12] J. van der Hoeven. Ball arithmetic. Technical report, HAL, 2009. `http://hal.archives-ouvertes.fr/hal-00432152/fr/`.

[13] J. van der Hoeven, G. Lecerf, B. Mourrain, P. Trébuchet, J. Berthomieu, D. N. Diatta, and A. Mantzaflaris. Mathemagix: the quest of modularity and efficiency for symbolic and certified numeric computation? *ACM Communications in Computer Algebra*, 45(3/4):186–188, January 2012. `http://mathemagix.org`.

ACM Communications in Computer Algebra, Vol. 47, No. 4, Issue 186, December 2013

NAClab: A Matlab Toolbox for Numerical Algebraic Computation

(extended abstract)

Zhonggang Zeng* Tien-Yien Li†

1 Introduction

We present a Matlab toolbox NAClab for numerical algebraic computation. This toolbox includes Matlab implementations of the basic numerical algorithms in algebraic computations and utility functions. Those functions can be used either directly in applications or as building blocks for implementing advanced computing methods. NAClab is a result of collective effort and its contributors include Liping Chen, Tianran Chen, Wenrui Hao, Tsung-Lin Lee, Andrew Sommese and Wenyuan Wu.

Numerical algebraic computation, particularly numerical polynomial algebra, emerges as a growing area of study in recent years with a solid foundation in place for building numerical and hybrid computing methods [8, 9]. Many robust numerical and numeric-symbolic algorithms have been developed for solving polynomial systems, polynomial factorizations, polynomial GCD, computing dual bases and multiplicity structures of polynomial ideals, etc, with implementations such as in [1, 2, 3, 5, 6, 10] Those algorithms have a broad spectrum of applications in scientific computing such as robotics, control, image processing, computational biology and chemistry, and so on.

One of the main difficulties for numerical algebraic computation is the ill-posedness that frequently occurs when the solution of a problem does not possess Lipschitz continuity with respect to data. Those ill-posed problems are not directly suitable for floating point arithmetic since the solutions are infinitely sensitive to rounding errors. However, it has been shown in recent studies that such a difficulty can be overcome by seeking a regularized numerical solution with a proper formulation. Algorithms implemented in NAClab are designed to regularize the problem for removing ill-posedness rather than extending machine precision for accurate computation. The package also includes generic routines for matrix building and Gauss-Newton iteration that are the main engines for handling ill-posed algebraic computaiton.

NAClab is an on-going project as a major upgrade and expansion from its predecessor ApaTools [13]. At this stage, the main objective is to provide researchers in numerical algebra with generic and versatile tools that simplify and accelerate algorithm development, experimentation, and implementation. We emphasize on achieving the highest possible accuracy and robustness in algorithm design and implementation.

NAClab is maintained at its permanent website[1] and freely accessible to academic researchers and educators for the foreseeable future.

2 Differences between symbolic and numerical algebra

Conventional symbolic computation assumes both data and arithmetic to be exact. In practical applications, however, problem data are likely to be empirical. As a result, exact solutions of those inexact problems may not serve the practical purposes. Using the polynomial

$$p = 81\,x^4 + 16\,y^4 - 648.001\,z^4 + 72x^2y^2 + .002x^2z^2 + .001y^2z^2 - 648x^2 - 288y^2 - .007z^2 + 1296 \qquad (1)$$

*Department of Mathematics, Northeastern Illinois University, Chicago, IL 60625, USA, email: zzeng@neiu.edu.
†Department of Mathematics, Michigan State University, E. Lansing, MI 48824, USA, email: li@math.msu.edu
[1] http://www.neiu.edu/∼naclab

in [7] as an example, Kaltofen proposed the Open Problem 1 in challenges of symbolic computation: Is there factorable polynomial nearby? While polynomial p in (1) is not factorable in conventional sense, it is a perturbed data representation of a factorable polynomial $\tilde{p}$. The objective of numerical factorization is to calculate the factorization of the hidden underlying polynomial $\tilde{p}$ using the imperfect data p. The function `PolynomialFactor` in `NAClab` is built to carry out this computation as follows. First of all, `NAClab` allows polynomials to be entered intuitively into Matlab as strings:

```
>> p = '81*x^4+16*y^4-648.001*z^4+72*x^2*y^2+.002*x^2*z^2+.001*y^2*z^2-648*x^2-288*y^2-.007*z^2+1296';
```

Then the numerical factorization can be obtained using the known relateve data error bound 10^{-5} as the error tolerance:

```
>> F = PolynomialFactor(p,1e-5,'row')     % factor p within error tolerance 1e-5, showing result in a row
F =
1296 * (1-0.25000013640167*x^2-0.11111232357041*y^2+0.707104626181296*z^2) * (1-0.249999863598339*x^2-0.111110989865191*y^2-0.707110027415863*z^2)
```

This result is an accurate approximation to the factorization of the hidden polynomial $\tilde{p}$ and reveals the graph of $p = 0$ is the union of an ellipsoid and a hyperboloid of one sheet.

In contrast to symbolic computation, another subtle issue in numerical algebraic computation is that the given empirical data may have different underlying solutions depending on the error tolerance. This phenominon can be further illustrated in polynomial factorization on

$$f \quad = \quad 0.47619031y + 0.5714288y^2 + 0.55555493x + 1.3809278yx + 0.8571143y^2x + 0.8333328x^2 + yx^2$$

Two numerical factorizations within different error tolerances can be computed by `NAClab` function `PolynomialFactor`:

```
>> PolynomialFactor(f,1e-4,'row')  % within tolerance 1e-4
ans =
0.999996404282881 * (0.833329772039985 + y)  * (0.666677452701414 + x)  * (0.85712030479082*y + x)

>> PolynomialFactor(f,1e-6,'row')  % within tolerance 1e-6
ans =
1.00000001100681 * (0.833332777619492 + y)  * (0.571428774473131*y + 0.666666352807009*x + 0.857114290319387*y*x + x^2)
```

They are accurate approximations to the factorizations of two nearby polynomials from the same data.

Many other algebraic computations follow a similar pattern: An accurate solution of an algebraic problem is to be computed but the problem data are imperfect so that the exact solution is meaningless since the solution is infinitely sensitive to data perturbations. The objective of the numerical algebraic computation is to solve the problem using the slightly perturbed data within an error tolerance similar to the factorization example above. `NAClab` is built for this purpose.

Algebraic problems are often ill-posed because the set of problems whose solutions possessing a distinct structure form a manifold of positive codimension, and perturbations generically pushes a given problem away from the manifold. Our strategy starts with formulating the *numerical solution* of an ill-posed algebraic problem following a "three strikes" principle consisting of *backward nearness, maximum codimension* and *minimum distance* [13] for removing the ill-posedness. Based on those formulation principles, computing the numerical solution can be carried out in two optimization processes: maximizing the codimension of manifolds followed by minimizing the distance to the manifold, leading to a two-staged strategy for designing robust algorithms.

The main mechanism at Stage I is matrix rank-revealing, while Stage II relies on solving nonlinear least squares problems. In `NAClab`, we provide matrix building/computation tools for Stage I and nonlinear least squares tools for Stage II.

3 NAClab overview

NAClab originated from its predecessor ApaTools [13]. Among many improvement areas, we implemented a user-friendly mechanism for direct polynomial manipulations and included a major package in numerical solution of polynomial systems by homotopy continuation method. For example, solving the polynomial system

$$x^5 - y^5 + 3y + 1 \quad = \quad 5y^4 - 3 \quad = \quad 20x - y + z \quad = \quad 0$$

can now be carried out in a simple call:

```
>> P = {'x^5-y^5+3*y+1','5*y^4-3','20*x-y+z'};    % enter the polynomial system directly and intuitively
>> [Solutions, variables] = psolve(P)             % call the polynomial system solver and obtain all the isolated solutions
   Solutions =
     Column 1
     0.778497746685646 + 0.893453081179308i
    -0.000000000000000 + 0.880111736793394i
   -15.569954933712914 -16.988949886792764i
     ... ...
     Column 20
     0.778497746685646 - 0.893453081179308i
     0.000000000000000 - 0.880111736793393i
   -15.569954933712925 +16.988949886792767i

   variables =
    'x'    'y'    'z'
```

Compared to Maple and Mathematica, Matlab has an advantage in efficient numerical matrix computations with an disadvantage in user interface. Cumbersome representations are required for carrying out polynomial and other algebraic computations. Using NAClab, polynomials can be entered and displayed as character strings in Matlab in a way similar to Maple.

```
>> f = '3*x^2 - (2-5i)*x^3*y^4 - 1e-3*z^5-6.8'
>> g = '-2*y^3 - 5*x^2*z + 8.2'
```

Using such an intuitive polynomial representation, users can now perform common polynomial operations such as addition, multiplication, power, evaluation, differentiation, factorization, extracting coefficients, finding greatest common divison (GCD), etc, by calling NAClab functions, such as

```
>> p = pplus('2*x^5-3*y','4+x*y')           % add any number of polynomials
>> q = ptimes(f,g,h)                        % multiply any number of polynomials
>> v = PolynomialEvaluate(f,{'x','z'},[3,4]) % evaluate f(x,y,z) for x=3, z=4

>> % and a lot more.  For example, to compute a greatest common divisor:
>> f = '10 - 5*x^2*y + 6*x*y^2 - 3*x^3*y^3';
>> g = '30 + 10*y + 18*x*y^2 + 6*x*y^3'
>> u = PolynomialGCD(f,g)
u =
33.5410196624968 + 20.1246117974981*x*y^2
```

In summary, NAClab is developed for numerical algebraic computations in Matlab including solving polynomial systems, polynomial factorizations, polynomial greatest common divisors, multiplicity and dual spaces of nonlinear systems at isolated zeros, numerical Jordan Canonical Forms, and numerical rank revealing. The package also contains a comprehensive library of programming utilities for building further algorithms for numerical algebraic computations.

NAClab is an on-going project. While continuing to refine the existing functions, we shall expand the package by developing more algorithms and their implementations for numerical algebraic computations.

References

[1] D.J. Bates, C. Peterson and A.J. Sommese, *A numerical-symbolic algorithm for computing the multiplicity of a component of an algebraic set*, IEEE Trans. Signal Processing, 52 (2003), pp. 3394–3402.

[2] D.J. Bates, J.D. Hauenstein, A.J. Sommese and C.W. Wampler II, *Software for numerical algebraic geometry: A paradigm and progress towards its implementation*, in Software for Algebraic Geometry, IMA Volume 148, M. Stillman, N. Takayama, and J. Verschelde, eds., Springer, 2008, pp. 1–14.

[3] R. M. Corless, S. M. Watt, and L. Zhi, *QR factoring to compute the GCD of univariate approximate polynomials*, IEEE Trans. Signal Processing, 52 (2003), pp. 3394–3402.

[4] B. Dayton, T.Y. Li and Z. Zeng, *Multiple zeros of nonlinear systems* Mathematics of Computation, Vol. 80, pp. 2143-2168, 2011

[5] S. Gao, E. Kaltofen, J. May, Z. Yang, and L. Zhi, *Approximate factorization of multivariate polynomials via differential equations.* Proc. ISSAC '04, ACM Press, pp 167-174, 2004.

[6] C.-P. Jeannerod and G. Labahn, *The SNAP package for arithemetic with numeric polynomials.* In International Congress of Mathematical Software, World Scientific, pages 61-71, 2002.

[7] E. Kaltofen, *Challenges of symbolic computation: My favorite open problems*, J. Symb. Comput., 29, pp.161-168, 2000.

[8] A.J. Sommese and C.W. Wampler II, *The Numerical Solution of Systems of Polynomials*, World Scientific Pub., Hackensack, NJ. 2005

[9] H. J. Stetter, *Numerical Polynomial Algebra*, SIAM, 2004.

[10] J. Verschelde, *Algorithm 795: PHCpack: A general-purpose solver for polynomial systems by homotopy continuation*, ACM Trans. on Math. Software, 25(1999), pp. 251–276.

[11] Z. Zeng, *A polynomial elimination method for symbolic and numerical computation.* 409(2008) pp. 318-331.

[12] Z. Zeng, *Computing multiple roots of inexact polynomials*, Math. Comp., 74 (2005), pp. 869–903.

[13] Z. Zeng, *ApaTools: A Maple and Matlab toolbox for approximate polynomial algebra*, in Software for Algebraic Geometry, IMA Volume 148, M. Stillman, N. Takayama, and J. Verschelde, eds., Springer, 2008, pp. 149–167.

[14] Z. Zeng and B. Dayton, *The approximate GCD of inexact polynomials. II: A multivariate algorithm.* Proceedings of ISSAC'04, ACM Press, pp 320-327. (2006).

ACM Communications in Computer Algebra, Vol. 47, No. 4, Issue 186, December 2013

The new `GroupTheory` package in Maple 17

E.J. Postma

Abstract

The `GroupTheory` package was added to Maple 17 [1]. It deals with discrete groups. The package offers visualizations of group theoretic concepts; accepts input and generates output that is relatively close to the classical textbook notation; and can deal with parameterized groups where the parameters are given only symbolically.

Maple [1] is a well-known computer algebra package, initially developed in the early 1980s by the Symbolic Computation Group at the University of Waterloo. Maple has had a package dealing with group theory dating back to the first decade of its existence. It is called `group` and has been showing its limitations for a long time. For Maple 17, a completely new package called `GroupTheory` was added that replaces the older package. It was written at Maplesoft, incorporating many ideas and a substantial amount of code and data from two earlier packages written by the Computer Algebra Group at Simon Fraser University. One is a library of group presentations and permutation representations, written by Vahid Dabbaghian. The other package [2] focuses on visualization (it contained most of the code described in Section 4) and also performs substantial computation; it was written by Asif Zaman and Michael Monagan.

Other notable software packages dealing with discrete groups are GAP [3], Magma [4], and Mathematica [5]'s group theory features. We believe that, while some features offered by these packages are similar, each of the Sections 2, 3, and 4 describes some functionality that is unique to Maple.

1 Basic functionality

Groups can be represented in one of five ways: by permutations, by generators and relations, by an explicit multiplication table, as a set with manually defined group operations, and as an abstract group where elements cannot be listed but only properties computed. (This last option is explored in Section 2.) There is a substantial library of groups, containing all small groups of up to 200 elements (numbered consistently with the small group databases in GAP and Magma), all simple groups, many linear groups, and many individually named groups. These groups can typically be constructed in multiple representations: for example, using the `DihedralGroup` command, one can construct either the permutation or the finitely presented representation.

Such a group can then be interrogated about its properties, such as its order, whether it is simple, or its Fitting subgroup. Isomorphism testing between arbitrary groups is also supported. In total, the package contains about 120 commands for constructing and interrogating groups.

2 Symbolic groups

The `GroupTheory` package can deal with groups where it cannot list any elements explicitly. There are two such classes of groups: those where the group is defined only up to a symbolic parameter (such as `DihedralGroup(n)` or $\mathbf{D}_n$, the dihedral group with $2n$ elements), and those where listing elements is merely highly impractical, such as the Monster group $\mathbb{M}$.

For groups of the latter kind, Maple simply has many properties stored explicitly. Here are a few examples concerning the Monster group:

```
> GroupOrder(Monster());
```

$$808017424794512875886459904961710757005754368000000000$$

```
> IsSimple(DirectProduct(Monster(), TrivialGroup()));
```

true

```
> IsSimple(DirectProduct(Monster(), CyclicGroup(2)));
```

false

For groups defined with a symbolic parameter, Maple has some predefined properties as for the purely symbolic groups, but it can also deduce some properties from assumptions made on the parameters:

```
> IsNilpotent(DihedralGroup(6*n)) assuming n :: posint;
```

true

3 Interface

The `GroupTheory` package attempts to accept input and generate output in typography that is as close as possible to what one would traditionally find it in a textbook.

The requirements for input are clearly much stricter than for output, in that the input has to be processed by the parser for the general Maple language. Nonetheless, the package understands as input finitely presented groups in the format

$$\langle g_1, \ldots, g_n | r_1, \ldots, r_k \rangle,$$

where $g_1, \ldots, g_n$ are the generators and $r_1, \ldots, r_k$ are the relations.

For output, there is much more functionality. A few examples follow.

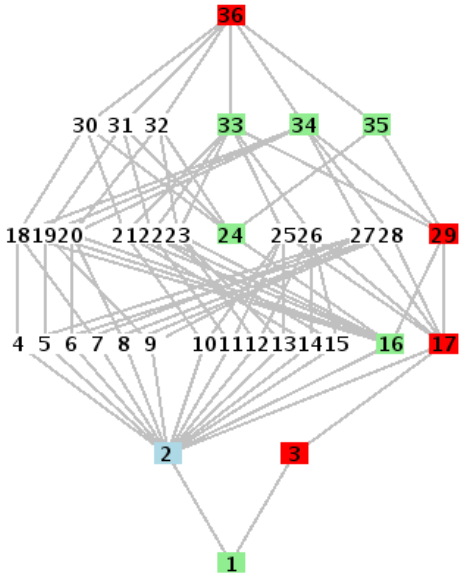


Figure 1: The subgroup lattice of the small group (48, 8). Highlighted in light blue is the centre, in light green are the normal subgroups, and in red is the lower central series.

Input and output	Discussion
`> DirectProduct(Monster(), Symm(n));` $\mathbb{M} \times \mathbf{S}_n$	The direct product of the monster group and the symmetric group of order n.
`> DerivedSeries(Symm(4));` $\mathbf{S}_4 \triangleright \langle (1,3,2),(2,4,3) \rangle \triangleright \langle (1,3)(2,4),(1,4)(2,3) \rangle \triangleright \langle \rangle$	The derived series of the symmetric group of order 4.

`> g := PSL(2, 3);`
 $g \coloneqq \mathbf{PSL}(2,3)$
 `> h := SylowSubgroup(3, g);`
 $h \coloneqq \langle$a permutation group on 4 letters$\rangle$
 `> k := Normaliser(h, g);`
 $k \coloneqq N_{\mathbf{PSL}(2,3)}(\langle$a permutation group on 4 letters$\rangle)$
 `> GroupOrder(h);`
 3
 `> k;`
 $N_{\mathbf{PSL}(2,3)}(\langle (2,3,4) \rangle)$
	The normalizer in $\mathbf{PSL}(2,3)$ of a Sylow-3 subgroup, h. Note that the generators of h are computed lazily, but once they are computed (in order to compute the order of h), they are remembered. This is discussed in Section 5.

4 Visualization

The `GroupTheory` package supports two visualizations: a subgroup lattice and a Cayley table.

An example of the subgroup lattice visualization can be found in Figure 1. This figure was obtained with the commands

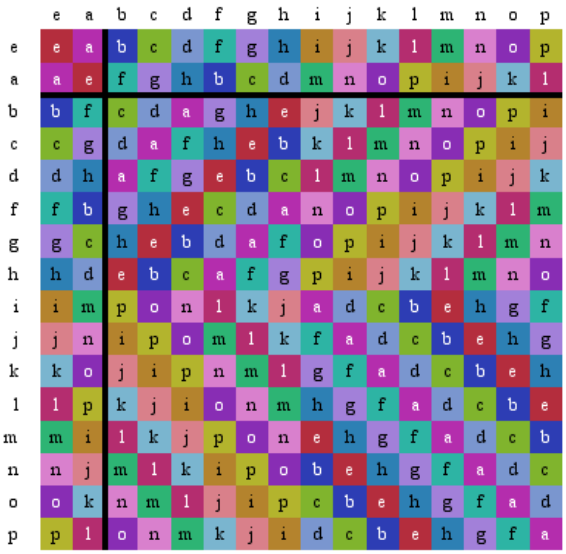


Figure 2: The Cayley table of the dicyclic group of order 16.

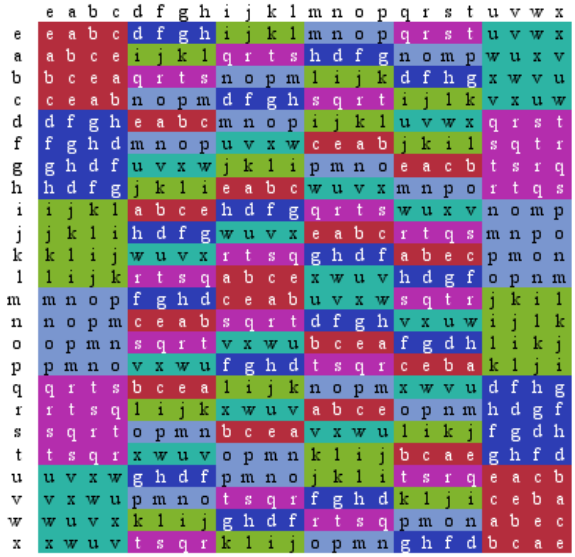


Figure 3: The Cayley table of a group of order 24.

```
> g := SmallGroup(48, 8):
> DrawSubgroupLattice(g, highlight=LowerCentralSeries(g));
```

Conjugate subgroups are placed next to each other, a little closer together than non-conjugate subgroups. This already shows, for example, what the normal subgroups are: the groups of size 1. For extra emphasis, any subset of subgroups can be highlighted in any combination of colours; the default is the centre and all normal subgroups. In the example, the lower central series is additionally highlighted (overriding the colours for the normal subgroups).

Two examples of the Cayley table visualization are shown in Figures 2 and 3. In Figure 2, the centre of the group is indicated by a thick black line. It consists of two elements. In Figure 3, the elements are highlighted depending on the coset of a particular subgroup they occur in.

5 Performance: memoization and autocompiled code

Groups are represented by objects (a type of modules). This provides part of the dispatching logic that selects the appropriate piece of code when a property needs to be computed for a given tuple of arguments. Another part is a memoization feature: most properties that are defined on a single argument, such as group order or simplicity, are computed only when necessary, but are remembered. That is, once such a property is computed, it is stored in a hash table in the object, with the property name as the key, and subsequent computations take advantage of this by looking it up rather than recomputing. A demonstration of this functionality is contained in Section 3.

For many of the low level algorithms, it was decided that the best tradeoff between performance and ease of programming was obtained by writing them in the subset of the Maple language that can be directly compiled into C code, and using the `autocompile` option. This means that the code is compiled on the fly.

As an example of these techniques, let us examine the `IdentifySmallGroup` command. It is used for identifying the isomorphism type of small groups (presently of size 200 or less) and uses the same numbering as its GAP and Magma equivalents. There are many shortcuts, for example if the group order n is prime. If

these do not apply, the algorithm determines, for each possible order $d|n$, the numbers of group elements of order d, and additionally the number of elements of the derived subgroup. These numbers are determined from the Cayley table of the group by autocompilable code. It then evaluates a perfect hash function on this sequence of numbers to find all groups that share these characterstics. (The hash function value is one of the values remembered as described in Section 5.) Subsequently, as long as there is more than one candidate left, the code uses its general isomorphism testing command `AreIsomorphic`.

References

[1] *Maple 17*. Maplesoft, a division of Waterloo Maple Inc., Waterloo, Ontario.

[2] Asif Zaman and Michael Monagan, *Visualizing Groups in Maple* (poster), `http://www.cecm.sfu.ca/research/posters/zaman09.pdf`.

[3] The GAP Group, *GAP – Groups, Algorithms, and Programming, Version 4.6.3*; 2013, `(http://www.gap-system.org)`.

[4] Wieb Bosma, John Cannon, and Catherine Playoust, *The Magma algebra system. I. The user language*, J. Symbolic Comput., 24 (1997), 235265.

[5] Wolfram Research, Inc., *Mathematica, Version 9.0*, Champaign, IL (2012).

ACM Communications in Computer Algebra, Vol. 47, No. 4, Issue 186, December 2013

Holonomic Functions in Mathematica

Christoph Koutschan
Johann Radon Institute for Computational and Applied Mathematics
Austrian Academy of Sciences
Altenberger Straße 69, A-4040 Linz, Austria
christoph.koutschan@ricam.oeaw.ac.at

Abstract

We present the Mathematica package HolonomicFunctions which provides a powerful framework for the automatic manipulation of multivariate holonomic functions, in the spirit of Zeilberger's holonomic systems approach. Its top-level functionalities are: converting a mathematical expression into a holonomic description, executing holonomic closure properties, and creative telescoping for general holonomic functions. To achieve these goals, many other, lower-level, functionalities had to be implemented which were not available in the Mathematica system: finding rational solutions of linear systems of (q-) difference / differential equations, noncommutative arithmetic in Ore algebras and computing Gröbner bases in such domains.

In his seminal paper *A holonomic systems approach to special functions identities* [11], Zeilberger writes: "I hope that more professional programmers and algorithmic designers will soon expand the rudimentary ideas in this paper and develop a symbolic software package to prove general special function identities." Meanwhile, his dream has certainly become true: numerous papers refining and extending his "rudimentary ideas" have appeared, new and faster algorithms have been developed (some of them by Zeilberger himself), and there are several software packages available which implement (at least parts of) his holonomic systems approach.

Our package HolonomicFunctions is definitely one of the most general ones, since it allows to "prove general special function identities", as anticipated by Zeilberger. As special cases, our package contains his fast algorithm for proving hypergeometric identities [10] and its q-analogue, as well as the Almkvist-Zeilberger algorithm [1] for evaluating integrals of hyperexponential functions. Moreover, it can likewise deal with sums (resp. integrals) of functions which don't satisfy first-order recurrences (resp. differential equations), but higher-order ones. The only other software package we know of, providing this degree of generality, is Chyzak's Mgfun package for Maple. Our main motivations to reimplement the algorithms already contained in Mgfun were 1) to make them available to Mathematica users, 2) to provide an independent implementation in a different computer algebra system, 3) and to create an easily accessible, user-friendly interface to methods that are relevant to many disciplines outside of computer algebra. Additionally, we implemented some very recent algorithms for creative telescoping [7, 2] and closure properties [5].

Due to space restrictions, we will limit this short exposition to the top-level functionalities of HolonomicFunctions, which are: 1) converting a mathematical expression into a holonomic description, 2) executing holonomic closure properties, and 3) creative telescoping for general holonomic functions. More details, also about lower-level functionalities, can be found in [6], and detailed descriptions of all available commands are listed in the user's guide [8]. These documents, the package itself, and a collection of examples can be downloaded from the website

http://www.risc.jku.at/research/combinat/software/HolonomicFunctions/

(the required password is given for free to researchers and non-commercial users). HolonomicFunctions has first been released in 2009 and since then it has been extended constantly.

In order to apply the holonomic systems approach to some special function identity, the first step consists in converting all input expressions to holonomic descriptions (strictly speaking, our package works with descriptions of *∂-finite functions* [3], but for sake of simplicity, we don't want to insist on this subtle difference for now). This means, for a given mathematical function f, determine all linear partial (q-) difference / differential equations with polynomial coefficients that f satisfies ("holonomic system"). This usually infinite set of equations has the structure of a left ideal in the corresponding operator algebra; we use *Ore algebras* for representing such equations. The command to obtain an *annihilating ideal* (ideal of annihilating operators) for f is **Annihilator**; it takes as input a mathematical expression f, together with a list of operator symbols like D_x, S_n, etc. to specify the type of equations, and outputs the reduced left Gröbner basis of an annihilating ideal for f (note, however, that it is not guaranteed to be the maximal one so that it may not contain all annihilating operators for f). Following [4], we have extended our package such that it can also deal with certain non-holonomic functions. The **Annihilator** command recognizes more than 100 elementary and special functions (holonomic and non-holonomic) that are available in the Mathematica system.

The class of holonomic functions exhibits some nice and useful closure properties, among them addition, multiplication, certain substitutions and application of operators: given annihilating ideals of functions f and g, respectively, there are algorithms for computing an annihilating ideal of $f + g$, $f \cdot g$, etc. Since these operations are implemented on the level of ∂-finite functions, the corresponding commands are **DFinitePlus**, **DFiniteTimes**, **DFiniteSubstitute**, and **DFiniteOreAction**. We refer to [6] where these closure properties are explained in detail. We want to emphasize that the **Annihilator** command analyzes the syntactical structure of an input expression and then makes use of the above closure properties, e.g., multiplication in the input line In[2], see below.

It is a classic result that holonomic functions are also closed under taking integrals and sums; for these operations, the method of *creative telescoping* is employed. The problem of computing creative telescoping relations (consisting of the *telescoper* and the *certificate*) has attracted a great deal of attention during the last years. Zeilberger's solution in [11] (later coined "the slow algorithm") is based on elimination; in HolonomicFunctions it can be achieved via the **OreGroebnerBasis** command or the **FindRelation** command. The latter finds an element in a given left ideal that satisfies certain properties, to be specified by the user. Takayama's algorithm [9] is also based on elimination, but is more efficient than the slow algorithm; the command **Takayama** exports it to the user. The command **CreativeTelescoping** computes telescopers and certificates using Chyzak's algorithm [3], whereas the command **FindCreativeTelescoping** executes the heuristic fast approach proposed in [7]. Very recently, a creative telescoping algorithm for bivariate hyperexponential functions based on Hermite reduction [2] has been implemented by the author; it will be available in the next release of HolonomicFunctions.

At the end, we come back to Zeilberger's seminal paper [11] and demonstrate the usage of our package on the toy example that he discusses in the introduction, namely the generating function of the venerable Legendre polynomials:

$$\sum_{n=0}^{\infty} P_n(x)t^n = (1 - 2xt + t^2)^{-1/2}.$$

We start by computing an annihilating ideal for the summand on the left-hand side, and perform creative telescoping on it; this gives a list of telescopers and corresponding certificates:

```
In[1]:= << HolonomicFunctions.m
        HolonomicFunctions package by Christoph Koutschan, RISC-Linz, Version 1.6 (12.04.2012)

In[2]:= ann = Annihilator[LegendreP[n, x]*t^n, {S[n], Der[t], Der[x]}]

Out[2]= {tDt − n, (n + 1)Sn + (t − tx²)Dx + (−ntx − tx), (x² − 1)Dx² + 2xDx + (−n² − n)}
```

In[3]:= **{ts, cs} = CreativeTelescoping[ann, S[n] − 1, {Der[t], Der[x]}]**

Out[3]= $\left\{\left\{(-t^2+2tx-1)D_x+t,\ (t^2-2tx+1)D_t+(t-x)\right\},\left\{(tx-1)D_x-nt,\ (x-1)(x+1)D_x+\frac{n-ntx}{t}\right\}\right\}$

The correctness of this result can be verified by showing that the creative telescoping operators are indeed in the annihilating ideal (done here only for the first one, note the usage of noncommutative arithmetic):

In[4]:= **ct1 = ts[[1]] + (S[n] − 1) ** cs[[1]]**

Out[4]= $(tx-1)S_nD_x-(n+1)tS_n+(tx-t^2)D_x+(nt+t)$

In[5]:= **OreReduce[ct1, ann]**

Out[5]= 0

Finally, one computes an annihilating ideal for the right-hand side: it agrees with the left ideal generated by the two telescopers above.

In[6]:= **Annihilator[(1 − 2*x*t + t^2)^(−1/2), {Der[t], Der[x]}]**

Out[6]= $\left\{(t^2-2tx+1)D_x-t,\ (t^2-2tx+1)D_t+(t-x)\right\}$

Comparing initial values (not done here) completes the proof.

References

[1] Gert Almkvist and Doron Zeilberger. The method of differentiating under the integral sign. *Journal of Symbolic Computation*, 10(6):571–591, 1990.

[2] Alin Bostan, Shaoshi Chen, Frédéric Chyzak, Ziming Li, and Guoce Xin. Hermite reduction and creative telescoping for hyperexponential functions. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC)*, New York, NY, USA, 2013. ACM. To appear (preprint on arXiv:1301.5038).

[3] Frédéric Chyzak. An extension of Zeilberger's fast algorithm to general holonomic functions. *Discrete Mathematics*, 217(1-3):115–134, 2000.

[4] Frédéric Chyzak, Manuel Kauers, and Bruno Salvy. A non-holonomic systems approach to special function identities. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC)*, pages 111–118, New York, NY, USA, 2009. ACM.

[5] Stavros Garoufalidis and Christoph Koutschan. Twisting q-holonomic sequences by complex roots of unity. In Joris van der Hoeven and Mark van Hoeij, editors, *Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC)*, pages 179–186. ACM, 2012.

[6] Christoph Koutschan. *Advanced applications of the holonomic systems approach*. PhD thesis, Research Institute for Symbolic Computation (RISC), Johannes Kepler University, Linz, Austria, 2009.

[7] Christoph Koutschan. A fast approach to creative telescoping. *Mathematics in Computer Science*, 4(2-3):259–266, 2010.

[8] Christoph Koutschan. HolonomicFunctions (user's guide). Technical Report 10-01, RISC Report Series, Johannes Kepler University, Linz, Austria, 2010. http://www.risc.jku.at/research/combinat/software/HolonomicFunctions/.

[9] Nobuki Takayama. An algorithm of constructing the integral of a module—an infinite dimensional analog of Gröbner basis. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC)*, pages 206–211, New York, NY, USA, 1990. ACM.

[10] Doron Zeiberger. A fast algorithm for proving terminating hypergeometric identities. *Discrete Mathematics*, 80(2):207–211, 1990.

[11] Doron Zeilberger. A holonomic systems approach to special functions identities. *Journal of Computational and Applied Mathematics*, 32(3):321–368, 1990.

ACM Communications in Computer Algebra, Vol. 47, No. 4, Issue 186, December 2013

Classifying Discrete Objects with *Orbiter*

Anton Betten

Abstract

Orbiter is a software package to classify discrete objects such as designs, graphs, codes, and objects from finite geometry. It employs the method of *breaking the symmetry* to attack difficult problem instances by means of subobjects that serve as a stepping stone. The algorithms combine techniques from Group Theory and from Combinatorics. *Orbiter* is a library of C++ functions that provide functionality for Discrete Mathematics. In order to be applied to a specific problem, code has to be written taylored to the specific application.

The Experimental Approach

Research in Mathematics often benefits from examples. There are many areas where the existing theory does not suffice to explain all the examples that are known. Therefore, studying examples is often the first step in finding new constructions or new theory. This new theory helps to explain where the known examples come from, and often predicts more examples for larger instances of the parameter set. In some cases, infinite families of objects can be constructed. In order to help with this process, knowing examples and their automorphism groups is crucial. In Discrete Mathematics and Combinatorics, we are able to examine (at least for small cases) complete lists of objects up to isomorphism. This kind of data is very valuable to researchers. *Orbiter* is intended to assist with these classification problems. One could call this the *experimental approach* to mathematics. Designs, graphs and codes are related topics, and all are well-suited to computer investigation. The various links between these areas are stressed in [7].

Classifying all orbits of a permutation group means listing a set of representatives for the orbits, together with their respective stabilizer subgroups. Also, if an object is given, we can compute a group element that maps the given object to its representative in the classification. These kinds of problems are notoriously difficult, since they involve the isomorphism problem as a subproblem and isomorphism is usually NP-hard.

Breaking The Symmetry

While group theoretic algorithms to classify orbits are available, experience shows that many problems require a combination of methods to be solved efficiently. The method of *breaking the symmetry* allows to classify objects using subobjects that serve as a stepping stone. The subobjects are easier to classify, and it is possible to lift the classification of subobjects to the classification of the original objects. The method combines group theory with traditional "solvers." Here, we understand solvers as any kind of computational primitive to solve the problem of *finding* all objects that arise from a given *starter object*. Once these objects have been found, an additional isomorph rejection step is performed to solve the classification problem. The theory behind this is explained in [2], where the method is applied to the classification of packings in $\mathrm{PG}(3,3)$. It is important to realize that the method is very general, and can be applied to broad classes of problems.

The use of subobjects is well-known. In [11], homomorphisms of group actions are discussed. In [5] and [6], the technique of "breaking the symmetry" is developed. In [16], an algebraic algorithm to compute the orbit decomposition is presented. In *Orbiter*, many of these algorithms are combined, and an isomorph classification module based on the theory described in [2] is present. *Orbiter* offers some algorithms to solve these systems of equations but also allows to interface with third party software. The communication between *Orbiter* and the outside solvers can happen through files. Once the data from the solver is received, the isomorph classification module starts its job and computes the final list of isomorphism types together with the stabilizer groups. Data from the classification is stored in files to allow identifying objects of the givem type at a later point in time. This means that given an object, a group element can be computed that maps the object to one of the representatives from the classification. An implementation of Knuth's dancing links (DLX) [10] is available. Wassermann's algorithm [17] or any other suitable piece of software can be used as an external solver.

The underlying idea behind *Orbiter* is to provide isomorph classification for a variety of types of objects. To be able to handle things uniformly, we rely on the use of C++ function pointers to realize permutation group algorithms for arbitrary objects. The only requirement is that the object can be represented as a set (or set of sets). The entries of the set are integers that represent the components of the object. For instance, when classifying sets of points in a finite projective space subject to certain conditions, the components are projective points, and they are represented numerically. When classifying combinatorial designs, the objects consist of sets of subsets of a set X. These subsets are known as blocks, and they form the components of the object. We can use the lexicographic ordering of subsets of X to identify blocks with integers. The process of converting components into integers and integers into components is called *ranking* and *unranking*. Sometimes, the terms indexing or enumerating are used also. Basically, the possible components are mapped bijectively to an interval of integers. We require that rank and unrank functions to encode the object under consideration are available. For may types of combinatorial objects, such functions exist or can be devised easily. Using this kind of methodology, *Orbiter* is able to realize permutation groups acting on objects. The permutation group algorithms and the functionality for the specific object are completely separate. The group does not know what objects it is acting on, and the objects does not know what group is acting on them. This methodology makes the code easily adaptable to different actions. The most basic group actions are that of the symmetric group acting on a set, and the projective linear group acting on projective space (as well as the affine group acting on a vector space). From these basic actions, one can define induced actions in various ways. For instance, the symmetric group induces an action on the k-subsets. The projective linear group induces an action on the Grassmannian of subspaces. Many other induced actions are available.

Orbiter's predecessor is DISCRETA [4], which is specialized to t-designs with prescribed groups of automorphisms, and comes with a graphical user interface. Since *Orbiter* applies to a much more general class of problems, there was no longer the possibility for a graphical user interface. Instead, the user of *Orbiter* will have to write code to facilitate the algorithms that are provided. *Orbiter* is available from the website [15].

Applications

Recently, *Orbiter* has been used to classify packings [2], unitals [1] and BLT-sets [3]. Other applications exist. Some are described in the Orbiter Manual.

Other Work

A general reference for the problem of classifying designs and codes is [9]. This book emphasizes the use of canonical forms, facilitated for instance via the software package nauty [14], or the partition backtrack approach [12]. Both of these algorithms are available through the computer algebra system MAGMA [13]. Nauty is also available through *Orbiter*. A different computer algebra system with an emphasis on Group Theory is GAP [8].

References

[1] John Bamberg, Anton Betten, Cheryl Praeger, and Alfred Wassermann. Unitals in the Desarguesian Projective Plane of Order Sixteen. To appear in Journal of Statistical Planning and Inference.

[2] Anton Betten. The Packings of PG$(3, 3)$. Submitted to Journal of Combinatorial Designs.

[3] Anton Betten. Rainbow Cliques and the Classification of Small BLT-Sets. Accepted for the Proceedings of ISSAC 2013.

[4] Anton Betten, Reinhard Laue, and Alfred Wassermann. DISCRETA, a tool for constructing t-designs. In: Computer Algebra Handbook, Edited by Johannes Grabmeier, Erich Kaltofen, Volker Weispfennig, Springer 2003, pp 372-375.

[5] Cynthia A. Brown, Larry Finkelstein, and Paul Walton Purdom, Jr. Backtrack searching in the presence of symmetry. In *Applied algebra, algebraic algorithms and error-correcting codes (Rome, 1988)*, volume 357 of *Lecture Notes in Comput. Sci.*, pages 99–110. Springer, Berlin, 1989.

[6] Cynthia A. Brown, Larry Finkelstein, and Paul Walton Purdom, Jr. Backtrack searching in the presence of symmetry. *Nordic J. Comput.*, 3(3):203–219, 1996.

[7] P. J. Cameron and J. H. van Lint. *Designs, graphs, codes and their links*, volume 22 of *London Mathematical Society Student Texts*. Cambridge University Press, Cambridge, 1991.

[8] GAP – Groups, Algorithms, and Programming, Version 4.4. The GAP Group, Aachen, Germany and St. Andrews, Scotland, 2004.

[9] P. Kaski and P. Östergård. *Classification algorithms for codes and designs*, volume 15 of *Algorithms and Computation in Mathematics*. Springer-Verlag, Berlin, 2006.

[10] D. E. Knuth. Dancing links. *eprint arXiv:cs/0011047*, November 2000. in Davies, Jim; Roscoe, Bill; Woodcock, Jim, Millennial Perspectives in Computer Science: Proceedings of the 1999 Oxford-Microsoft Symposium in Honour of Sir Tony Hoare, Palgrave, pp. 187-214.

[11] R. Laue. Construction of combinatorial objects—a tutorial. *Bayreuth. Math. Schr.*, 43:53–96, 1993. Konstruktive Anwendungen von Algebra und Kombinatorik (Bayreuth, 1991).

[12] J.S. Leon. Partitions, refinements, and permutation group computation. In *Groups and computation, II (New Brunswick, NJ, 1995)*, volume 28 of *DIMACS Ser. Discrete Math. Theoret. Comput. Sci.*, pages 123–158. Amer. Math. Soc., Providence, RI, 1997.

[13] Magma. The Computational Algebra Group within the School of Mathematics and Statistics of the University of Sydney, 2004.

[14] Nauty User's Guide (Version 2.4), Brendan McKay, Australian National University, Nov 4, 2009.

[15] Orbiter – A Program To Classify Discrete Objects,
http://www.math.colostate.edu/∼betten/orbiter/orbiter.html, Anton Betten, 2013.

[16] B. Schmalz. t-Designs zu vorgegebener Automorphismengruppe. *Bayreuth. Math. Schr.*, 41:1–164, 1992. Dissertation, Universität Bayreuth, Bayreuth, 1992.

[17] Alfred Wassermann. Finding simple t-designs with enumeration techniques. *J. Combin. Des.*, 6(2):79–90, 1998.

Jenks Prize 2013 Award Citation

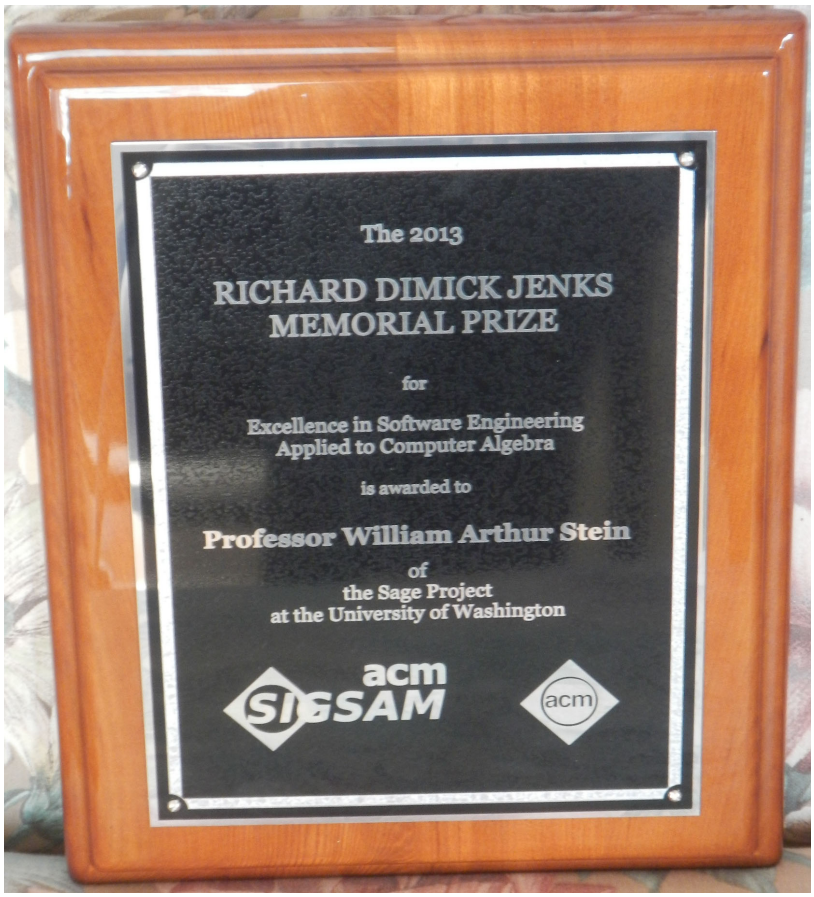
Communicated by Erich Kaltofen

The 2013 Richard D. Jenks Memorial Prize for Excellence in Software Engineering Applied to Computer Algebra was announced by members of the Prize Selection Committee, Mark Giesbrecht representing its chair Erich Kaltofen, at ISSAC in Boston, MA, on June 28, 2013 to have been awarded to **Professor William Arthur Stein of the Sage Project at the University of Washington**. The Prize includes a Plaque and has a monetary award of $1,000.

William Stein is the creator of the Sage platform (`www.sagemath.org`), which, based on the Python programming language, makes available for integrated use a number of computer algebra programs such as Magma, Maple, Mathematica, MuPAD, and which includes Axiom, GAP, GP/PARI, LinBox, Macaulay2, Maxima, Octave, and Singular, among others. Sage also provides its own programming language Cython.

William Stein has been able to attract hundreds of followers to Sage, who have contributed to the open source base of Sage and who have already met for dozens of Sage Days on several continents. Some of the algorithms in Sage constitute the fastest implementations for the given problems.

Sage's free software philosophy has attracted thousands of users all over the world to undertake computations for mathematical research and development.



Mark Giesbrecht ACM Distinguished Scientist

On November 20, 2013, ACM has selected distinguished members in recognition of their individual achievements and contributions. ACM Sigsam is proud to announce that our colleague **Mark Giesbrecht** from the University of Waterloo has been selected as Distinguished Scientist.

Quoting from the corresponding ACM press release:

> ACM (the Association for Computing Machinery) has named 40 Distinguished Members for their individual contributions and their singular impacts on the vital field of computing. Their achievements have advanced the science, engineering, and education of computing, and highlight the widening role that computing plays in a range of disciplines and domains around the globe. The 2013 Distinguished Members hail from universities in Denmark, Japan, Israel, Italy, China, and the United Kingdom in addition to North America, and from leading international corporations and research institutions.
>
> ACM President Vinton G. Cerf described the recipients as "the problem solvers, prophets, and producers who are powering the future of the digital age." He noted that these ACM members "are the driving force for enabling the computing community to change how we live and work. They demonstrate the advantages of ACM membership, which empowers self-improvement and inspires a bold vision for their own careers as well as their impact on the future."
>
> The ACM Distinguished Member program can recognize the top 10 percent of ACM worldwide membership based on professional experience as well as significant achievements in the computing field.

`http://www.acm.org/press-room/news-releases/2013/distinguished-2013`

ACM Communications in Computer Algebra, Vol. 47, No. 4, Issue 186, December 2013

Abstracts of Recent Doctoral Dissertations in Computer Algebra

Each month we are pleased to present abstracts of recent doctoral dissertations in Computer Algebra and Symbolic Computation. We encourage all recent Ph.D. graduates (and their supervisors), who have defended in the past two years, to submit their abstracts for publication in CCA.
Please send abstracts to the CCA editors `<editors_SIGSAM@acm.org>` *for consideration.*

Author: Nan Li
Title: On Isolated Singular Solutions in Polynomial System Solving
Institution: Chinese Academy of Sciences, Beijing, China
Thesis Advisor: Lihong Zhi
Defended: June 2013

Nowadays, polynomial models are ubiquitous and widely applied across the engineering and sciences, such as in robotics, coding theory, optimization, mathematical biology, computer vision, game theory, statistics, machine learning, control theory, cryptography, and numerous others. A main challenge in algebra and geometry computing is to identify and tackle singular points, which naturally occur when computing the topology of implicit curves or surfaces, the intersection of parametric surfaces in geometric modeling.

A numerical approximation is usually computed to identify an isolated solution of a polynomial system. In practice, we often need to improve the quality of numerical approximations, but numerical methods such like Newton's method converge slowly at singular solutions (or not converge). On the other hand, it is well known that to certify whether a polynomial system has an isolated singular solution is an ill-posed problem, since arbitrary small perturbations of coefficients may transform the singular solution into a cluster of simple roots (or even make it disappear). Therefore, it is hardly possible to verify this problem, if not the entire computation is performed without any rounding error (exact arithmetics).

In this thesis, we first introduce the local dual space for characterizing an isolated singular solution of a polynomial system. By employing some regularization and reduction techniques, we present a novel algorithm for computing a reduced basis of such a space for the special case of breadth one. The algorithm also works for inputs only with limited accuracy, and is efficient both in time and memory use. Moreover, it leads to a parametric representation for a reduce basis (multiplicity structure) of the local dual space.

Based on such a parametric representation and presolving a regularized least squares problem, we propose a regularized Newton's method for refining an approximate singular solution of a given polynomial system. By a careful analysis, we prove the quadratic convergence of the algorithm if the numerical approximation is close to a breadth-one isolated singular solution.

By introducing some well-chosen smoothing parameters to the given system, we develop an improved deflation technique, which derives a square and regular augmented system from an isolated singular solution in a finite number of deflations. Based on this technique, we propose an algorithm for computing verified error bounds such that a slightly perturbed polynomial system is guaranteed to possess an isolated singular solution within the computed bounds.

Author: Maximilian Jaroschek
Title: Removable Singularities of Ore Operators
School: RISC, Johannes Kepler University, Linz
Thesis Advisor: Manuel Kauers
Defended: December 2013

Ore algebras are an algebraic structure used to model many different kinds of functional equations like differential and recurrence equations. The elements of an Ore algebra are polynomials for which the multiplication is defined to be usually non-commutative. As a consequence, Gauß' lemma does not hold in many Ore polynomial rings and hence the product of two primitive Ore polynomials is not necessarily primitive. This observation leads to the distinction of non-removable and removable factors and to the study of desingularizing operators.

Desingularization is the problem of finding a left multiple of a given Ore operator in which some factor of the leading coefficient of the original operator is removed. We derive a normal form for such left factors and unify known results for differential and shift operators into one desingularization algorithm. Furthermore, we analyze the effect of removable and non-removable factors on computations with Ore operators.

The set of operators of an Ore algebra that give zero when applied to a given function forms a left ideal. The cost of computing an element of this ideal depends on the size of the coefficients (the degree) and the order of the operator. In order to be able to predict or reduce these costs, we derive an order-degree curve. For a given Ore operator, this is a curve in the (r, d)-plane such that for all points (r, d) above this curve, there exists a left multiple of order r and degree d of the given operator. We show how desingularization yields order-degree curves which are extremely accurate in examples. When computed for the generator of an operator ideal from applications like physics or combinatorics, the resulting bound is usually sharp.

The generator of a left ideal in an Ore polynomial ring is the greatest common right divisor of the ideal elements, which can be computed by the Euclidean algorithm. Polynomial remainder sequences contain the intermediate results of the Euclidean algorithm when applied to (non-)commutative polynomials. The running time of the algorithm is dependent on the size of the coefficients of the remainders. Different methods have been studied to make these as small as possible. The subresultant sequence of two polynomials is a polynomial remainder sequence in which the size of the coefficients is optimal in the generic case, but when taking the input from applications, the coefficients are often larger than necessary. We generalize two improvements of the subresultant sequence to Ore polynomials, in which we show that the non-removable factors of the greatest common right divisor appear as content. Based on this result we show how to divide out this content during the Euclidean algorithm and derive a new bound for the minimal coefficient size of the remainders. Our approach also yields a new proof for the results in the commutative case, providing a new point of view on the origin of the extraneous factors of the coefficients.

Recent and Upcoming Events

March 31–April 2, 2014
Functional Equations in Limoges (FELIM'14)
Limoges, France
Organizers: Moulay Barkatou, Thomas Cluzeau, and Jacques-Arthur Weil

---

May 15–17, 2014
6. Tagung Fachgruppe Computeralgebra
Kassel, Germany
Organizers: Wolfram Koepf (local organization)
Dates: Registration with talk: March 15, 2014; Registration without talk: May 1, 2014
Website: `http://www.fachgruppe-computeralgebra.de/tagung-kassel-2014/`

May 21–22, 2014
17th Workshop on Computer Algebra
Dubna, Russia
Organizers: Sergei Abramov, Vladimir Gerdt, Alla Bogolubskaya
Dates: Submission: May 11, 2014 (for visa arrangements: March 20, 2014)
Website: `http://compalg.jinr.ru/Dubna2014/index.html`

June 18–20, 2014
XIV Encuentro de Algebra Computacional y Aplicaciones (EACA 2014)
Barcelona, Spain
Dates: Submission: February 21, 2014; Notification: April 15, 2014; Final Version: May 10, 2014
Website: `http://www.ub.edu/eaca2014`

June 23–25, 2014
9th GASCom conference on random generation of combinatorial structures
Bertinoro, Italy
Organizers: Marilena Barnabei (scientific chair), Flavio Bonetti (organization chair)
Dates: Submission: February 28, 2014; Notification: April 11, 2014; Final version: May 9, 2014.
Website: `http://gascom2014.dm.unibo.it/`

July 2–4, 2014
8th International Workshop on Parallel Matrix Algorithms and Applications (PMAA'14)
Lugano, Switzerland
Organizers: Peter Arbenz, Ahmed Sameh, Rolf Krause, Olaf Schenk
Dates: Session proposals: March 15, 2014; Talk submission: March 30, 2014; Notification: April 6, 2014
Website: `http://pmaa14.ics.usi.ch/`

July 7–11, 2014
Conferences on Intelligent Computer Mathematics
Coimbra, Portugal

Organizers:	Stephen Watt (general chair)
Dates:	Workshop proposals: January 17, 2014; Conference submissions: February 28 (abstracts) and March 7 (full papers)
Website:	`http://cicm-conference.org/2014/cicm.php`

July 23–25, 2014
39th International Symposium on Symbolic and Algebraic Computation (ISSAC'14)
Kobe, Japan

Organizers:	Kosaku Nagasaka and Franz Winkler (general chairs), Agnes Szanto (PC chair)
Dates:	Submission: January 12, 2014 (abstracts) and January 19, 2014 (full papers), Notification: March 30, 2014, Final Version: April 30, 2014
Website:	`http://www.issac-symposium.org/2014`

July 28–31, 2014
Symbolic Numeric Computation (SNC 2014)
Shanghai, China

Organizers:	Lihong Zhi (general chair), Stephen Watt (PC chair), Zhengfeng Yang (Local chair)
Dates:	Submission: March 24, 2014; Notification: April 28, 2014; Final version: May 19, 2014
Website:	`http://symbolic-numeric-computation.org/snc-2014/`

August 5–9, 2014
The 4th International Congress on Mathematical Software (ICMS)
Seoul, Korea

Organizers:	Chee K. Yap (general chair), Hoon Hong (program chair), Deok-Soo Kim (local chair)
Dates:	Session proposals: Jan 31 2014
Website:	`http://voronoi.hanyang.ac.kr/icms2014/`

September 8–12, 2014
16th Computer Algebra in Scientific Computing (CASC 2014)
Warsaw, Poland

Organizers:	Vladimir P. Gerdt and Werner M. Seiler (General Chairs); Wolfram Koepf and Evgenii V. Vorozhtsov (PC Chairs)
Dates:	Submission: April 06, 2014; Notification: May 25, 2014; Final version: June 08, 2014
Website:	`http://www14.in.tum.de/CASC2014/`

Call for Papers

39th International Symposium on Symbolic and Algebraic Computation (ISSAC 2014)

http://www.issac-symposium.org/2014/
July 23–25 2014, Kobe, Japan

The International Symposium on Symbolic and Algebraic Computation is the premier conference for research in symbolic computation and computer algebra. ISSAC 2014 is the 39th meeting in the series. The conference traditionally presents a range of invited speakers, tutorials, poster sessions and software demonstrations with a centre-piece of contributed research papers.

ISSAC 2014 is held July 23-25, 2014 at Kobe University, Japan. ISSAC 2014 is affiliated with "Kobe Computing Week 2014", an event of Academic Exchange Weeks, Graduate School of Human Development and Environment, Kobe University.

ISSAC 2014 is one of satellite conferences of ICM 2014 (International Congress of Mathematicians), Korea. Also, SNC 2014 (Symbolic-Numeric Computation), Shanghai, China, is a satellite conference of ISSAC 2014.

Important Dates
Event:

- Workshops and tutorials: July 21–22, 2014
- ISSAC 2014 conference: July 23–25, 2014

Regular papers:

- Paper abstract submission: January 12, 2014
- Full paper submission deadline: January 19, 2014
- Notification of acceptance/rejection: March 30, 2014
- Final version due: April 30, 2014

Posters and Software presentations:

- Abstract submission: April 20, 2014
- Notification: May 16, 2014
- Final version: June 6, 2014

Conference Topics:

ISSAC 2014 invites the submission of original research contributions to be considered for publication and presentation at the conference. All areas of computer algebra and symbolic mathematical computation are of interest. These include, but are not limited to:
Algorithmic aspects:

- Exact and symbolic linear, polynomial and differential algebra
- Symbolic-numeric, homotopy, perturbation and series methods
- Computational algebraic geometry, group theory and number theory

- Computer arithmetic
- Summation, recurrence equations, integration, solution of ODEs & PDEs
- Symbolic methods in other areas of pure and applied mathematics
- Complexity of algebraic algorithms and algebraic complexity

Software aspects:

- Design of symbolic computation packages and systems
- Language design and type systems for symbolic computation
- Data representation
- Considerations for modern hardware
- Algorithm implementation and performance tuning
- Mathematical user interfaces

Application aspects:

Applications that stretch the current limits of computer algebra algorithms or systems, use computer algebra in new areas or new ways, or apply it in situations with broad impact.

Invited Speakers:

Nokiko Arai, David Stoutemyer, and Bernd Sturmfels

Organizers:

General Chairs: Kosaku Nagasaka and Franz Winkler
PC Chair: Agnes Szanto
Local Chair: Kosaku Nagasaka
Publicity: Ekaterina Shemyakova
Treasurer: Akira Terui
Poster Chair: Wen-shin Lee
Software Presentations Chair: Daniel Lichtblau
Tutorial Chair: Tetsu Yamaguchi
Workshop Chair: Takuya Kitamoto
Webmaster: Masaru Sanuki
Program Committee: Shaoshi Chen (Chinese Academy of Sciences, China), Carlos D'Andrea (U. Barcelona, Spain), Wayne Eberly (U. Calgary, Canada), Ioannis Emiris (U. Athens, Greece), Jean-Charles Faugere (INRIA, France), Mark Giesbrecht (U. Waterloo, Canada), Jonathan Hauenstein (North Carolina State University, USA), Evelyne Hubert (INRIA, France), Alexander Hulpke (Colorado State University, USA), Gabor Ivanyos (MTA SZTAKI, Hungary), Joseph Maurice Rojas (Texas A&M University, USA), Julio Rubio (Universidad de La Rioja, Spain), Mohab Safey el Din (Univ. Pierre and Marie Curie, France), Tateaki Sasaki (University of Tsukuba, Japan), Yosuke Sato (Tokyo U. of Science, Japan), Josef Schicho (RISC, Austria), Michael Singer (North Carolina State University, USA), Elena Smirnova (Texas Instruments, USA), Agnes Szanto (North Carolina State University, USA), Chee Yap (NYU, USA)

Call for Session Proposals

4th International Congress on Mathematical Software (ICMS 2014)

http://voronoi.hanyang.ac.kr/icms2014
August 5–9 2014, Seoul Korea
Satellite Conference of ICM 2014
http://www.icm2014.org

The 4th International Congress on Mathematical Software will consist of several topical sessions. Each session will provide an overview of the challenges, achievements and progress in a subfield of mathematical software research, development and use. The program committee will consist of the session organizers. We solicit session proposals.

You are invited to propose a session if you

are active in mathematical software research, development and use, want to serve the research community by nurturing and facilitating mathematical software work in your area, and would like to focus only on the scientific matters in the organization (not on other matters such as administrative, logistic, etc).

How to propose a session

- Prepare a session proposal with the following contents.
  - title of the session
  - name(s) of the organizer(s), with contact addresses and emails
  - aim and scope of the session (at most 150 words)
- Submit it
  - to the program chair (Hoon Hong)
  - by email hong@ncsu.edu
  - at latest by Jan 31 2014.
- The decision on the proposal will be made by the program chair, the general chair and the advisory committee within a week of the submission.

How to organize a session

- Maintain a session web page (a template will be provided).
- Send a call for abstracts to the potential speakers in the topic area of the session (a template will be provided).
- Review the submitted abstracts and make decision on their acceptance, as soon as each one arrives.

- Complete the process by May 15 2014.
- During the meeting, chair the session.

Format of a session

- A session will consist of one or more time slots.
- A time slot will consist of about 5 talks.
- Each talk will last about 30 minutes (including Q/A).
- We encourage that each session begins with one general overview talk (may be given by a session organizer).

Topics for sessions

Any mathematical software topics are welcome. For suggestions, see "Call for Session Proposals" at `http://voronoi.hanyang.ac.kr/icms2014`