ALG48

An algebra library for the HP48

Version 4.0

Claude-Nicolas Fiechter
Mika Heiskanen

Contents

1 Acknowledgments, Copyright & Disclaimer of Warranty

Special thanks to Dominique Rodriguez for his LaTeX version of the documentation for `ALG48` v2.1, on which this document is based, and to Joe Horn for his many useful comments, suggestions and detailed bug reports.

2 Overview

`ALG48` is a comprehensive symbolic math package for the `HP48`. It includes commands for algebraic simplification, factorization, partial fraction expansion, symbolic integration, symbolic matrices manipulation, and for solving systems of nonlinear polynomial equations.

`ALG48` differs from other math packages for the `HP48` in two important aspects:

1. `ALG48` can manipulate, simplify, and factorize *multivariate* polynomials and functions, i.e., algebraic expressions with several variables.
2. `ALG48` only does *exact* calculation, using unlimited precision integers and advanced computer algebra algorithms (as opposed to doing approximate calculation using floating point numbers and numerical algorithms). This not only means you will not get wrong or approximate results (2.00001, 4.9999 and the like), but also that all and only exact simplifications are performed.

Below are some examples of `ALG48` operations. The time taken for the commands on a `HP48GX` (with 60K free) are given in brackets.

- Simplification of multivariate polynomials and rational functions:

$$\frac{1-\dfrac{1-\dfrac{y}{x+y}}{1-\dfrac{x}{x+y}}}{y^2-\dfrac{x}{1+\dfrac{x}{y-x}}\left(\dfrac{xy}{y-x}-x\right)} \quad \texttt{RSIM}\,[1.8\mathrm{s}] \Rightarrow \ \mathrm{inv}(x^2+xy+y^2)$$

- including polynomials and rational functions with non-rational subexpressions:

$$\frac{\cos(a)\sin(a)-\cos(a)-\sin(a)+1}{\cos(a)\sin(a)+\cos(a)-\sin(a)-1} \quad \texttt{RSIM}\,[0.9\text{s}] \Rightarrow \ \frac{\sin(a)-1}{\sin(a)+1}$$

- Complete factorization of polynomials and rational functions:

$$75x^9-435x^8+852x^7-576x^6-663x^5+3027x^4-4911x^3+3402x^2-735x$$

$$\texttt{FCTR}\,[4.9\text{s}] \Rightarrow$$

$$3x\cdot(x^2-3x+1)\cdot(x^4+x-5)\cdot(5x-7)^2$$

- including polynomials and rational functions in several variables:

$$\begin{gathered}3x^5y+9x^4y^2-3x^3y^2+21x^3y-2x^3+x^2y^2-6x^2y+5x^2\\ +3xy^3+17xy-14x-y^3+7y^2-5y+35\end{gathered}$$

$$\texttt{FCTR}\,[11.2\text{s}] \Rightarrow$$

$$(x^2+3xy-y+7)\cdot(3x^3y-2x+y^2+5)$$

- Simplification of non-rational expressions:

$$\frac{\sqrt{x^3+x^2-x-1}}{\sqrt{12\sqrt{5}+49}\cdot(x+1)} \quad \texttt{ASIM}\,[4.3\text{s}] \Rightarrow \ \frac{\sqrt{x-1}}{3\sqrt{5}+2}$$

- including exponential functions:

$$\frac{x\exp(3\ln(x)+\ln(x^2))-1}{\ln\left(\sqrt{\exp(x^2-1)}\right)} \quad \texttt{ASIM}\,[1.8\text{s}] \Rightarrow \ 2x^4+2x^2+2$$

- and trigonometric functions:

$$\frac{\cos(\operatorname{asin}(\sin(x)-\cos(x)))^2}{\ln(\sin(x)\cos(x)\tan(x))} \quad \texttt{ASIM}\,[8.6\text{s}] \Rightarrow \ \frac{\sin(x)\cos(x)}{\ln(\sin(x))}$$

- Partial fraction expansion along one or several variables:

$$\frac{x^3y^2-3x^3y+3x^3-x^2y^2+2x^2y-3x^2-xy^4+5xy^3-8xy^2+5xy-y^3+3y^2-2y}{x^2y^2-3x^2y+2x^2-xy^3+2xy^2+xy-2x+y^3-3y^2+2y}$$

$$\texttt{PF}\,[8.3\text{s}] \Rightarrow$$

$$y+x+\frac{x}{y-2}-\frac{x}{y-1}+\frac{y}{x-y}+\frac{y}{x-1}$$

- Rational function integration:

$$\frac{27x^7-42x^6-106x^5-47x^4+224x^3-147x^2+313x+138}{15x^8-15x^7+15x^6-60x^5+90x^4-105x^3+45x^2+60x-45}$$

$$x \quad \texttt{RINT}\,[8.8\text{s}] \Rightarrow$$

$$\frac{9x-11}{3x^2-6x+3}+\sqrt{2}\cdot\operatorname{atan}\left(\frac{x+1}{\sqrt{2}}\right)+\frac{3}{5}\ln(x^3+x+1)$$

- Symbolic vector and matrix operations:
  - Inverse

$$\begin{pmatrix} 3 & 2x^2 & 1 \\ 4x & 2x^3 & 2x \\ 2x^2 & -1 & 2x^2 \end{pmatrix} \quad \texttt{AINV}\,[5.5\mathrm{s}] \Rightarrow \begin{pmatrix} 2x^4+1 & (-4x^4-1)/(2x) & x^2 \\ -2x^2 & 2x & -1 \\ -2x^4-2 & (4x^4+3)/(2x) & -x^2 \end{pmatrix}$$

  - Determinant

$$\begin{pmatrix} 1 & t & t & t \\ 1 & k & t & t \\ 1 & t & k & t \\ 1 & t & t & k \end{pmatrix} \quad \texttt{MDET}\,[1.6\mathrm{s}] \Rightarrow\ k^3 - 3k^2t + 3kt^2 - t^3 \quad \texttt{FCTR}\,[1.8\mathrm{s}] \Rightarrow\ (k-t)^3$$

  - Addition, Subtraction, Negation, Multiplication, Division, Exponentiation, Transpose, ...
- Solution of systems of linear equations ($Ax = b$) with symbolic coefficients

$$b : \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \quad A : \begin{pmatrix} 1-t & 2 & -4 \\ 3/2-t & 3 & -5 \\ 5/2+t & 5 & -7 \end{pmatrix} \quad \texttt{ADIV}\,[3.3\mathrm{s}] \Rightarrow\ x : \begin{pmatrix} \mathrm{inv}(-2t) \\ (7t+1)/(4t) \\ 3/4 \end{pmatrix}$$

- Solution of systems of nonlinear polynomial equations using Gröbner bases

$$\left.\begin{array}{r} 2x^2 + xy - y + 1 = 0 \\ -3xy - x + 2y^2 - 2 = 0 \\ 2x^2 - 3xy^2 + 2y^3 - 3y + 1 = 0 \end{array}\right\} \quad \{\texttt{x y}\}\ \texttt{GSOLVE}\,[4.9\mathrm{s}] \Rightarrow \left\{\begin{array}{l} x \\ y-1 \end{array}\right., \left\{\begin{array}{l} 2x-6y-5 \\ 14y^2+21y+9 \end{array}\right.$$

Additional features include:

- Provides arithmetic operations on unlimited-size integer numbers, including modular arithmetic, integer factorization, and primality testing.
- Can be used to easily calculate with fractions.
- Can perform algebraic operations over finite fields (modular polynomials).
- Can handle polynomials and rational functions of arbitrary degrees and with arbitrary many variables (limited only by your `HP48`'s memory and by your patience).
- Entirely written in system-RPL and machine language.

3 Installation

3.1 Platforms and ports

`ALG48` version 4.0 takes approximately 49Kb of memory and should work in *any* port of a `HP48GX` or `SX`. Because of its size `ALG48` version 4.0 cannot be use on a `HP48G`. The library was developed on a HP48GX version P and tested in ports 0, 1 and 2. Previous versions were reported to also work properly on the HP48SX.

ALG48 can safely be run from a covered port (2-33) of a HP48GX and uses a special technique to avoid the usual slowdown associated with running a library from a covered port. The downside of this special technique, however, is that ALG48 *cannot be run from a covered port if the RAM card in Card Slot 1 is merged with user memory.* If you try, rather than risking to crash your calculator, ALG48 will generate a "Missing Library" error. If you want to keep your card 1 merged you have to put ALG48 in port 0.

Note: If you have ALG48 in a covered port and want to use the special functions library SpecFun or any other library that uses ALG48 internal routines, you must put that library in the same port as ALG48 or in port 0. Also, if you are running the library from a covered port, only use the provided user commands and do not try to access internal routines of the library directly, since this might crash your calculator.

3.2 Installation procedure

ALG48 is a regular auto-attaching library (library number 909). To install it on your HP48 download the file alg48.lib onto your calculator (in *binary* mode), put the content of the created variable on the stack, store it in the port of your choice (e.g., 'ALG48.LIB' DUP RCL SWAP PURGE 0 STO), and power-cycle the calculator.

The basic operation commands defined in ALG48 (AADD, ASUB, ANEG, AMUL, ADIV, AINV, APOW, see Section 4) are intended to be assigned to the corresponding keys (+ - +/- * / 1/x ^) of the calculator. To do this you can type:

```
{S APOW 45.1 AINV 46.1 ANEG 52.1 ADIV 65.1 AMUL 75.1 ASUB 85.1 AADD 95.1} STOKEYS
```

Thereafter, whenever the calculator is in user mode, pressing one of these keys will call the corresponding ALG48 command, and, if the arguments provided do not match those handled by the library (see the Command Reference in appendix) the standard command will be called. Therefore you can stay in user mode and still perform "regular" operations. In addition, these commands have algebraic aliases, which means that you can stay in user mode when typing a symbolic expression (in what HP calls algebraic-entry mode) and still get the usual symbols (+ - +/- * / 1/x ^) when pressing the corresponding keys. Note, however, that in program-entry mode you will get the ALG48 command names (APOW, AINV, etc.), and that ANEG will not behave exactly like the regular +/- operation. The program AKEYS, distributed with the library, performs a more sophisticated key assignment that solves these problems (see the file AKEYS.TXT for a detailed explanation).

4 Commands

4.1 Generalities

To ensure exact results ALG48 *only works with integer and rational numbers* and produces a "Bad Argument Type" error if it finds a fractional number in the input. If the expressions you have contain fractional numbers, you must convert them first into rational numbers by using the command ->Q or ->Qpi of the HP48, or by using the program ->QpiRac © by A. Coulom. An advantage of ->QpiRac is that it will also convert real arrays into symbolic matrices and complex numbers in (a, b) form into the $a + bi$ form, appropriate for `ALG48`.

The commands in `ALG48` can be divided into eight groups, according to the kind of operations they perform:

1. **Simplification commands:** `RSIM FCTR ASIM RORD RAT->`

   They are used to simplify symbolic expressions or all the elements of a symbolic matrix or vector.

2. **Basic operations commands:** `AADD ASUB ANEG AMUL ADIV AINV APOW`

   They are used to do basic calculation `(+ - +/- * / 1/x ^)` on several kind of objects:

   - Symbolic matrices and vectors;
   - Symbolic expressions;
   - Fractions;
   - Modular polynomials;
   - Unlimited precision integers.

3. **Gröbner bases commands** `GBASIS GSIMP GSOLVE`

   They are used to solve systems of nonlinear polynomial equations.

4. **Symbolic matrix commands:** `MDET MLU MTRN MIDN`

   Perform specific operations on symbolic matrices.

5. **Calculus commands:** `PF RINT`

   Perform partial fraction expansion and integration on symbolic rational functions.

6. **Algebraic commands:** `GCD LCM`

   Perform specific operations on polynomials or unlimited precision integers.

7. **Modular arithmetic:** `MOD+ MOD- MOD* MOD/ MODPOW MODINV`

   Perform modular arithmetic operations on unlimited precision integers.

8. **Prime number operations:** `PRIM? PRIM+ PRIM-`

   Perform operations related to prime numbers on unlimited precision integers.

We describe below how to use these groups of commands to manipulate different kinds of objects. The Command Reference in appendix gives a brief definition and the stack diagram of each command.

4.2 Algebraic expressions simplification

`ALG48` provides two powerful commands for simplifying multivariate polynomials and rational functions. These commands will work on any algebraic expressions by treating them as the quotient of two polynomials in several "variables", which can actually be non-rational subexpressions (see the second example in Section 2).

RSIM – Simplifies a symbolic expression as a rational function and returns it in (expanded) canonical form;

FCTR – Simplifies a symbolic expression as a rational function and factorizes it into a product of irreducible factors.

The simplification consists of two main steps

1. Multiplying out all products of polynomials and collecting the terms of same degree for the numerator and denominator polynomials;
2. Simplification of the rational function by the multivariate polynomial greatest common divisor (GCD) of the the numerator and denominator.

Depending on the type of the simplified rational function, RSIM returns it in one of the following forms:

- If the denominator of the rational function is a constant, then the result is returned as a polynomial with possibly rational coefficients;
- If the numerator of the rational function is a constant, then the result is returned as the inverse of a polynomial with possibly rational coefficients;
- If both the numerator and the denominator of the rational function contain variables, then the result is returned as the ratio of two polynomial with integer coefficients.

Similarly, FCTR returns the simplified rational function as either

- The product of its factors (which are all polynomials with integer coefficients) multiplied by a possibly rational coefficient;
- The inverse of such a product;
- The ratio of two such products.

FCTR computes the true factorization of a polynomial over the integers (or, equivalently, over the field of rational numbers), and not approximate roots over the real or complex field as computed by a root finding program (like the command ROOT or PROOT of the HP48). Polynomials of arbitrary degree can be irreducible over the integer, and a factorization might therefore entail high degree polynomials. For instance:

$$x^{20} - 2x^{15} + 2x^{10} - x^5 - 2 \quad \texttt{FCTR}\,[3.6\text{s}] \Rightarrow \ (x^{10} - x^5 - 1) \cdot (x^{10} - x^5 + 2)$$

ALG48 version 4.0 uses Berlekamp p-adic factorization algorithm and can compute the complete factorization of virtually any polynomial (up to degree 256). For more on RSIM and FCTR performances see Section 4.16.

ALG48 also provides the command RAT->, which operates like RSIM, but returns the numerator and denominator of the simplified rational function as two separated polynomials. In addition, the commands GCD and LCM respectively compute the greatest common divisor and the least common multiple of two polynomials. These two functions do *not* accept rational functions as input, since the GCD and LCM are not well-defined notions in this case.

All the simplification commands produce a "Infinite Result" error if the denominator of the simplified expression is zero.

4.3 Output format for polynomials

All the ALG48 commands output polynomials in expanded canonical form. In the polynomials the terms are arranged into descending order of their degrees, and the "variables" in the terms are arranged in lexicographic order, with the true variables (global and local names) first, followed by the non-rational subexpressions. E.g.,

$$'ax^2y^3\exp(x)\exp(y) + xy^2 - 2\exp(x) + 3'.$$

Sometimes, however, this output format may not be the most appropriate. In some cases a different order for the variables or a "recursive" format, where one or several variables are considered "main" variables and the others are treated as coefficient, may be preferable. The command RORD in ALG48 let you simplify and "re-order" polynomials in such ways.

The command RORD takes two arguments, viz., the symbolic expression to simplify and a main variable or list of main variables. The output polynomial will be expanded with respect to the main variable(s), while the remaining variables will be treated as coefficients. In addition, the order of the variables in the list will be used in the output. The following examples illustrate different possible outputs of RORD on a particular polynomial.

$2ax^2 - ay + bx^2y + bx^2 + x^2y + 3y$

x RORD $\Rightarrow$	$(2a + by + b + y)x^2 - (ay - 3y)$
{x y} RORD $\Rightarrow$	$(b + 1)x^2y + (2a + b)x^2 - (a - 3)y$
{y x} RORD $\Rightarrow$	$(b + 1)yx^2 - (a - 3)y + (2a + b)x^2$
{x y a b} RORD $\Rightarrow$	$x^2yb + x^2y + 2x^2a + x^2b - ya + 3y$

4.4 General algebraic expressions simplification

RSIM and FCTR leave any non-rational (sub)expressions unchanged and treat i (the complex unit) like any other variable. To simplify non-rational algebraic expressions (like square- and yth-root, exponentials, logarithmic, trigonometric and hyperbolic functions) and expressions that involve complex arguments ALG48 provides the command ASIM.

Unlike the simplification of rational expressions, the simplification of general algebraic expressions is somewhat subjective and heuristic in nature. No algorithm will do it optimally in all cases. ASIM does the following:

- Recursively applies RSIM to every "rational" subexpression
- Simultaneously apply rules to simplify non-rational expressions
- Expand the exponentials, logarithms, etc.
- Simplify the resulting expression using RSIM
- Merge back the remaining exponentials, logarithms, etc.

In addition, ASIM substitutes the exact value of the trigonometric functions for arguments that are integer multiples of π, $\pi/2$, and $\pi/4$. A table of the rules that ASIM uses to simplify non-rational expressions is given in appendix.

ASIM takes the natural "principal solution" approach to simplification, that is, it performs simplification that hold in the "principal" case, but that are not necessarily true in all cases. For instance, ASIM simplifies $\sqrt{x^2}$ into x, even though, strictly speaking, this is valid only when x is positive (the "principal" case).

4.5 Automatic simplification flag

ALG48 uses the user flag number 5 as an automatic simplification flag. When the flag is set the result of any basic operation commands is automatically simplified, as by an application of RSIM. For instance:

$$x - \frac{1}{2} \quad \texttt{<enter>} \quad 2x \quad \texttt{AMUL} \Rightarrow \begin{cases} '2x^2 - x' & \text{when the automatic simplification flag is set} \\ '(x - \frac{1}{2})(2x)' & \text{when the flag is clear} \end{cases}$$

Since the simplification of rational functions can be time consuming (see 4.16), it is sometimes preferable to do a series of operations without simplifying the intermediate results (i.e., with the automatic simplification flag clear) and then to simplify explicitly the final result by using RSIM.

4.6 Partial fraction expansion

The command PF computes the partial fraction expansion of a rational function. By default, if the rational function contains several variables, PF computes the partial fraction expansion along *all* the variables. More precisely, PF first computes the partial fraction expansion along the first variable (in the usual lexicographic order) and then, if there is a term whose denominator does not depend on the first variable, expands it along the second variable, and so on, until all the terms have been expanded as much as possible.

In general, the final result will depend on the order in which the expansion along the different variables is performed. Because of that, PF takes a list of variables as an optional second argument. This list of variable specifies along which variables the expansion should be done and the order in which it should be computed. As an example, consider the rational function given in Section 2,

$$\frac{x^3y^2 - 3x^3y + 3x^3 - x^2y^2 + 2x^2y - 3x^2 - xy^4 + 5xy^3 - 8xy^2 + 5xy - y^3 + 3y^2 - 2y}{x^2y^2 - 3x^2y + 2x^2 - xy^3 + 2xy^2 + xy - 2x + y^3 - 3y^2 + 2y}.$$

Here is the output of PF, first with no optional argument, then with $\{x\}$ and $\{y, x\}$ respectively as optional arguments.

$$\begin{array}{ll} \texttt{PF} \Rightarrow & y + x + \dfrac{x}{y-2} - \dfrac{x}{y-1} + \dfrac{y}{x-y} + \dfrac{y}{x-1} \\ \texttt{\{x\} PF} \Rightarrow & \dfrac{xy^2 - 3xy + 3x + y^3 - 3y^2 + 2y}{y^2 - 3y + 2} + \dfrac{y}{x-y} + \dfrac{y}{x-1} \\ \texttt{\{y x\} PF} \Rightarrow & x - 1 + y + \dfrac{y}{x-1} - \dfrac{x}{y-x} + \dfrac{x}{y-2} - \dfrac{x}{y-1} \end{array}$$

In the first case the expansion was done on x first and then on y; in the second case the expansion was done on x alone; and in the third case the expansion was done on y first and then on x. Note in particular that the first and third outputs are different (though equal, of course), and not merely the same fractions in different orders.

4.7 Rational functions integration

The command `RINT` computes the indefinite integral of rational functions. It takes two arguments: the expression to integrate and the integration variable. If the expression to integrate contains non-rational subexpressions that depend on the integration variable or contains irrational numbers (like $\sqrt{2}$) then `RINT` produces a `"Bad Argument Value"` error. Even though it is not explicit in the output, like any indefinite integral, the integral returned by `RINT` is defined up to an additive constant. That is, the general solution for the indefinite integral is the output of `RINT` plus an arbitrary constant.

In general, the indefinite integral of a rational function will have a rational part and a logarithmic part. The rational part is a "regular" rational function in the integration variable, and the logarithmic part is a sum of logarithms whose arguments are polynomials in the integration variable and whose coefficients are constants. E.g.,

$$\frac{505x^6 - 884x^5 - 2028x^4 + 7965x^3 - 11218x^2 + 8771x - 4119}{28x^7 - 28x^6 - 224x^5 + 812x^4 - 1428x^3 + 1484x^2 - 840x + 196}$$

$$\texttt{x RINT}\,[6.5\mathrm{s}] \Rightarrow$$

$$\frac{15x^2 - 39x + 28}{3x^3 - 9x^2 + 9x - 3} + \frac{19}{4}\ln(x-1) + \frac{31}{7}\ln(x^3 + 3x^2 - 2x + 7).$$

It is always possible to compute the rational part of the integral and `RINT` uses Horowitz's algorithm to compute it quickly without computing the partial fraction expansion of the rational function. For instance,

$$\frac{441x^7 + 780x^6 - 2861x^5 + 4085x^4 + 7695x^3 + 3713x^2 - 43253x + 24500}{9x^6 + 6x^5 - 65x^4 + 20x^3 + 135x^2 - 154x + 49}$$

$$\texttt{x RINT}\,[3.3\mathrm{s}] \Rightarrow$$

$$\frac{441x^6 + 678x^5 - 2412x^4 - 14472x^3 + 18033x^2 + 11256x - 12544}{18x^4 - 12x^3 - 72x^2 + 108x - 42}.$$

On the other hand, the coefficients in the logarithmic part are solutions of potentially high-degree equations and cannot always be represented analytically (in closed-form). `RINT` gives an analytical solution only if the coefficients are solutions of equations of degree two or less, i.e., if the coefficients can be expressed exactly in terms of rational numbers and radicals. Otherwise `RINT` leaves the corresponding part of the integral unsolved. For instance, `RINT` completely solves the following integral since the it can be given explicitely in terms of radicals and fractions,

$$\frac{1}{x^2-2} \quad \texttt{x RINT} \Rightarrow \frac{1}{4}\sqrt{2}\cdot\ln(x-\sqrt{2}) - \frac{1}{4}\sqrt{2}\cdot\ln(x+\sqrt{2}).$$

In the contrary, `RINT` leaves the following integral unsolved because the solution can only be expressed in terms of the roots of an equation of degree three,

$$\frac{1}{x^3+2} \quad \texttt{x RINT} \Rightarrow \operatorname{int}\left(\frac{1}{x^3+2}, x\right).$$

When appropriate, to avoid logarithms with complex arguments and coefficients, `RINT` uses arctangents in the logarithmic part of the integral. E.g.,

$$\frac{1}{x^2+2} \quad \texttt{x RINT} \Rightarrow \frac{1}{2}\sqrt{2}\cdot\operatorname{atan}\left(\frac{x}{\sqrt{2}}\right).$$

Because of the limited speed of the calculator, `RINT` does not use the general Rothstein-Trager method to compute the logarithmic part of the integral and in some cases will fail to give an analytical solution when one exists. For instance, `RINT` fails to solve completely the following integral,

$$\frac{6x^7+7x^6-38x^5-53x^4+40x^3+96x^2-38x-39}{x^8-10x^6-8x^5+23x^4+42x^3+11x^2-10x-5}$$

$$\texttt{x RINT}\,[16.7\mathrm{s}] \Rightarrow$$

$$\frac{1}{10}\sqrt{5}\cdot\ln(x-\sqrt{5})-\frac{1}{10}\sqrt{5}\cdot\ln(x+\sqrt{5})+\mathrm{int}\left(\frac{6x^5+6x^4-8x^3-18x^2+8x+8}{x^6-5x^4-8x^3-2x^2+2x+1},x\right),$$

even though the integral of the last term can be given analytically as

$$(1+\sqrt{3})\ln(x^3-\sqrt{3}x^2-(1+\sqrt{3})x-1)+(1-\sqrt{3})\ln(x^3+\sqrt{3}x^2-(1-\sqrt{3})x-1).$$

Note however that `RINT` never introduces unnecessary algebraic extensions to express the integral and can always solve integral whose logarithmic part only entails logarithms with polynomials of degree two or less, regardless of the degree of the rational function itself.

4.8 Symbolic matrix manipulation

`ALG48` represent $(n \times m)$ symbolic matrices by lists of the form

$$\{\{a_{11}\ldots a_{1m}\}\{a_{21}\ldots a_{2m}\}\ldots\{a_{n1}\ldots a_{nm}\}\}$$

where each element a_{ij} is either a real number, a variable or a symbolic expression. Similarly, symbolic vectors $[(n \times 1)$ matrices$]$ are represented by lists of the form $\{a_1 \ldots a_n\}$.

All the symbolic matrix commands of `ALG48` check that their arguments are valid symbolic matrices and will produce a `"Bad Argument Type"` error otherwise. In addition, the commands that accept non-square matrices as arguments will also accept symbolic vectors and will return symbolic vectors when appropriate.

`ALG48` provides the following symbolic matrix commands [below, "scalar" denotes a real number, a variable or a symbolic expression, and I is the identity matrix]:

- `AADD` – Adds two symbolic matrices or vectors, or, given a square matrix A and a scalar x, computes $A + xI$.
- `ASUB` – Subtracts two symbolic matrices or vectors, or, given a square matrix A and a scalar x, computes $A - xI$ (or $xI - A$).
- `ANEG` – Negates all the elements of a symbolic matrix or vector.
- `AMUL` – Multiplies two symbolic matrices or vectors, or a scalar with a symbolic matrix or vector.
- `ADIV` – Multiplies a symbolic matrix, vector or scalar by the inverse of a square symbolic matrix or scalar; can be used to solve systems of linear equations as shown in Section 2.
- `AINV` – Computes the inverse of a square symbolic matrix.

`APOW` – Raises a square symbolic matrix to an integer power.

`MDET` – Computes the determinant of a square symbolic matrix.

`MLU` – Computes the Crout (LU) decomposition of a square symbolic matrix.

`MTRN` – Transposes a symbolic matrix or vector.

`MIDN` – Given an integer number n returns the $(n \times n)$ identity symbolic matrix.

The Crout LU decomposition computed by the command `MLU` combines the lower triangular matrix L and the upper triangular matrix U in a single square matrix. The command also returns the number of "pivots" (iterations) completed, which is a lower bound on the rank of the matrix. If the matrix is invertible then the number is equal to the dimension of the matrix. Both `AINV` and `ADIV` produce a `"Infinite Result"` error if applied to a non-invertible (singular) matrix.

The result of the basic operation `AADD`, `ASUB`, `AMUL`, and `APOW` is simplified or not depending on whether the automatic simplification flag is set (see 4.5), whereas the result of `ADIV`, `AINV`, `MDET`, and `MLU` is always simplified. In addition, `RSIM`, `FCTR`, and `ASIM` can be used to simplify all the elements of a symbolic matrix or vector.

In general the time taken by the matrix manipulation commands increases quickly with the dimensions of the matrices involved. Specifically, for square $n \times n$ matrices, the time taken by the commands `AINV`, `ADIV`, `AMUL`, `APOW`, `MDET`, and `MLU` is proportional to n^3, and the time taken by the other commands is proportional to n^2. `ALG48` version 4.0 can nevertheless handle relatively large matrices in a reasonable amount of time. For instance, `ALG48` takes only 3.5s to invert exactly the following 6×6 matrix, and about 18s to invert it back.

$$\begin{pmatrix} 1 & 2 & 0 & 4 & 0 & 1 \\ 5 & 0 & 4 & 0 & 6 & 3 \\ 0 & 2 & 5 & 6 & 2 & -1 \\ 0 & -1 & 2 & -1 & -1 & 9 \\ -5 & 3 & 1 & -2 & 8 & 0 \\ 1 & 0 & -2 & 1 & 0 & 3 \end{pmatrix}$$

Note also that the time taken by these commands largely depends on whether the elements of the symbolic matrices are numbers or symbolic expressions, and on the number of variables involved in the symbolic expressions.

4.9 Nonlinear equations and Gröbner bases commands

As mentionned in the previous section, `ADIV` let you easily compute the exact solutions of a system of linear equations. Solving a system of *nonlinear* equations is usually much harder. Even a single univariate equation of degree greater than four cannot in general be explicitely solved in terms of rational numbers and radicals. It is however possible, using a root finder program (like the command `ROOT` or `PROOT` of the `HP48`), to compute good approximate numerical solutions of a nonlinear univariate equation. We can therefore consider that a system of nonlinear equations is solved if we have reduced it into an equivalent form in which the roots can be obtained easily with a root finder program.

Solving a linear system typically involves "eliminating" unknowns from equations to obtain an equivalent triangularized system which is then easy to solve. This process is known as Gaussian elimination. Gröbner bases generalize this approach to solve systems of nonlinear polynomial equations. The Gröbner basis of a system of polynomial equations is a set of equations that has the same solutions as the original system but that is simpler, in a mathematically well-defined way, than the original system.[1] For example, consider the following system of nonlinear equations,

$$\begin{aligned} x^2 + yz &= 2 \\ y^2 + xz &= 3 \\ xy + z^2 &= 5, \end{aligned}$$

which is represented in **ALG48** by a list containing three algebraics. Its Gröbner basis computed by the command **GBASIS** is

$$\begin{gathered} 361x - 88z^7 + 872z^5 - 2690z^3 + 2375z \\ 361y + 8z^7 + 52z^5 - 740z^3 + 1425z \\ 8z^8 - 100z^6 + 438z^4 - 760z^2 + 361, \end{gathered}$$

where the missing right-hand-side of the equations are implicitly understood to be zero. Even though this new system might look at first more complex than the original one, it is actually much easier to solve because it is triangularized. The last equation depends on z alone, the second equation depends only on y and z, and the first equation depends only on x and z. Thus, using a root finder program, you can easily compute the (eight) solutions for z, and then, by backsubstitution, the corresponding solutions for y and x.

It is well known that the existence and number of solutions of a system of linear equations can be neatly characterized in terms of the number of variables and independent equations of the system. There is no such simple characterization for *nonlinear* systems. In general the Gröbner basis for a system of nonlinear equations can have fewer or more equations than the original system. If there are as many equations as there are variables, and if the equations are sorted according to their leading term, the basis will often, but not always, be in triangular form suitable for backsubstitution. Moreover, if the system has *no solution* then the basis will include a constant and will reduce to 1. E.g.,

$$\left.\begin{array}{r} x^2 + 4y^2 - 17 = 0 \\ 2xy - 3y^3 + 8 = 0 \\ xy^2 - 5xy + 1 = 0 \end{array}\right\} \quad \{\texttt{x y}\}\ \texttt{GBASIS} \Rightarrow 1.$$

Even though Gröbner bases are in a sense minimal, they are not unique, and a system of equations can have several different (though equivalent) bases. The basis depends in particular on the order of the terms in the polynomials and on the order in which the variables are "eliminated". **ALG48** always uses a lexicographic ordering for the terms (see Section 4.3) and all the Gröbner commands expect a list of variables on Stack Level 1 that specifies the order of the variables. The list of variables also specifies which variables are main variables (as opposed to coefficients) for the output format, like in the command **RORD**. For instance, in the example below, x, y and z are the main variables, and c is treated as a parameter,

$$\left.\begin{array}{r} cx + (c+1)y + z = 1 \\ x + cy + (c+1)z = 2 \\ (c+1)x + y + cz = -1 \end{array}\right\} \quad \{\texttt{x y z}\}\ \texttt{GBASIS} \Rightarrow \left\{\begin{array}{l} x + cz \\ y - c^2z + 1 \\ (c^3+1)z - (c+2). \end{array}\right.$$

[1] A mathematical definition of Gröbner bases is beyond the scope of this document. Interested readers are referred to, e.g., *Gröbner bases: a computational approach to commutative algebra*, Thomas Becker and Volker Weispfenning, Springer-Verlag, NY, 1993. A more detailed explanation about the Gröbner commands in **ALG48** and a large number of examples can also be found in the "Grobner" document that is distributed with **ALG48**.

A different and much more complicated basis would have been obtained if the regular lexicographic order, with c first, had been used.

Beside `GBASIS`, there are two additional Gröbner commands in `ALG48`: `GSOLVE` and `GSIMP`. The command `GSOLVE` computes the Gröbner basis of a system of polynomial equations and then factors the basis as much as possible to determine independent sets of solutions. Each set of solutions is represented by its own set of equations, and the number of independent sets of solutions is return on Stack Level 1. For instance,

$$\left.\begin{array}{r} 2xy(x+y-1)^3+3x^2y(x+y-1)^2 \\ x^2(x+y-1)^3+3x^2y(x+y-1)^2 \end{array}\right\} \quad \texttt{\{x y\} GSOLVE}$$

returns the real number 3 on Stack Level 1, and the following three systems of equations on Stack Level 2, 3 and 4, respectively:

$$\left\{\begin{array}{l} 3x-1 \\ 6y-1 \end{array}\right. , \quad \{\ x+y-1\ , \quad \{\ x\ .$$

This means that the system has three independent sets of solutions: the point $x = 1/3, y = 1/6$; the line $x = 1 - y$; and the line $x = 0$. Incidentally, the equations in this example are the partial derivatives, with respect to x and y, of the bivariate function $f(x,y) = x^2y(x+y-1)^3$. Hence, the solutions found correspond to the critical points and singularity lines of f.

Section 2 provides an other example of `GSOLVE`. There the system has two independent sets of solution, one corresponding to $x = 0, y = 1$, the other given by the system

$$\left\{\begin{array}{l} 2x-6y-5 \\ 14y^2+21y+9. \end{array}\right.$$

Using for instance the command `QUAD` or `PROOT` of the `HP48` it is easy to determine the two complex solutions corresponding to that latter system: $x = \frac{1}{4}+\frac{9}{28}\sqrt{7}i, y = -\frac{3}{4}+\frac{3}{28}\sqrt{7}i$ and their conjugates.

The command `GSIMP` computes the Gröbner basis for a given system of polynomial equations and then reduces an equation with respect to that system. The equation to reduce is given on Stack Level 3, the system of equations on Level 2, and the list of variables on Level 1. `GSIMP` lets you answer questions of the form *what is the value of this equation, given that these side relations hold.* For instance, consider the following problem from the Dutch Mathematics Olympiad of 1991:

> Let a, b, c be real numbers such that $a+b+c = 3$, $a^2+b^2+c^2 = 9$, and $a^3+b^3+c^3 = 24$. Compute $a^4+b^4+c^4$.

With `GSIMP` you immediatly get the solution.

3: $a^4+b^4+c^4$

2: $\{a+b+c=3$
$a^2+b^2+c^2=9$
$a^3+b^3+c^3=24\}$

1: $\{a,b,c\}$

`GSIMP` [1.7s] $\Rightarrow$ 69.

Note that if we had computed the solutions for a, b, and c using for instance `GBASIS` or `GSOLVE` we would have obtained an irreducible third degree polynomial (with 3 real roots). Hence, computing the actual values for a, b, c and then substituting them back into $a^4+b^4+c^4$ would have involved considerably more work than with `GSIMP`.

4.10 Verbose mode flag

Gröbner bases calculations are complex operations. Even some apparently simple nonlinear systems can lead to extremely complex bases, with high-degree equations and large coefficients. Needless to say, these calculations can be time-consuming, and not all systems can be solved within the memory and speed limits of the `HP48`.

`ALG48` uses the user flag number 1 as a verbose mode flag. When the flag is set the Gröbner commands display some messages on the top three lines of the screen while they execute. The messages describe the operations that the command is currently performing. This allows the user to monitor the progresses the calculator is making toward a solution. If it becomes apparent that a solution cannot be obtained in a reasonable amount of time, the command can be aborted as usual, by pressing the `CANCEL` (`ON`) key.

4.11 Calculating with fractions

`ALG48` can be used to easily calculate with fractions, especially if the basic operation commands are assigned to the corresponding keys of the calculator (see Section 3).

To facilitate the keying of fractions, `ADIV` and `AINV` applied to integer arguments return a symbolic fraction instead of evaluating the result as a real number. Thus, to calculate an expression using fractions just type the expression in regular `RPN`, as you would to evaluate it using real numbers. For instance, to compute $3/4 + 1/6$, you just type:

3 `<enter>` 4 `ADIV`	$\Rightarrow$	$'3/4'$
6 `AINV`	$\Rightarrow$	$'1/6'$
`AADD`	$\Rightarrow$	$'11/12'$

Note: Make sure the automatic simplification flag is set when calculating with fractions, otherwise the expressions will not be evaluated (see 4.5).

4.12 Algebraic operations on modular polynomials

`ALG48` can be used to perform algebraic operations on modular polynomials, i.e., polynomials whose coefficients belong to the finite ring Z_n generated by some positive number n, and all the operations are performed modulo n. Usually, n will be a prime number; in that case Z_n is a finite field.

Modular polynomials are represented in `ALG48` by regular symbolic expressions with a `MOD` operation at the end. E.g.,

```
'(2*X^2+5*X-1) MOD 13'.
```

`ALG48` uses the "symmetric" representation when it outputs modular polynomials, that is, it uses coefficients in the range $-\lfloor n/2 \rfloor \ldots \lfloor n/2 \rfloor$ when the modulo is n.

All the basic operations commands `AADD`, `ASUB`, `ANEG`, `AMUL`, `ADIV`, `AINV`, and `APOW`, as well as the commands `GCD`, `LCM`, and `RSIM` can be used with modular polynomials. For instance,

$$\begin{array}{l} (2x-2) \bmod 5 \\ (3x-2) \bmod 5 \end{array} \quad \texttt{AMUL} \Rightarrow (x^2-1) \bmod 5.$$

When used with modular polynomials ADIV returns both the quotient (on Stack Level 2) and the remainder of the division (on Stack Level 1). E.g.,

$$\begin{array}{l} (x^2+1) \bmod 5 \\ (3x-2) \bmod 5 \end{array} \quad \texttt{ADIV} \Rightarrow \quad \begin{array}{l} (2x-2) \bmod 5 \\ 2 \bmod 5 \end{array} .$$

All these commands generate a "`Bad Argument Type`" error if the expressions are not polynomials, if the moduli are not positive whole numbers, or if the moduli do not have the same value in all the arguments.

4.13 Unlimited precision integer arithmetic

Internally `ALG48` does all its calculations using unlimited precision integers. These unlimited precision integers are represented by hexstrings (binary integers) of variable length (*not* limited to 64 bits), with a sign-magnitude format (one sign nibble and a variable length unsigned magnitude). For instance, the number 1 is represented by the two-nibble hexstring `#01h`, whereas the number 2^{64} is represented by the eighteen-nibble hexstring `#0100..0h`. Negative numbers are identical except for the sign nibble which is set to `F`. For instance, the number -1 is represented by the two-nibble hexstring `#F1h` and -2^{64} is represented by the eighteen-nibble hexstring `#F100..0h`. Finally, zero is represented by the one-nibble hexstring `#0h`. Note that the two-nibble hexstring `#F1h` does *not* represent the same number as the three-nibble hexstring `#0F1h` (which represents the number 241).

You can use `ALG48` to do unlimited precision integer arithmetic directly by using the basic operation commands (except `AINV`) with binary integer arguments (or one binary integer and a real number). For example

```
#2 <enter> 65   APOW
```

computes the exact value of 2^{65}. Note, however, that the `HP48` will only display the value of the 64 (or whatever your binary word size currently is) lowest-significant bits of long hexstrings. Therefore, in the example above, the result will be displayed as `#0h` since the lowest 64 bits are all zeros. To overcome this problem, `ALG48` provides the command `Z<->S` that converts a variable length hexstring into a (character) string giving its value in decimal, or vice versa. Thus typing

`#2 <enter> 65   APOW Z<->S` returns `"36893488147419103232"`

which is the exact value of 2^{65}. As an additional example, the following little user-RPL program computes the exact factorial of its single real argument, and returns it as a string:

```
<< #1h 1 ROT FOR i i AMUL NEXT Z<->S >>
```

Running it with 100 as argument gives

"93326215443944152681699238856266700490715968264381621468592963895217599993229915608941463976156518286253697920827223758251185210916864000000000000000000000000"

As a typing short-cut, `ZS` is an alias name for the `Z<->S` command.

4.14 Advanced algebraic operations on unlimited precision integers

In addition to the basic operation commands, several commands of ALG48 perform advanced algebraic computations on unlimited precision integers.

`GCD` – Greatest common divisor of two integers

`LCM` – Least common multiple of two integers

`PRIM?` – Check whether a number is prime

`PRIM+` – Returns the next (larger) prime number

`PRIM-` – Returns the previous (next smaller) prime number

`FCTR` – Factorization into prime numbers

The integer argument(s) for all these commands, except `FCTR`, can be given (in any combination) as (integer) real numbers, unlimited precision binary integers, or strings representing the number in decimal. Here are some examples of the primality testing commands. The first number below is $2^{127} - 1$, which is the 12th Mersenne number, and which is known to be prime.

`"170141183460469231731687303715884105727"`	`PRIM?` [58s] $\Rightarrow$	1 (= prime)
`"1305293778369724882512685 78591"`	`PRIM?` [8s] $\Rightarrow$	0 (= not prime)

`#1234567890123456d`	`PRIM+` [6s] $\Rightarrow$	`#1234567890123481d`
`#1234567890123456d`	`PRIM-` [4s] $\Rightarrow$	`#1234567890123439d`

Because of its use as a simplification command, `FCTR` leaves real numbers unchanged and will only factor integers given as binary integers or as strings. If the number is given as a binary integer `FCTR` returns a list with the prime factors. If the number is given as a string, the factors are given in a symbolic form. For instance

`#1305293778369724882512685 78591d` `FCTR` [34s] $\Rightarrow$
{ `#2647d #3691d #5113d #11779d #398609d #556517681d` }

and

`"1305293778369724882512685 78591"` `FCTR` [34s] $\Rightarrow$
$'2647 \cdot 3691 \cdot 5113 \cdot 11779 \cdot 398609 \cdot 556517681'$

`FCTR` uses advanced integer factorization algorithms and can factor relatively large numbers. However no "efficient" (polynomial-time) algorithm is known for factoring arbitrarily large integers and it is widely believed that no such algorithm can exist. Thus `FCTR` will factor completely only reasonably small numbers (up to 20-30 digits) or larger numbers that consist only of small factors. To avoid running forever, if `FCTR` does not make any progress after one minute it returns the number unfactored (or only partially factored). As always, you can also abort the computation by pressing the `CANCEL` (`ON`) key.

In the contrary, the primality testing algorithm (used by `PRIM?`, `PRIM+` and `PRIM-`) *can* handle very large numbers. However the algorithm is randomized and might, with very low probability, say that a number is prime when it is not (but will never say that a number is not prime when in fact it is).

4.15 Modular arithmetic on unlimited precision integers

ALG48 provides six commands specifically to perform modular arithmetic on unlimited precision integers. These commands take three arguments (two operands A and B, and a modulus N), except **MODINV** which takes only two arguments (A and N). Here again the arguments can be given as (integer) real numbers, binary integers, or strings.

MOD+ – $(A + B) \bmod N$

MOD- – $(A - B) \bmod N$

MOD* – $(A \cdot B) \bmod N$

MOD/ – $(A \cdot C) \bmod N$, where C is the inverse modulo N of B

MODPOW – $\left(A^B\right) \bmod N$

MODINV – inverse modulo N of A

If A and N are relatively prime numbers (with $A < N$), the inverse modulo N of A is the (unique) number C that satisfies

$$(A \cdot C) \bmod N = 1.$$

If no such inverse exists, i.e., if A and N are not relatively prime, then **MODINV** returns **#0h**. Similarly, **MOD/** returns **#0h** if its second and third arguments are not relatively prime.

4.16 Performances

Algebraic computations, like the simplification, factorization, partial fraction expansion or integration of rational functions, are complex operations and are generally time-consuming for non-trivial problems. Therefore, even though **ALG48** is written in sysRPL and machine language, any operation that involves such operations is not instantaneous on the **HP48**.

ALG48 version 4.0 can nevertheless perform most simplifications quite quickly. Section 2 gives some examples with their timings. [The times given throughout this document were obtained on a **HP48GX** with approximately 60Kb of free memory.] Even complex simplifications can be handled in a reasonable amount of time. For instance, **ALG48** version 4.0 takes only 8s to simplify the relatively large three-variable rational below.

$$\frac{6x^6-126x^4y^3z+78x^4yz^2+x^4y+x^4z+13x^3-21x^2y^4z-21x^2y^3z^2+13x^2y^2z^2+13x^2yz^3-21xy^3z+13xyz^2+2xy+2xz+2}{9x^5+2x^4yz-189x^3y^3z+117x^3yz^2+3x^3-42x^2y^4z^2+26x^2y^2z^3+18x^2-63xy^3z+39xyz^2+4xyz+6}$$

$$\texttt{RSIM}\,[8.1\text{s}] \Rightarrow \frac{6x^3+xy+xz+1}{9x^2+2xyz+3}.$$

ALG48 is also very fast at simplifying polynomials (multiplying out the products and collecting the terms of same degree). For instance, **RSIM** takes only 1s to simplify the following expression:

$$(x+1)^{12} - (x-1)^{12} \quad \texttt{RSIM}\,[1.1\text{s}] \Rightarrow 24x^{11} + 440x^9 + 1584x^7 + 1584x^5 + 440x^3 + 24x.$$

As a comparison, the program **EXCO** (Expand & Collect completely), described in **HP**'s Advanced User's Reference Manual (p.2-20), was not able to obtain the solution in 10 hours. Using extrapolation from the time taken by **EXCO** to perform simpler binomial expansions, John Stebbins estimated that it would take **EXCO** more than 18 days (!) to find the solution of this example.

The factorization of polynomials is comparatively slow, especially for multivariate problems. Simple factorizations, however, are performed quite fast. For instance, `FCTR` takes only slightly more than 2s on the following example:

$$3x^2y + 9x^2 - xy^3 - 5xy^2 - 2xy - 18x + y^4 - y^3 + 6y^2 - y + 5$$

$$\texttt{FCTR}\,[2.2\text{s}] \Rightarrow$$

$$(3x - y^2 + y - 5) \cdot (xy + 3x - y^2 - 1).$$

Even some seemingly complex factorizations can be performed quickly, like the following one, taken from Mathematica's book, that `ALG48` solves in 15s,

$$\begin{gathered}4096x^8-14336x^7y+43008x^7+16768x^6y^2-155904x^6y+169344x^6-5600x^5y^3+195552x^5y^2-635040x^5y+296352x^5\\ -1919x^4y^4-83244x^4y^3+849366x^4y^2-1148364x^4y+194481x^4+700x^3y^5-9744x^3y^4-433944x^3y^3+1629936x^3y^2\\ -777924x^3y+262x^2y^6+8568x^2y^5+15876x^2y^4-963144x^2y^3+1166886x^2y^2+28xy^7+1680xy^6\\ +31752xy^5+148176xy^4-777924xy^3+y^8+84y^7+2646y^6+37044y^5+194481y^4\end{gathered}$$

$$\texttt{FCTR}\,[15.2\text{s}] \Rightarrow\ (x-y)^4(8x+y+21)^4.$$

`ALG48` version 4.0 uses Berlekamp p-adic factorization algorithm and, given enough time, can compute the complete factorization of virtually any polynomials (up to degree 256), even if they are square-free and contain high-degree factors. E.g.,

$$x^{40}+x^{30}-x^{20}-2x^{15}-x^{10}-2x^5-1 \quad \texttt{FCTR}\,[125\text{s}] \Rightarrow\ (x^{20}-x^{15}+x^{10}-x^5-1)\cdot(x^{20}+x^{15}+x^{10}+x^5+1);$$

$$x^{18}-y^{18} \quad \texttt{FCTR}\,[16\text{s}] \Rightarrow\ (x-y)(x+y)(x^2-xy+y^2)(x^2+xy+y^2)(x^6-x^3y^3+y^6)(x^6+x^3y^3+y^6).$$

Even large multivariate polynomials can be handled in a reasonable amount of time. For instance, `ALG48` can factor the numerator and the denominator of the large three-variable rational above in about a minute.

$$6x^6-126x^4y^3z+78x^4yz^2+x^4y+x^4z+13x^3-21x^2y^4z-21x^2y^3z^2+13x^2y^2z^2+13x^2yz^3-21xy^3z+13xyz^2+2xy+2xz+2$$

$$\texttt{FCTR}\,[59\text{s}] \Rightarrow\ (x^3-21xy^3z+13xyz^2+2)\cdot(6x^3+xy+xz+1);$$

$$9x^5+2x^4yz-189x^3y^3z+117x^3yz^2+3x^3-42x^2y^4z^2+26x^2y^2z^3+18x^2-63xy^3z+39xyz^2+4xyz+6$$

$$\texttt{FCTR}\,[65\text{s}] \Rightarrow\ (9x^2+2xyz+3)\cdot(x^3-21xy^3z+13xyz^2+2).$$

As a first approximation, the running time of the simplification and factorization algorithms used in `ALG48` increases exponentially with the number of variables and polynomially (but fast —like n^3) with the maximum total degree n of the polynomials involved. In theory the factorization algorithm can also take exponential time in the degree of the polynomial, but this is usually not the case in practice. However, as illustrated by the examples above, the actual time taken by the simplification and factorization commands varies greatly depending on the properties of the polynomial involved (e.g., whether the polynomial is square-free or not, whether the factors are all linears, etc.).

4.17 Remarks

Here are a few additional things to note about `ALG48`'s commands.

- ASIM is the only command of ALG48 that handles complex arguments directly in their (x, y) form. All other commands will produce a "Bad Argument Type" error if they finds such complex argument in the input. However, complex arguments in the form $'x + yi'$ are handled properly, with i treated as any other variable.

- If they are given equations as input RSIM and ASIM will simplify both side of the equation independently. On the other hand, FCTR does not accept equations as input, and produce a "Bad Argument Type" error in this case.

- Before trying to factor a polynomial or rational function, FCTR blindly simplifies it into canonical form (expands it completely and collects the terms of same degree). Therefore, it is in general a good idea if you have a polynomial already partially factored to split it first using the command OBJ-> of the calculator, and then apply FCTR to each term separately.

- Even though ALG48 internally uses unlimited precision integers for all its computations, the HP48 can only handle real numbers inside symbolics. Thus ALG48 cannot input or output unlimited precision integers from or into algebraic expressions, and will produce a "Bad Argument Value" error if a number in the input is too big to be represented exactly by a real. For instance:

  '1/2' 65 APOW $\Rightarrow$ '1/3.6894..E19' 1 AAAD $\Rightarrow$ Bad Argument Value

  Therefore, if for example you want to do unlimited precision arithmetic with fractions, you have to handle the numerator and denominator separately as two unlimited precision binary integers.

5 History of changes

Changes from version 3.0 to 4.0

- The partial fraction expansion command PF, the rational function integration command RINT, the Gröbner bases commands GBASIS, GSOLVE, and GSIMP, as well as the polynomial re-ordering command RORD, were added.

- FCTR was rewritten to use a better and faster factorization algorithm (Berlekamp's p-adic algorithm).

- The advanced symbolic matrix operations (determinant, inversion, division) have been rewritten to use better algorithms and much faster code. The command MADJ was replaced by the command MLU.

- The basic operation commands can now also handle modular polynomials.

- The command MOD^ was renamed MODPOW.

Changes from version 2.1 to 3.0

- The commands ASIM, MOD+, MOD-, MOD*, MOD/, MOD^, MODINV, PRIM?, PRIM+, and PRIM- were added.

- Both the integer and polynomial factorization algorithms used by `FCTR` have been greatly improved.
- The conversion from/to real numbers to/from binary integers is now done by `ALG48`'s own routine. They do not depend any longer on the binary word size, and handle large numbers correctly.
- Many speed optimizations and minor bug fixes.

Changes from version 1.2 to 2.1

- The command `SQFF` was replaced by the command `FCTR` that computes the complete (and not only square-free) factorization of a polynomials (or of an integer).
- The commands `RAT->`, `GCD`, and `LCM` where added.
- Further code optimizations and new machine language routines have increased the speed on most operations by a factor two to six.
- Algebraic aliases where added for the basic operation commands (see Section 3).
- The library can now be run from a covered port of a `HP48GX` safely and without affecting the performances.
- Fixed bug so that SQ in symbolics would be handled properly.

Changes from version 1.0 to 1.2

- Large part of the library was rewritten to use better internal representations, faster algorithms and new machine language routines. The net result is that version 1.2 is 5 to 10 times faster than version 1.0 for the simplifications and approximately twice as fast on the other operations.
- Errors (Bad arguments, etc...) are now also handled correctly in the symbolic matrix operations (in some cases they would leave garbage on the stack in version 1.0);
- The command `Z2S` to convert a long integer into a string was replaced by the command `Z<->S` that can perform the conversion in both directions.

6 Contact

Gifts `:)`, bug reports, and constructive comments and suggestions can be sent to either one of the following addresses.

Claude-Nicolas Fiechter
Department of Computer Science
University of Pittsburgh
Pittsburgh, PA 15260, U.S.A.
e-mail: `fiechter@cs.pitt.edu`

Mika Heiskanen
Jämeräntaival 7 C 355
02150 Espoo, Finland
e-mail: `mheiskan@delta.hut.fi`

A Simplification Rules for ASIM

The tables below list the rules that `ASIM` uses to simplify and expand non-rational expressions. Converse rules are used to merge back the exponential and trigonometric functions remaining after the simplification. Here x and y stand for any subexpression, n is any integer number and i is the complex unit.

Square Function

expression	simplified
$\mathrm{sq}(\sqrt{x})$	x
$\mathrm{sq}(i)$	-1
$\mathrm{sq}(\cos(x))$	$1-\mathrm{sq}(\sin(x))$
$\mathrm{sq}(\tan(x))$	$\mathrm{sq}(\sin(x)/\cos(x))$
$\mathrm{sq}(\cosh(x))$	$1+\mathrm{sq}(\sinh(x))$
$\mathrm{sq}(\tanh(x))$	$\mathrm{sq}(\sinh(x)/\cosh(x))$
$\mathrm{sq}(x^y)$	$x^{2\cdot y}$

Power Function

expression	simplified	condition
e^x	$\exp(x)$	if x nonreal
$(-x)^n$	x^n	if n even
$(\sqrt{x})^n$	$x^{n/2}$	if n even
i^n	1, i, -1, or $-i$	depending on n
$\exp(x)^y$	$\exp(y\cdot x)$	if y nonreal
$\mathrm{alog}(x)^y$	$\mathrm{alog}(y\cdot x)$	if y nonreal

Square Root

expression	simplified	condition
$\sqrt{x^2}$	x	
$\sqrt{x^n}$	$x^{n/2}$	n even
$\sqrt{x^n}$	$\sqrt{x}\cdot x^{(n-1)/2}$	n odd
$\sqrt{x\cdot y}$	$\sqrt{x}\cdot\sqrt{y}$	
$\sqrt{x/y}$	$\sqrt{x}/\sqrt{y}$	
$\sqrt{-x}$	$\sqrt{x}\cdot i$	
$\sqrt{a\sqrt{x}+b}$	$c+d\sqrt{x}$	for integer solutions

Exponential

expression	simplified	condition
$\exp(0)$	1	
$\exp(1)$	e	
$\exp(-1)$	$\mathrm{inv}(e)$	
$\exp(\pi\cdot i)$	-1	
$\exp(x)$	e^x	if x is real
$\exp(\ln(x))$	x	
$\exp(-x)$	$\mathrm{inv}(\exp(x))$	
$\exp(x+y)$	$\exp(x)\cdot\exp(y)$	
$\exp(x-y)$	$\exp(x)/\exp(y)$	
$\exp(xy)$	$\exp(y)^x$	if x is real

ALOG Function

expression	simplified	condition
$\mathrm{alog}(n)$	integer	
$\mathrm{alog}(\log(x))$	x	
$\mathrm{alog}(-x)$	$\mathrm{inv}(\mathrm{alog}(x))$	
$\mathrm{alog}(x+y)$	$\mathrm{alog}(x)\cdot\mathrm{alog}(y)$	
$\mathrm{alog}(x-y)$	$\mathrm{alog}(x)/\mathrm{alog}(y)$	
$\mathrm{alog}(xy)$	$\mathrm{alog}(y)^x$	if x is real

Natural Logarithm

expression	simplified
$\ln(1)$	0
$\ln(e)$	1
$\ln(-1)$	$\pi\cdot i$
$\ln(i)$	$\pi\cdot i/2$
$\ln(\exp(x))$	x
$\ln(xy)$	$\ln(x)+\ln(y)$
$\ln(x/y)$	$\ln(x)-\ln(y)$
$\ln(x^y)$	$y\cdot\ln(x)$
$\ln(\mathrm{inv}(x))$	$-\ln(x)$
$\ln(\sqrt{x})$	$\ln(x)/2$

Base 10 Logarithm

expression	simplified
$\log(1)$	0
$\log(\mathrm{alog}(x))$	x
$\log(xy)$	$\log(x)+\log(y)$
$\log(x/y)$	$\log(x)-\log(y)$
$\log(x^y)$	$y\cdot\log(x)$
$\log(\mathrm{inv}(x))$	$-\log(x)$
$\log(\sqrt{x})$	$\log(x)/2$

Trigonometric Functions

expression	simplified
$\sin(\mathrm{asin}(x))$	x
$\sin(\mathrm{acos}(x))$	$\sqrt{1-x^2}$
$\sin(\mathrm{atan}(x))$	$x/\sqrt{1+x^2}$
$\sin(-x)$	$-\sin(x)$
$\cos(\mathrm{acos}(x))$	x
$\cos(\mathrm{asin}(x))$	$\sqrt{1-x^2}$
$\cos(\mathrm{atan}(x))$	$1/\sqrt{1+x^2}$
$\cos(-x)$	$\cos(x)$
$\tan(\mathrm{atan}(x))$	x
$\tan(\mathrm{asin}(x))$	$x/\sqrt{1-x^2}$
$\tan(\mathrm{acos}(x))$	$\sqrt{1-x^2}/x$
$\tan(-x)$	$-\tan(x)$
$\mathrm{asin}(0)$	0
$\mathrm{atan}(0)$	0

Hyperbolic Functions

expression	simplified
$\sinh(0)$	0
$\sinh(\mathrm{asinh}(x))$	x
$\sinh(\mathrm{atanh}(x))$	$x/\sqrt{1-x^2}$
$\cosh(0)$	1
$\cosh(\mathrm{acosh}(x))$	x
$\cosh(\mathrm{asinh}(x))$	$\sqrt{1+x^2}$
$\cosh(\mathrm{atanh}(x))$	$1/\sqrt{1-x^2}$
$\tanh(0)$	0
$\tanh(\mathrm{atanh}(x))$	x
$\tanh(\mathrm{asinh}(x))$	$-i \cdot x/\sqrt{1+x^2}$
$\mathrm{asinh}(0)$	0
$\mathrm{atanh}(0)$	0

Division

expression	simplified
$\mathrm{inv}(i)$	$-i$
$\mathrm{inv}(\mathrm{inv}(x))$	x
$x/\sqrt{x}$	$\sqrt{x}$
$\sin(x)/\cos(x)$	$\tan(x)$
$\sin(x)/\tan(x)$	$\cos(x)$
$\cos(x) \cdot \tan(x)$	$\sin(x)$
$\tan(x)/\sin(x)$	$1/\cos(x)$

Absolute Value

expression	simplified
$\mathrm{abs}(\mathrm{abs}(x))$	$\mathrm{abs}(x)$
$\mathrm{abs}(\mathrm{re}(x))$	$\mathrm{re}(x)$
$\mathrm{abs}(\mathrm{im}(x))$	$\mathrm{im}(x)$
$\mathrm{abs}(\pi)$	π
$\mathrm{abs}(e)$	e
$\mathrm{abs}(i)$	1
$\mathrm{abs}(-x)$	$\mathrm{abs}(x)$

Real Part

expression	simplified
$\mathrm{re}(\mathrm{re}(x))$	$\mathrm{re}(x)$
$\mathrm{re}(\mathrm{im}(x))$	$\mathrm{im}(x)$
$\mathrm{re}(\mathrm{conj}(x))$	$\mathrm{re}(x)$
$\mathrm{re}(\mathrm{abs}(x))$	$\mathrm{abs}(x)$
$\mathrm{re}(\pi)$	π
$\mathrm{re}(e)$	e
$\mathrm{re}(i)$	0
$\mathrm{re}(-x)$	$-\mathrm{re}(x)$
$\mathrm{re}(x+y)$	$\mathrm{re}(x)+\mathrm{re}(y)$
$\mathrm{re}(x-y)$	$\mathrm{re}(x)-\mathrm{re}(y)$

Imaginary Part

expression	**simplified**
$\mathrm{im}(\mathrm{im}(x))$	0
$\mathrm{im}(\mathrm{re}(x))$	0
$\mathrm{im}(\mathrm{conj}(x))$	$-\mathrm{im}(x)$
$\mathrm{im}(\mathrm{abs}(x))$	0
$\mathrm{im}(\pi)$	0
$\mathrm{im}(e)$	0
$\mathrm{im}(i)$	1
$\mathrm{im}(-x)$	$-\mathrm{im}(x)$
$\mathrm{im}(x+y)$	$\mathrm{im}(x)+\mathrm{im}(y)$
$\mathrm{im}(x-y)$	$\mathrm{im}(x)-\mathrm{im}(y)$

Conjugate

expression	**simplified**
$\mathrm{conj}(\mathrm{conj}(x))$	x
$\mathrm{conj}(\mathrm{re}(x))$	$\mathrm{re}(x)$
$\mathrm{conj}(\mathrm{im}(x))$	$\mathrm{im}(x)$
$\mathrm{conj}(\mathrm{abs}(x))$	$\mathrm{abs}(x)$
$\mathrm{conj}(\pi)$	π
$\mathrm{conj}(e)$	e
$\mathrm{conj}(i)$	$-i$
$\mathrm{conj}(-x)$	$-\mathrm{conj}(x)$
$\mathrm{conj}(x+y)$	$\mathrm{conj}(x)+\mathrm{conj}(y)$
$\mathrm{conj}(x-y)$	$\mathrm{conj}(x)-\mathrm{conj}(y)$

B Command Reference

We give below a listing of all the commands provided by `ALG48` with stack diagrams showing the argument(s) they require. We use the following abbreviations

x or y	Real numbers.
z	Integer real number.
n	Positive integer real number.
$\#z$	Unlimited precision binary integer (hexstring).
$\#n$	Positive binary integer (hexstring).
"z"	Character string representing a number in decimal.
'symb'	Symbolic expression or variable.
'x'	Variable (local or global).
s	Scalar (real number, variable, or symbolic expression).
{ vector }	Symbolic vector, represented by a list of the form $\{a_1 \ldots a_n\}$ where each a_i is a scalar.
{{ matrix }}	Symbolic matrix, represented by a list of the form $\{\{a_{11} \ldots a_{1m}\} \ldots \{a_{n1} \ldots a_{nm}\}\}$ where each a_{ij} is a scalar.
{{ sq-matrix }}	Square symbolic matrix.
{ 'x' 'y' 'z' }	List of variables.
{ 'eq1' 'eq2' }	System of equations, represented by a list of symbolic equations or expressions. Expressions are interpreted as equations with zero on the right-hand side.
{ s-list }	List whose elements are either scalars or s-list themselves (includes vectors, matrices, and system of equations).

The entries marked with an asterisk ($*$) in the stack diagrams are the operations affected by the status of the automatic simplification flag (see Section 4.5).

- `RSIM` – Rational simplification command

Level 1	→	**Level 1**
'symb_1'	→	'symb_2'
{ s-list_1 }	→	{ s-list_2 }
z	→	z

- `FCTR` – Factorization command

Level 1	→	**Level 1**
'symb_1'	→	'symb_2'
{ s-list_1 }	→	{ s-list_2 }
z	→	z
#z	→	{ #z1 #z2 ... #zk }
"z"	→	'z1*z2* ... *zk'

- `AADD` – Algebraic addition command

Level 2	Level 1	→	Level 1	
{{ matrix_1 }}	{{ matrix_2 }}	→	{{ matrix_2 + matrix_1 }}	(∗)
{ vector_1 }	{ vector_2 }	→	{ vector_2 + vector_1 }	(∗)
{{ sq_matrix }}	s	→	{{ I*s + sq_matrix }}	(∗)
s	{{ sq_matrix }}	→	{{ sq_matrix + I*s }}	(∗)
'symb_1'	'symb_2'	→	'symb_2+symb_1'	(∗)
'symb'	x	→	'x+symb'	(∗)
x	'symb'	→	'symb+x'	(∗)
#z1	#z2	→	#z3	
#z1	z2	→	#z3	
z1	#z2	→	#z3	

- `ASUB` – Algebraic subtraction command

Level 2	Level 1	→	Level 1	
{{ matrix_1 }}	{{ matrix_2 }}	→	{{ matrix_2 - matrix_1 }}	(∗)
{ vector_1 }	{ vector_2 }	→	{ vector_2 - vector_1 }	(∗)
{{ sq-matrix }}	s	→	{{ I*s - sq_matrix }}	(∗)
s	{{ sq-matrix }}	→	{{ sq_matrix - I*s }}	(∗)
'symb_1'	'symb_2'	→	'symb_2-symb_1'	(∗)
'symb'	x	→	'x-symb'	(∗)
x	'symb'	→	'symb-x'	(∗)
#z1	#z2	→	#z3	
#z1	z2	→	#z3	
z1	#z2	→	#z3	

- `AMUL` – Algebraic multiplication command

Level 2	Level 1	→	Level 1	
{{ matrix_1 }}	{{ matrix_2 }}	→	{{ matrix_2 * matrix_1 }}	(∗)
{{ matrix }}	{ vector }	→	{{ matrix * vector }}	(∗)
{ vector }	{{ matrix }}	→	{{ matrix * vector }}	(∗)
{{ matrix }}	s	→	{{ s * matrix }}	(∗)
s	{{ matrix }}	→	{{ matrix * s }}	(∗)
{ vector }	s	→	{ s * vector }	(∗)
s	{ vector }	→	{ vector * s }	(∗)
'symb_1'	'symb_2'	→	'symb_2*symb_1'	(∗)
'symb'	x	→	'x*symb'	(∗)
x	'symb'	→	'symb*x'	(∗)
#z1	#z2	→	#z3	
#z1	z2	→	#z3	
z1	#z2	→	#z3	

- `ADIV` - Algebraic division command

Level 2	**Level 1**	→	**Level 1**	
{{ matrix }}	{{ sq-matrix }}	→	{{ matrix * (sq-matrix)^-1 }}	
{ vector }	{{ sq-matrix }}	→	{{ vector * (sq-matrix)^-1 }}	
s	{{ sq-matrix }}	→	{{ s * (sq-matrix)^-1 }}	
{{ matrix }}	s	→	{{ matrix / s }}	(∗)
{ vector }	s	→	{ vector / s }	(∗)
'symb_1'	'symb_2'	→	'symb_1/symb_2'	(∗)
'symb'	x	→	'symb/x'	(∗)
x	'symb'	→	'x/symb'	(∗)
x	y	→	'x/y'	(∗)
#z1	#z2	→	#z3	
#z1	z2	→	#z3	
z1	#z2	→	#z3	

- `APOW` - Algebraic exponentiation command

Level 2	**Level 1**	→	**Level 1**	
{{ sq-matrix }}	z	→	{{ (sq-matrix)^z }}	(∗)
'symb'	z	→	'symb^z'	(∗)
#z1	#n	→	#z2	
#z1	n	→	#z2	
z1	#n	→	#z2	

- `ANEG` - Algebraic negation command

Level 1	→	**Level 1**
{ s-list_1 }	→	{ s-list_2 }
'symb'	→	'-symb'
#z1	→	#z2

- `AINV` - Algebraic inverse command

Level 1	→	**Level 1**	
{{ sq-matrix }}	→	{{ (sq-matrix)^-1 }}	
'symb'	→	'INV(symb)'	(∗)
x	→	'1/x'	(∗)

- `MDET` - Symbolic matrix determinant

Level 1	→	**Level 1**
{{ sq-matrix }}	→	'det(sq-matrix)'

- `MLU` - Symbolic matrix LU decomposition

Level 1	→	**Level 2**	**Level 1**
{{ sq-matrix_1 }}	→	{{ sq-matrix_2 }}	n

- `MTRN` - Symbolic matrix transpose

Level 1	→	**Level 1**
{{ matrix_1 }}	→	{{ matrix_2 }}
{ vector }	→	{{ matrix }}

- `MIDN` - Symbolic identity matrix

Level 1 →	**Level 1**
n →	{{ $(n \times n)$ identity-matrix }}

- `Z<->S` or `ZS` - Conversion from unlimited precision integer to string

Level 1	→	**Level 1**
#z	→	"z"
"z"	→	#z
z	→	#z

- `GCD` - Greatest Common Divisor command

Level 2	**Level 1**	→	**Level 1**
'poly_1'	'poly_2'	→	'poly_gcd'
'poly'	x	→	z
x	'poly'	→	z
z1	z2	→	z3
z1	#z2/"z2"	→	#z3
#z1/"z1"	z2/#z2/"z2"	→	#z3

- `LCM` - Least Common Multiple command

Level 2	**Level 1**	→	**Level 1**
'poly_1'	'poly_2'	→	'poly_lcm'
'poly'	x	→	'poly_lcm'
x	'poly'	→	'poly_lcm'
z1	z2	→	z3
z1	#z2/"z2"	→	#z3
#z1/"z1"	z2/#z2/"z2"	→	#z3

- `RAT->` - Rational to stack command

Level 1	→	**Level 2**	**Level 1**
'rational function'	→	'numerator'	'denominator'
x	→	x	1.0

- `ASIM` – Algebraic simplification command

Level 1	→	**Level 1**
'symb_1'	→	'symb_2'
{ s-list_1 }	→	{ s-list_2 }
(x, y)	→	$'x + yi'$
z	→	z

- `MOD+` – Modular addition

Level 3	**Level 2**	**Level 1**	→	**Level 1**
z1/#z1/"z1"	z2/#z2/"z2"	n/#n/"n"	→	#z3

- `MOD-` – Modular substraction

Level 3	**Level 2**	**Level 1**	→	**Level 1**
z1/#z1/"z1"	z2/#z2/"z2"	n/#n/"n"	→	#z3

- `MOD*` – Modular multiplication

Level 3	Level 2	Level 1	→	Level 1
z1/#z1/"z1"	z2/#z2/"z2"	n/#n/"n"	→	#z3

- `MOD/` – Modular division

Level 3	Level 2	Level 1	→	Level 1
z1/#z1/"z1"	z2/#z2/"z2"	n/#n/"n"	→	#z3

- `MODPOW` – Modular exponentiation

Level 3	Level 2	Level 1	→	Level 1
z1/#z1/"z1"	z2/#z2/"z2"	n/#n/"n"	→	#z3

- `MODINV` – Inverse modulo N

Level 2	Level 1	→	Level 1
z1/#z1/"z1"	n/#n/"n"	→	#z2

- `PRIM?` – Prime testing

Level 1	→	Level 1
z/#z/"z"	→	0/1

- `PRIM+` – Next prime

Level 1	→	Level 1
z/#z/"z"	→	#n

- `PRIM-` – Previous prime

Level 1	→	Level 1
z/#z/"z"	→	#n

- `RORD` – Reorder polynomial

Level 2	Level 1	→	Level 1
'poly_1'	'x'	→	'poly_2'
z	'x'	→	z
'poly_1'	{ 'x' 'y' 'z' }	→	'poly_2'
z	{ 'x' 'y' 'z' }	→	z

- `PF` – Partial fraction expansion

Level 2	Level 1	→	Level 1
'symb_1'	{ 'x' 'y' 'z' }	→	'symb_2'
z	{ 'x' 'y' 'z' }	→	z
	'symb_1'	→	'symb_2'
	z	→	z

- `RINT` – Rational function integration

Level 2	Level 1	→	Level 1
'symb_1'	'x'	→	'symb_2'
z	'x'	→	'z*x'

- `GBASIS` – Gröbner basis of a system of polynomial equations

Level 2	Level 1	$\rightarrow$	Level 1
{ 'eq1' 'eq2' }	{ 'x' 'y' 'z' }	$\rightarrow$	{ 'eq1' 'eq2' }$'$

- `GSOLVE` – Solutions of a system of polynomial equations

Level 2	Level 1	$\rightarrow$	Level $n+1$	...	Level 2	Level 1
{ 'eq1' 'eq2' }	{ 'x' 'y' 'z' }	$\rightarrow$	{ 'eq1' 'eq2' }$_n$	...	{ 'eq1' 'eq2' }$_1$	n

- `GSIMP` – Reduction of an expression given a system of side relations

Level 3	Level 2	Level 1	$\rightarrow$ Level 1
'symb1'	{ 'eq1' 'eq2' }	{ 'x' 'y' 'z' }	$\rightarrow$ 'symb2'