WARSAW UNIVERSITY OF TECHNOLOGY

Programming in Logic and Symbolic Programming

Faculty of Mathematics and Information Science

Symbolic Computation on Polynomials

- Theoretical Document -

By **Elie SAAD & Matthieu Dabrowski**

May 7, 2021

Contents

1 Definitions

Definition 1.1 (Mathematical Pseudo-Language)**.** Mathematical pseudo-language (MPL) is an algorithmic language that is used throughout this paper to describe the concepts, examples, and algorithms of computer algebra.

Definition 1.2 (Symbolic Computation)**.** Symbolic computation is a scientific area in computational mathematics that refers to the study and development of algorithms and software for manipulating mathematical expressions and other mathematical objects.

Definition 1.3 (Group)**.** A group consists of a set G and a binary operation "$\cdot$" defined on G, for which the following conditions are satisfied:

1. Associative: $(a \cdot b) \cdot c = a \cdot (b \cdot c)$, for all $a, b, c \in G$
2. Identity: There exist $1 \in G$ such that $a \cdot 1 = 1 \cdot a = a$, for all $a \in G$
3. Multiplicative inverse: Given $a \in G$, there exists $b \in G$ such that $a \cdot b = b \cdot a = 1$.

Definition 1.4 (Ring)**.** A commutative ring (with identity) consists of a set k and two binary operations "$\cdot$" and "$+$", defined on k, for which the following conditions are satisfied:

1. Associative: $(a + b) + c = a + (b + c)$ and $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ for all $a, b, c \in k$
2. Commutative: $a + b = b + a$ and $a \cdot b = b \cdot a$, for all $a, b \in k$.
3. Distributive: $a \cdot (b + c) = a \cdot b + a \cdot c$, for all $a, b, c \in k$.
4. Identity: There exist $0, 1 \in k$ such that $a + 0 = a \cdot 1 = a$, for all $a \in k$
5. Additive inverse: Given $a \in k$, there exists $b \in k$ such that $a + b = 0$.

Definition 1.5 (Field)**.** A field consists of a set k and two binary operations "$\cdot$" and "$+$", defined on k, for which the following conditions are satisfied:

1. Associative: $(a + b) + c = a + (b + c)$ and $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ for all $a, b, c \in k$
2. Commutative: $a + b = b + a$ and $a \cdot b = b \cdot a$, for all $a, b \in k$.
3. Distributive: $a \cdot (b + c) = a \cdot b + a \cdot c$, for all $a, b, c \in k$.

4. Identity: There exist $0, 1 \in k$ such that $a + 0 = a \cdot 1 = a$, for all $a \in k$
5. Additive inverse: Given $a \in k$, there exists $b \in k$ such that $a + b = 0$.
6. Multiplicative inverse: Given $a \in k$, $a \neq 0$, there exists $c \in k$ such that $a \cdot c = 1$.

Definition 1.6 (Polynomial)**.** Consider a given field k, and let $x_1, ..., x_n$ be indeterminates. Then a polynomial f in $x_1, ..., x_n$ with coefficients in a field k is a finite linear combination of monomials:

$$f = \sum_{\alpha} c_\alpha x^\alpha = \sum_{\alpha} c_\alpha x_1^{\alpha_1} ... x_n^{\alpha_n}, \qquad c_\alpha \in k$$

where the sum is over a finite number of n-tuples $\alpha = (\alpha_1, ..., \alpha_n)$, $\alpha_i \in \mathbb{N}_0$. The set of all polynomials in $x_1, ..., x_n$ with coefficients in k is denoted $k[x_1, ..., x_n]$.

Definition 1.7 (General Monomial Expression)**.** Let $c_1, c_2, ..., c_r$ be algebraic expressions, and let $x_1, x_2, ..., x_m$ be algebraic expressions that are not integers or fractions. A general monomial expression (GME) in $\{x_1, x_2, ..., x_m\}$ has the form

$$c_1 c_2 ... c_r x_1^{n_1} x_2^{n_2} ... x_m^{n_m},$$

where the exponents n_j are non-negative integers and each c_i satisfies the independence property explained in the next chapter

$$Free\_of(c_i, x_j) \to true, j = 1, 2, ..., m.$$

The expressions x_j are called generalized variables because they mimic the role of variables, and the expressions c_i are called generalized coefficients because they mimic the role of coefficients. The expression $x_1^{n_1} x_2^{n_2} ... x_m^{n_m}$ is called the variable part of the monomial, and if there are no generalized variables in the monomial, the variable part is 1. The expression $c_1 c_2 ... c_r$ is called the coefficient part of the monomial, and if there are no generalized coefficients in the monomial, the coefficient part is 1.

Definition 1.8 (General Polynomial Expression)**.** An expression u is a general polynomial expression (GPE) if it is either a GME or a sum of GMEs in $\{x_1, x_2, ..., x_m\}$.

2 Mathematical Expressions

MPL mathematical expressions are constructed with the following symbols and operators:

- Integers and fractions that utilize infinite precision rational number arithmetic.
- Identifiers that are used both as programming variables that represent the result of a computation and as mathematical symbols that represent indeterminates (or variables) in a mathematical expression.
- The algebraic operators $+, -, \cdot, /, \wedge$ (power), and ! (factorial). (As with ordinary mathematical notation, we usually omit the "$\cdot$" operator and use raised exponents for powers.)
- Function forms that are used for mathematical functions ($\sin(x)$, $\exp(x)$, $\arctan(x)$, etc.), mathematical operators ($Expand(u)$, $Factor(u)$, $Integral(u, x)$, etc.), and undefined functions ($f(x), g(x, y)$, etc.).
- The relational operators $=, \neq, <, \leq, >$, and $\geq$, the logical constants *true* and $false$, and the logical operators, *and*, *or*, and *not*.
- Finite sets of expressions that utilize the set operations $\cup$, $\cap$, $\sim$ (set difference), and $\in$ (set membership). Following mathematical convention, sets do not contain duplicate elements and the contents of a set does not depend on the order of the elements (e.g., $\{a, b\} = \{b, a\}$).
- Finite lists of expressions. A list is represented using the brackets [and] (e.g., $[1, x, x2]$). The empty list, which contains no expressions, is represented by []. Lists may contain duplicate elements, and the order of elements is significant (e.g., $[a, b] \neq [b, a]$).

The MPL set and list operators and the corresponding operators in computer algebra systems are given as follows:

- Set notation: $\{a, b, c\}$
- Set union: $A \cup B$
- Set intersection: $A \cap B$
- Set difference: $A \sim B$
- Set membership: $x \in B$

- List notation: $[a, b, c]$
- First member of L: $First(L)$
- A new list with first member of L removed: $Rest(L)$
- A new list with x adjoined to the beginning of L: $Adjoin(L)$
- A new list with members of L followed by members of M: $Join(L, M)$
- List membership: $x \in B$

MPL mathematical expressions have two (somewhat overlapping) roles as either program statements that represent a computational step in a program or as data objects that are processed by program statements.

3 Assignments, Functions, and Procedures

The MPL assignment operator is a colon followed by an equal sign (:=) and an assignment statement has the form $f := u$ where u is a mathematical expression.
An MPL function definition has the form $f(x_1, ..., x_l) \overset{\text{function}}{:=} u$, where $x_1, ..., x_l$ is a sequence of symbols called the formal parameters, and u is a mathematical expression. MPL procedures extend the function concept to mathematical operators that are defined by a sequence of statements. The general form of an MPL procedure is given as follows:

s[example algorithm]

Functions and procedures are invoked with an expression of the form $f(a_1, ..., a_l)$, where $a_1, ..., a_l$ is a sequence of mathematical expressions called the actual parameters.
In order to promote a programming style that works for all languages, we adopt the following conventions for the use of local variables and formal parameters in a procedure:

- An unassigned local variable cannot appear as a symbol in a mathematical expression. In situations where a procedure requires a local (unassigned) mathematical symbol, we either pass the symbol through the parameter list or use a global symbol.

- Formal parameters are used only to transmit data into a procedure and not as local variables or to return data from a procedure. When we need to return more than one expression from a procedure, we return a set or list of expressions.

4 Decision and Iteration Structures

MPL provides three decision structures: the if structure, the $if - else$ structure which allows for two alternatives, and the multi-branch decision structure which allows for a sequence of alternatives.
MPL contains two iteration structures that allow for repeated evaluation of a sequence of statements, the *while* structure and the for structure. Some of our procedures contain for loops that include a *Return* statement. In this case, we intend that both the loop and the procedure terminate when the Return is encountered.

5 Preliminaries and Notation

The following technical notation and assumptions are used:

- $[i, j] = \{i, i + 1, ..., j\}$ stands for the ordered set of integers ranging from i to j.

- $Kind(u)$: This operator returns the type of expression (e.g., *symbol*, *integer*, $fraction$, $+$, $\cdot$, a, $!$, $=$, $<$, $\leq$, $>$, $\geq$, $\coloneqq$, *and*, *or*, *not*, *set*, *list*, and function names). For example,

$$Kind(m \cdot x + b) \rightarrow +.$$

- $Operand(u, i)$: This operator returns the i-th operand of the main operator of u. For example,

$$Operand(m \cdot x + b, 2) \rightarrow b.$$

- $Free\_of(u, t)$: Let u and t (for target) be mathematical expressions. This operator returns false when t is identical to some complete subexpression of u and otherwise returns true. For example

$$Free\_of((a + b)c, a + b) \rightarrow false.$$

- $Substitute(u, t = r)$: Let u, t, and r be mathematical expressions. This operator forms a new expression with each occurrence of the target expression t in u replaced by the replacement expression r. The substitution occurs whenever t is structurally identical to a complete sub-expression of u. For example

$$Substitute((a+b)c, a+b=x) \to xc.$$

- $Degree\_gpe(u, x)$: Let $u = c_1 c_r \cdot x_1^{n_1} x_m^{n_m}$ be a monomial with non-zero coefficient part. The degree of u with respect to x_i is denoted by $deg(u, x_i) = n_i$. By mathematical convention, the degree of the 0 monomial is $-\infty$. If u is a GPE in x_i that is a sum of monomials, then $deg(u, x_i)$ is the maximum of the degrees of the monomials. If the generalized variable x_i is understood from context, we use the simpler notation $deg(u)$. For example,

$$Degreegpe(\sin^2(x) + b\sin(x) + c, \sin(x)) \to 2$$

- $Coefficient\_gpe(u, x, j)$: Let u be a GPE in x, and let j be a nonnegative integer. The MPL operator Coefficient $gpe(u, x, j)$ returns the sum of the coefficient parts of all monomials of u with variable part x_j. For example,

$$Coefficient\_gpe(ax + bx + y, x, 1) \to a + b.$$

- $Leading\_coefficient\_gpe(u, x)$: Let u be a GPE in x. The leading coefficient of a GPE $u \neq 0$ with respect to x is the sum of the coefficient parts of all monomials with variable part $x^{deg(u,x)}$. The zero polynomial has, by definition, leading coefficient zero. The leading coefficient is represented by $lc(u, x)$, and when x is understood from context by $lc(u)$. The MPL operator $Leading\_coefficient\_gpe(u, x)$ returns $lc(u, x)$. For example,

$$Leading\_coefficient\_gpe(ax + bx + y, x) \to a + b.$$

- $Iquot(a, b)$ and $Irem(a, b)$: For integers a and $b \neq 0$, there are unique integers q and r such that

$$a = qb + r$$

and

$$0 \leq r \leq |b| - 1$$

The integer q is the quotient and is represented by the operator $iquot(a, b)$ (for integer quotient). The integer r is the remainder and is represented by $irem(a, b)$.

- $Find\_min\_deg(S, x)$: Let u be a polynomial in $Q[x]$, $Find\_min\_deg(S, x)$ is a procedure that returns a polynomial of smallest degree in S. If $S = \emptyset$, it returns the global symbol $Undefined$.
- $Polynomial\_divisors(u, x)$: Let u be a polynomial in $Q[x]$, $Polynomial\_divisors(u, x)$ is a procedure that returns the set of monic divisors of u with positive degree. If u is not a polynomial in x, it returns the global symbol $Undefined$.
- $Algebraic_expand(u)$: Let u be a polynomial in $Q[x]$, $Algebraic_expand(u)$ transforms an algebraic expression u into its expanded form.
- $Concatenate(a, b)$: Let a and b be two lists, $Concatenate(a, b)$ returns a list composed of the elements of a followed by the elements of b.

6 Algorithms

6.1 Polynomial Addition and Subtraction

Algorithm 1 Polynomial Addition

Input:

Q_1, Q_2: two polynomials in $Q[X]$;

Output:

A polynomial equal to the sum of Q_1 and Q_2;

$n_1 \coloneqq deg(Q_1)$;
$n_2 \coloneqq deg(Q_2)$;
if $Q_1 = [\,]$ and $Q_2 = [\,]$ **then**
    return $[\,]$
else if $Q_1 = [\,]$ and $Q_2 \neq [\,]$ **then**
    return Q_2
else if $Q_1 \neq [\,]$ and $Q_2 = [\,]$ **then**
    return Q_1
else
    $R_1 \coloneqq Q_1[2 : (1 + n_1)]$;
    $R_2 \coloneqq Q_2[2 : (1 + n_2)]$;
    return $Concatenate([Q_1[1] + Q_2[1]], Polynomial\_Addition(R_1, R_2))$
end if

Algorithm 2 Polynomial Substraction

Input:

Q_1, Q_2: two polynomials in $Q[X]$;

Output:

A polynomial equal to the difference between of Q_1 and Q_2;

1: $n_1 \coloneqq deg(Q_1)$;
2: $n_2 \coloneqq deg(Q_2)$;
3: **if** $Q_1 = [\,]$ and $Q_2 = [\,]$ **then**
4: **return** $[\,]$
5: **else if** $Q_1 = [\,]$ and $Q_2 \neq [\,]$ **then**
6: **return** $(-1) * Q_2$
7: **else if** $Q_1 \neq [\,]$ and $Q_2 = [\,]$ **then**
8: **return** Q_1
9: **else**
10: $R_1 \coloneqq Q_1[2 : (1 + n_1)]$;
11: $R_2 \coloneqq Q_2[2 : (1 + n_2)]$;
12: **return** $Concatenate([Q_1[1]-Q_2[1]], Polynomial\_Substraction(R_1, R_2))$
13: **end if**

These algorithms take the first element of each of the two vectors, add or substract them, then delete these elements from their original vectors. This way we can call the methods on the new shorten vectors since none of the elements from these vector have been added/substracted yet.
As soon as one of the two vectors is empty, we directly put the other vector (multiplied by -1 if necessary) at the end of the result vector, because we don't need to calculate the new elements. The resulting vector represents the sum of the two polynomials.
If the two vectors become empty at the same time, then the final result vector (representing the sum of the two polynomials) has already been found.

6.2 Polynomial Multiplication

Algorithm 3 Monomial Polynomial Multiplication

Input:

P_1: a polynomial in $Q[X]$;
M_2: a monomial in $Q[X]$;

Output:

Q_1: a polynomial equal to the product of P_1 and M_2;

1: $Q_1 \coloneqq P_1$;
2: $S_2 \coloneqq M_2$;
3: $n_1 \coloneqq deg(Q_1)$;
4: $n_2 \coloneqq deg(S_2)$;
5: **if** $n_2 = 0$ **then**
6: $\quad Q_1 \coloneqq S_2[1] * Q_1$;
7: $\quad$ **return** Q_1
8: **else**
9: $\quad S_2 \coloneqq S_2[2 : (1 + n_2)]$;
10: $\quad Q_1 \coloneqq Concatenate([0], Q_1)$;
11: $\quad$ **return** $Monomial\_Polynomial\_Multiplication(Q_1, S_2)$
12: **end if**

Here we simply add at the beginning the vector representing the polynomial P_1 a number of zeros equal to the degree of the monomial M_2. Then we multiply the obtained polynomial by the only non-zero coefficient of M_2.
But of course, we do this in a recursive way, which means that, at each iteration of our recursion, we take basically a zero from the vector representing M_2 and put it at the beginning of the vector representing P_1. At some point the vector representing M_2 will only contain its only non-zero coefficient. At that point, P_1 will be multiplied by this number, and we will obtain the product of the two original elements.
If the monomial M_2 is equal to zero, then at the very end the polynomial P_1 will be multiplied by zero, and we will still obtain the product of our two inputs.

Algorithm 4 Polynomials Multiplication

Input:

P_1, P_2: two polynomials in $Q[X]$;

Output:

R_{es}: a polynomial equal to the product of P_1 and P_2;

1: $R_{es} \coloneqq [\,]$
2: $Z \coloneqq [\,]$
3: $n_1 \coloneqq deg(P_1)$
4: $n_2 \coloneqq deg(P_2)$
5: **for** $i = 1, \ldots, (1 + n_2)$ **do**
6: **if** $i \geq 2$ **then**
7: $Z \coloneqq Concatenate([0], Z)$;
8: **end if**
9: **if** $P_2[i] \neq 0$ **then**
10: $M_i \coloneqq Concatenate(Z, P_2[i])$;
11: $N_i \coloneqq Monomial\_Polynomial\_Multiplication(P_1, M_i)$;
12: $R_{es} \coloneqq Polynomial\_Addition(R_{es}, N_i)$;
13: **end if**
14: **end for**
15: **return** R_{es}

Here, for each non-zero element a_i of the vector representing P_2, a monomial M_i is created, containing a_i as its only non-zero element. The position of a_i is kept in M_i, which means zeros are added before a_i to make sure it still applies to the same power of X as it did in P_2.
The final result is the sum of the products of P_1 with each monomial M_i obtained this way.
At each iteration, the multiplication of P_1 with M_i is computed using the **Algorithm 3**.

6.3 Polynomial Division

For single variable polynomials, the familiar process of long division of u by v is defined formally using recurrence relations.

Definition 6.1. Let u and $v \neq 0$ be two polynomials in $F[x]$, and consider

the sequence of quotients q_i and remainders r_i:

$$\begin{aligned} q_0 &= 0, \qquad r_0 = u, \\ q_i &= q_{i-1} - \frac{lc(r_{i-1})}{lc(v)} x^{deg(r_{i-1})-deg(v)}, \\ r_i &= r_{i-1} - \frac{lc(r_{i-1})}{lc(v)} x^{deg(r_{i-1})-deg(v)} v \end{aligned} \tag{1}$$

The iteration terminates when

$$deg(r_i) < deg(v).$$

If the process terminates with $i = \sigma$, then q_σ is the quotient of u divided by v, and r_σ is the remainder. We also represent the quotient and the remainder by the operators $quot(u, v, x)$ and $rem(u, v, x)$.

We also know that for a polynomial with a positive degree $u \in F[x]$, a root $c \in F$ of $u = 0$, then u can be factored as $u = (x - c)q$ where $q \in F[x]$. The proof for this theorem can be found in any math book that discusses polynomial operations.

We give the following algorithm for polynomial division and functions that extract the quotient and remainder:

Algorithm 5 Polynomial Division

Input:

u, v: GPEs in x with $v \neq 0$;

x: a symbol;

Output:

a list $[q, r]$ with quotient q and remainder r;

1: $q \coloneqq 0$;
2: $r \coloneqq u$;
3: $m \coloneqq Degree\_gpe(r, x)$;
4: $n \coloneqq Degree\_gpe(v, x)$;
5: $lcv \coloneqq Leading\_Coefficient\_gpe(v, x)$;
6: **while** $m \geq n$ **do**
7: $\quad lcr \coloneqq Leading\_Coefficient\_gpe(r, x)$;
8: $\quad s \coloneqq lcr/lcv$;
9: $\quad q \coloneqq q + s \cdot x^{m-n}$;
10: $\quad r \coloneqq Algebraic\_expand((r - lcr \cdot x^m) - (v - lcv \cdot x^n) \cdot s \cdot x^{m-n})$;
11: $\quad m \coloneqq Degree\_gpe(r, x)$;
12: **end while**
13: **return** $[q, r]$

Observe that in line 10 we have used the expanded form of equation (1). In fact, as long as all the field operations in F are obtained by automatic simplification, we can use equation (1). However, when some field operations are not included in automatic simplification, equation (1) may not eliminate the leading term from r_{i-1}, and the algorithm may not terminate.

6.4 Polynomial GCD

Algorithm 6 Polynomial GCD

Input:

X, Y: two polynomials in $Q[X]$ with $X \neq [0]$;

Output:

The greatest common diviser of X and Y;

1: **if** $Y = [0]$ **then**
2: $n \coloneqq deg(X)$;
3: $lc \coloneqq Leading_coefficient_gpe(X)$;
4: $(D, R) \coloneqq Polynomial\_Division(X, [lc])$;
5: **return** D
6: **else**
7: $(Q, R) \coloneqq Polynomial\_Division(X, Y)$;
8: **return** $Polynomial\_GCD(Y, R)$
9: **end if**

Three theorems ([1], theorems **4.21**, **4.22** and **4.23**) have been used to create this algorithm. They all use the assumption that U and V are two polynomials in $F[X]$.

The first theorem states that the Greatest Common Diviser of U and V exists, is unique, and that if $V = 0$, then the GCD of U and $V = 0$ is equal to U divided by $Leading\_coefficient\_gpe(U)$.

The second one states that if $V \neq 0$, and if we call R the remainder of the division of U by V, then the GCD of U and V is equal to the GCD of V and R. Then of course it is also equal to the GCD of R and R_1 the remainder of the division of Y by R, and so on.

The third one relies on the two previous theorems. The main idea is that if we use the second theorem (which implies considering $V \neq 0$) to find the GCD of U and V a sufficient number of time, which implies computing a remainder R_i at each iteration i, then at some point one of the remainders R_i computed will be equal to 0. Which means that the GCD of U and V is equal to the GCD of the last non-zero remainder R_p and $R_{p+1} = 0$. Thanks to the first theorem, we then know that the GCD of U and V is equal to $R_p / Leading\_coefficient\_gpe(R_p)$.
The main interest of this third theorem is to ensure that our algorithm is converging to a solution and not endlessly computing remainders.

6.5 Polynomial Expansion

The polynomial expansion of u in terms of v involves the representation of u as a sum whose terms contain non-negative integer powers of v. The following polynomial expansion theorem is based on polynomial division. We use the theorem that states that for the polynomials $u, v \in F[x]$, suppose that v has a positive degree. Then there are unique polynomials

$$d_k(x), d_{k-1}(x), ..., d_0(x)$$

with $deg(d_i) < deg(v)$ such that

$$u = d_k v^k + d_{k-1} v^{k-1} + ... + d_1 v + d_0. \tag{2}$$

The representation of u in equation (2) is called the polynomial expansion of u in terms of v. The proof of this theorem can be found in any calculus book. When $v = x - c$, the polynomial expansion coincides with the Taylor expansion of u about $x = c$.

Definition 6.2. Let u and $v \neq 0$ be two polynomials in $F[x]$ with $deg(v) > 0$, and let t be a symbol. Define the polynomial expansion polynomial

$$P_u(t) = d_k t^k + d_{k-1} t^{k-1} + ... + d_1 t + d_0,$$

where the coefficients $d_i(x)$ are defined by the quotient sequence

$$\begin{aligned} u &= c_0 v + d_0, \\ c_0 &= c_1 v + d_1, \\ c_1 &= c_2 v + d_2, \\ &\vdots \\ c_{k-1} &= c_k v + d_k, \\ c_k &= 0 \end{aligned}$$

The polynomial P_u is simply the expansion (2) with the polynomial v replaced by the symbol t. In the Polynomial Expansion algorithm, we use the well known theorem that states that for polynomials u and $v \in F[x]$ with $deg(u, v) > 0$, suppose that from polynomial division, $u = qv + r$. Then, the expansion polynomial satisfies the recurrence relation

$$P_u = tP_q + r.$$

A recursive procedure to compute P_u is given in the algorithm below:

Algorithm 7 Polynomial Expansion

Input:

u: a GPEs in x;
v: a GPEs in x with $deg(v, x) > 0$;
x, t: symbols;

Output:

The polynomial P_u;

1: **if** $u = 0$ **then**
2: **return** 0
3: **else**
4: $d \coloneqq Polynomial\_division(u, v, x)$;
5: $q \coloneqq Operand(d, 1)$;
6: $r \coloneqq Operand(d, 2)$;
7: **return** $Algebraic\_expand(t \cdot Polynomial\_expansion(q, v, x, t) + r)$
8: **end if**

The procedure uses the condition in the theorem of polynomial expansion properties that states that $P_0 = 0$ for termination (lines 1–2). The Algebraic expand operator is applied to the recurrence relation (line 7) so that the expression is returned as a polynomial in t rather than as a composite expression. Notice that the procedure accepts u and v that are GPEs in x, although in some cases more simplification is required to obtain the expansion in a usable form.
Since $deg(v) > 0$, it follows that $deg(q) < deg(u)$, which implies that each successive recursive call involves a polynomial u of smaller degree. Therefore, for some recursive call $deg(u) < deg(v)$, which implies $q = 0$ at line 5, and the next call terminates the recursion. To get the representation for u in equation (2), evaluate the expression

$$Substitute(Polynomial_expansion(u, v, x, t), t = v).$$

6.6 Polynomial Decomposition

Finally, we describe an algorithm that finds a complete decomposition for u. The algorithm either finds a sequence of indecomposable polynomials $g_1, g_2, ..., g_p$ (with $p > 1$) such that

$$u = g_p \circ g_{p-1} \circ ... \circ g_1$$

or determines that u is indecomposable. The general idea is to build up the decomposition, one component at a time, taking care not to miss any intermediate components. We show below that it is possible to arrange things so that each of the partial compositions

$$g_i \circ g_{i-1} \circ ... \circ g_1, \quad 1 \leq i < p,$$

is a divisor of $u - u_0$ where the components g_i are obtained with polynomial expansions. We insure that each g_i is indecomposable by applying a minimal degree condition to the divisors of $u - u_0$. Let's see how to find the component g_i (for $i < p$) assuming that the components $g_1, g_2, ..., g_{i-1}$ have already been found. (The last component g_p is computed in a different way.) To simplify the notation, let

$$\begin{aligned} u &= g_p \circ g_{p-1} \circ ... \circ g_{i+1} \circ g_i \circ g_{i-1} \circ ... \circ g_1 \\ &= R \circ g_i \circ C, \end{aligned} \tag{3}$$

where

$$C = g_{i-1} \circ ... \circ g_1$$

is the decomposition so far and

$$R = g_p \circ g_{p-1} \circ ... \circ g_{i+1}$$

is the decomposition that remains to be found after g_i has been found. (For $i = 1$ we initialize C to x.) Since $i < p$, we can assume that $deg(R, x) > 0$. We suggest the following approach for finding a complete decomposition of $u(x)$. First, find all the divisors of $u - u_0$. To find g_1, initialize C to x, and find a divisor w of $u - u_0$ of smallest degree that satisfies the following properties

1. $deg(C, x) < deg(g_i \circ C, x) < deg(u, x)$.

2. $deg(g_i \circ C, x) | deg(u, x)$.

3. $(g_i \circ C)(0) = 0$.

4. $(g_i \circ C)(x) | (u(x) - u0)$.

and the following conditions

1. The polynomial R is found by computing the polynomial expansion of u in terms of $g_i \circ C$, and this expansion has coefficients that are free of x.

2. The polynomial g_i is found by computing the polynomial expansion of $g_i \circ C$ in terms of C, and this expansion has coefficients that are free of x.

In this case, we find g_i using the polynomial expansion of w in terms of C as is done in the following equation

$$(g_i \circ C)(x) = a_p(C(x))^p + a_{p-1}(C(x))^{p-1} + ... + a_C(x) + a_0$$

If no such divisor w exists, then C includes the first $p-1$ components in the decomposition and we need only compute the last component g_p. This component can be found by computing the polynomial expansion of u in terms of C. It is not difficult to show that g_p computed in this way is indecomposable.
An algorithm that obtains the complete decomposition of a polynomial in $Q[x]$ is shown below:

Algorithm 8 Polynomial Decomposition

Input:

u: a polynomial in $Q[x]$;

x: a symbol;

Output:

A decomposition $[g_p, g_{p-1}, ..., g_1]$ or u if no decomposition exists;

```
1:  U ≔ u − Coefficient_gpe(u, x, 0);
2:  S ≔ Polynomial_divisors(U, x);
3:  decomposition ≔ [ ];
4:  C ≔ x;
5:  while S ≠ ∅ do
6:      w ≔ Find_min_deg(S, x);
7:      S ≔ S − {w};
8:      if Degree_gpe(C, x) < Degree_gpe(w, x) and
            Degree_gpe(w, x) < Degree_gpe(u, x) and
            Irem(Degree_gpe(u, x), Degree_gpe(w, x)) = 0
        then
9:          g ≔ Polynomial_expansion(w, C, x, t);
10:         R ≔ Polynomial_expansion(u, w, x, t);
11:         if Free_of(g, x) and Free_of(R, x) then
12:             decomposition ≔ Adjoin(Substitute(g, t = x),
                                       decomposition);
13:             C ≔ w;
14:             final_component ≔ R;
15:         end if
16:     end if
17: end while
18: if decomposition = [ ] then
19:     return u
20: else
21:     return Adjoin(Substitute(final_component, t = x),
                      decomposition)
22: end if
```

The procedure returns either a list $[g_p, g_{p-1}, ..., g_1]$ that represents the decomposition or u if the polynomial is indecomposable. The while loop (lines 5–14) examines the divisors of $U = u - u_0$ and tries to choose an appropriate one to build up the decomposition. At line 6, the *Find_min_deg*

operator chooses a trial divisor w of minimal degree from the set S, and the statement at line 8 checks the degree requirements mentioned in the four properties above. If w passes these tests, the polynomial expansions referred to in (3) are computed (lines 9–10), and the two conditions above are checked in line 11. (Recall that the operator *Polynomial_expansion* returns a polynomial in terms of the fourth input parameter t with coefficients that may depend on x.) If both expansions are free of x, we have found the next component which is adjoined to the list *decomposition* (line 12). At line 13, C is assigned the value w in preparation for searching for the next component.

At line 14, the polynomial expansion R is assigned to $final\_component$. At this point, R has been used only to check the validity of w for computing the next component (line 11). However, when the next to last component $(g_p - 1)$ is computed, the current value of R is the polynomial expansion of u in terms of $g_{p-1} \circ ... \circ g1$ which is the last component g_p in the decomposition.

If we go through the entire loop without finding any components, the polynomial is indecomposable and we return the input polynomial (line 19). Otherwise, we add the $final\_component$ to the *decomposition* list (line 21).

Notice that nowhere in the algorithm do we use the condition $w(0) = 0$. In fact, no harm is done by choosing a divisor that does not satisfy this condition as long as all the other conditions are satisfied. The condition $w(0) = 0$ simply guarantees that we can limit our search to the divisors of $u - u_0$. If a divisor in S without this property is used, we simply get another decomposition.

7 Bibliography

[1] Joehl S. Cohen, 2002, *"Computer Algebra and Symbolic Computation - Mathematical Methods"*,
https://www.ukma.edu.ua/~yubod/teach/compalgebra/%5BJoel_S._Cohen%5D_Computer_algebra_and_symbolic_comp(BookFi.org).pdf