COMPUTER ALGEBRA

Franz Winkler *University of Delaware*

GLOSSARY

Algebraic number: Root of a polynomial in $\mathbb{Q}[x]$.

Euclidean domain: Integral domain D together with a grading function $\deg: D\text{-}\{0\} \to N$ such that $\deg(a^*b) \geq \deg(a)$ for all $a, b \in D$, $b \neq 0$, and for all $a, b \in D$, $b \neq 0$ there exist q and r in D such that $a = b^*q + r$ and $r = 0$ or $\deg(r) < \deg(b)$.

Factor: An element f of an integral domain D is a factor of $g \in D$ if f divides g.

Field: $(F, +, {}^*, 0, 1)$ is a ring such that $0 \neq 1$, * is commutative, and every nonzero element has a multiplicative inverse (i.e., for every $a \neq 0$ there is an a^{-1} such that $a^*a^{-1} = 1 = a^{-1*}a$).

Greatest common divisor: An element g of an integral domain D is a greatest common divisor of a and b in D if g divides both a and b and every other divisor of a and b divides g.

Group: $(G, +, 0)$ is a set G together with a binary operation $+$ such that $+$ is associative [i.e., $(a + b) + c = a + (b + c)$ for all a, b, c in G], 0 is a unit element for G (i.e., $0 + a = a = a + 0$ for all a in G), and every element in G has an inverse (i.e., for every a in G there is a $-a$ in G such that $a + (-a) = 0 = (-a) + a$). G is a commutative group if $+$ is also commutative (i.e., $a + b = b + a$ for all a, b in G).

Integral domain: Commutative ring without zero divisors (i.e., if $a^*b = 0$ then $a = 0$ or $b = 0$) and $0 \neq 1$.

Polynomial: A polynomial in the indeterminates $x_1, \ldots, x_\nu$ over the ring R is a function $p: \{x_1^{n_1} \cdots x_\nu^{n_\nu} | n_1, \ldots, n_\nu \in N\} \to N$, which is zero for all but a finite number of arguments; p is sometimes written as $\sum_{i=1}^{k} p(x_1^{n_{1,i}} \cdots x_\nu^{n_{\nu,i}}) x_1^{n_{1,i}} \cdots x_\nu^{n_{\nu,i}}$, where $x_1^{n_{1,i}} \cdots x_\nu^{n_{\nu,i}}$, $1 \leq i \leq k$, are the only arguments for which p is nonzero.

Polynomial ring: The polynomial ring in the indeterminates $x_1, \ldots, x_\nu$ over the ring R is the ring formed by all the polynomials in $x_1, \ldots, x_\nu$ over R, written $R[x_1, \ldots, x_\nu]$.

Relatively prime: Two elements a and b of an integral domain are relatively prime if 1 is a greatest common divisor of a and b.

Ring: $(R, +, {}^*, 0, 1)$ is a set R with two binary operations $+$ (addition) and * (multiplication) such that $(R, +, 0)$ is an abelian group, * is an associative operation on R, 1 is a unit element for *, and * is distributive over $+$ [i.e., $a^*(b + c) = (a^*b) + (a^*c)$ and $(a + b)^*c = (a^*c) + (b^*c)$ for all a, b, c in R].

Unique factorization domain: Integral domain in which every nonzero element can be uniquely expressed as a product of primes (up to multiplication by units and permutation of the factors).

The field of computer algebra comprises a wide range of basic goals, methods, and applications. As the name suggests, in computer algebra one is concerned with computing in algebraic structures. This might mean in the abstract setting of group theory, in differential fields with additional functions adjoined, in algebraic extensions of the rational number field, or in polynomial rings. In general the inputs to an algorithm are formulas and one also expects formulas as the result. Another important characteristic of computer algebra is that, in contrast to other kinds of scientific computation, every operation that is performed on the given objects is carried out in an exact way. This means that the result of the computation is the precise solution to the given problem and not just an approximation of it. As a consequence, computer algebra allows one to construct decision algorithms (e.g., for the validity of a formula in the elementary theory of real closed fields, for the solvability of a system of algebraic equations, for the solvability of a given integration problem), in which it is essential to have exact answers to certain questions. [*See* COMPUTER ALGORITHMS.]

I. Integration, Summation, and Differential Equations

A. INTEGRATION

The methods for indefinite integration that are commonly taught in calculus courses are heuristic methods: they may lead to an integral, but if they do not work, one still does not know whether an integral of some specified form exists or not. Recently the theory of integration has been greatly advanced by the development of decision procedures for certain classes of functions. [*See* CALCULUS; INTEGRAL EQUATIONS.]

The integral of a rational function $\int p(x)/q(x)\, dx$, where $p(x), q(x) \in \mathbb{Q}[x]$, $gcd(p,q) = 1$, and q monic, can be expressed as

$$\frac{g(x)}{f_2(x) \cdots f_r(x)^{r-1}} + c_1 \log(x - \alpha_1) + \cdots + c_n \log(x - \alpha_n)$$

for some $g(x) \in \mathbb{Q}[x]$, $\alpha_1, \ldots, \alpha_n$ the distinct roots of q and some $c_1, \ldots, c_n$ in $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$. Let $f_1 f_2^2 \cdots f_r^r, f_r \neq 1$, be the square-free decomposition of q (see Section VI). Then there exist two uniquely determined polynomials $g(x)$ and $h(x)$ with $\deg(g) < \deg(f_2 \ldots f_r^{r-1})$, $\deg(h) < \deg(f_1 \ldots f_r)$, such that

$$\int \frac{p(x)}{q(x)}\, dx = \frac{g(x)}{f_2(x) \cdots f_r(x)^{r-1}} + \int \frac{h(x)}{f_1(x) \cdots f_r(x)}\, dx \qquad (1)$$

The polynomials g and h can be computed by decomposing p/q into the square-free partial fraction decomposition

$$g_0 + \frac{g_{11}}{f_1} + \frac{g_{21}}{f_2} + \frac{g_{22}}{f_2^2} + \cdots + \frac{g_{r1}}{f_r} + \cdots + \frac{g_{rr}}{f_r^r}$$

where $g_0, g_{ij} \in \mathbb{Q}[x]$, $\deg(g_{ij}) < \deg(f_i)$ for $1 \leq j \leq i \leq r$, and then using the Hermite trick and integration by parts, which reduces an integral of the form

$$\int \frac{g(x)}{f(x)^m}\, dx \quad \deg(g) < \deg(f),\ f \text{ squarefree},\ n \geq 2$$

to an integral

$$\int \frac{h(x)}{f(x)^{m-1}}\, dx \qquad \deg(h) < \deg(f)$$

Another way of computing g and h is to differentiate Eq. (1), and solve the resulting equations for the coefficients of g and h by equating the coefficients of equal powers of x^i. That determines the rational part of the integral. What remains to be done is to compute the logarithmic part of the integral,

$$\int \frac{h(x)}{f_1(x) \cdots f_r(x)}\, dx$$

In the splitting field of q, $h(x)/f_1(x) \cdots f_r(x)$ can be expressed as $\sum_{i=0}^{n} c_i/(x - \alpha_i)$ and therefore $\int h(x)/f_1(x) \cdots f_r(x)\, dx = \sum_{i=0}^{n} c_i \log(x - \alpha_i)$. However, the computation of the splitting field, which in the worst case is of degree $n!$ over $\mathbb{Q}$, is practically impossible. Fortunately, the splitting field is not always necessary to express the integral. If p, q are relatively prime polynomials in $\mathbb{Q}[x]$ with q square-free and $\deg(p) < \deg(q)$, then

$$\int p(x)/q(x)\, dx = \sum_{i=0}^{n} c_i \log(v_i)$$

where the c_i are the distinct roots of $R(c) = \text{resultant}_x[p(x) - cq'(x), q(x)]$ and $v_i = gcd(p - c_i q', q)$.

Techniques similar to the ones in rational function integration can be used in an algorithm for the integration of transcendental elementary functions. The elementary functions are the meromorphic functions on some region of the complex numbers $\mathbb{C}$, built up from the rational functions by repeatedly doing algebraic operations, taking logarithms, and taking exponentials. This algorithm, due to R. H. Risch, is based on a theorem of J. Liouville, which is best presented in terms of differential fields.

A differential field is a field F, together with a derivation ($'$) on F, that is, a map of F into itself, that satisfies the rules $(a + b)' = a' + b'$ and $(ab)' = a'b + ab'$ for all a, b in F. An element a in F is called a constant if $a' = 0$. The constants of F form a subfield of F, the field of constants. A field E is called a differential extension field of F if the field F is contained in E and the derivation of E restricted to F is a derivation on F. If for x, y in E with $y \neq 0$, the relation $x' = y'/y$ holds, then x is called a logarithm of y or y an exponential of x. A differential extension E of the differential field F is an elementary extension of F if there exists a tower of differential fields $F = F_0 \subset F_1 \subset \cdots \subset F_n = E$ such that for each $1 \leq i \leq n$,

either $F_i = F_{i-1}(t)$ where t is a logarithm of an element of F_{i-1}
or $F_i = F_{i-1}(t)$ where t is an exponential of an element of F_{i-1}
or F_i is a finite algebraic extension of F_{i-1}

If $F = C(x)$, where C is a subfield of the field of complex numbers, then E is a field of elementary functions. If in addition each F_i is a transcendental extension of F_{i-1}, $1 \leq i \leq n$, then E is a field of transcendental elementary functions.

The decision procedure for the integration of elementary functions is based on

Liouville's theorem. Let F be a differential field of characteristic zero and let f be in F. There exists an elementary differential extension field of F having the same field of constants and containing an element y such that $y' = f$ if and only if there are constants $c_1, \ldots, c_n$ in F and elements $u, u_1, \ldots, u_n$ in F such that

$$f = u' + \sum_{i=1}^{n} c_i u_i'/u_i$$

For a given elementary integrand $f(x)$, one first has to find a field $C(x, t_1, \ldots, t_n) = F_n$ containing f that is an elementary extension of $C(x)$. Once f is represented as a rational function in $C(x, t_1, \ldots, t_n)$, Risch's algorithm decides whether f has an elementary integral, and if so finds it. For an overview of the transcendental case of Risch's algorithm, see the chapter on integration in finite terms in Buchberger *et al.* (1983).

As an example consider the problem of finding an integral for

$$f(x) = 2x\exp(x^2)\log(x) + \frac{\exp(x^2)}{x} + \frac{\log(x) - 2}{[\log^2(x) + x]^2} + \frac{(2/x)\log(x) + 1/x + 1}{\log^2(x) + x}$$

where $f(x)$ is in $\mathbb{Q}(x, t_1, t_2)$ and $t_1 = \exp(x^2)$, $t_2 = \log(x)$. The decision procedure treats $f(x)$ as the rational function

$$2xt_1t_2 + t_1/x + (t_2 - 2)/(t_2^2 + x)^2 + [(2/x)t_2 + (1/x) + 1]/(t_2^2 + x)$$

and yields the integral

$$t_1t_2 - t_2/(t_2^2 + x) + \log(t_2^2 + x) = \exp(x^2)\log(x) - \frac{\log(x)}{\log^2(x) + x} + \log[\log^2(x) + x]$$

as the solution to the integration problem.

B. Summation

Let a difference field be a field F together with an automorphism σ of F. For $f \in F$, let $\Delta f = \sigma(f) - f$ (upper difference operator), $\nabla f = f - \sigma^{-1}(f)$ (lower difference operator). Then $f \in F$ is called a constant of the difference field (F, σ), iff $\sigma(f) = f$. All the difference fields in this section are assumed to have characteristic zero.

1. EXAMPLE. $[\mathbb{Q}(x), \sigma]$, where $\sigma[f(x)] = f(x + 1)$, forms a difference field, $\Delta f(x) = f(x + 1) - f(x)$, $\nabla f(x) = f(x) - f(x - 1)$. Consider the elements of $\mathbb{Q}(x)$ as functions from the integers into $\mathbb{Q}$. Then

$$\Delta g = f \quad \text{if and only if}$$

$$\sum_{0 \leq i < n} f(i) = g(n) - g(0) \quad \text{for all } n \in N$$

$$\nabla h = f \quad \text{if and only if}$$

$$\sum_{0 < i \leq n} f(i) = h(n) - h(0) \quad \text{for all } n \in N$$

So if equations of the form $\Delta g = f$ ($\nabla h = f$) could be solved, then a "closed form" solution to the summation problem

$$\sum_{0 \leq i < n} f(i) = g(n) \qquad \left[\sum_{0 < i \leq n} f(i) = h(n)\right]$$

would be available.

The constants of a difference field (F, σ) form a subfield of F, the field of constants of (F, σ). The operators Δ and ∇ are linear and they satisfy the product rule

$$\Delta(f \cdot g) = f \cdot \Delta g + \Delta f \cdot g + \Delta f \cdot \Delta g$$
$$\nabla(f \cdot g) = f \cdot \nabla g + \nabla f \cdot g - \nabla f \cdot \nabla g$$

Let (F, σ) be a difference field, x transcendental over F, and extend σ by letting $\sigma(x) = x + 1$.

Then $(F(x), \sigma)$ is a difference field; $(F(x), \sigma)$ may be viewed as the field of rational functions from the integers to F. For $\alpha(x) \in F(x)$, one may extend $F(x)$ by $t(x)$, the "factorial" of $\alpha(x)$,

$$t(x) = \prod_{0 \le i < x} \alpha(i)$$

or by the "indefinite sum"

$$s(x) = \sum_{0 \le i < x} \alpha(i)$$

Karr calls extensions of this form Π-extensions or Σ-extensions. A $\Pi\Sigma$-field is a difference field constructed from some constant field by Π-extensions and Σ-extensions.

The problem of summation in finite terms is, given a difference field (F, σ) and $f \in F$, to find the solutions g of $\Delta g = f$ (or $\nabla g = f$) only in F.

Karr has developed a theory for solving arbitrary first-order linear difference equations—that is, equations of the form $\sigma(g)$-$a \cdot g = f$—for a and f in F, in any $\Pi\Sigma$-field F. This method is able to produce formulas (i.e., rational functions of n) for the indefinite sums

$$\sum_{i=1}^{n} i, \quad \sum_{i=0}^{n} 2^i, \quad \sum_{i=1}^{n} i^2 2^i,$$

$$\sum_{i=1}^{n} 1/(i^2 + 2i), \quad \sum_{i=1}^{n} i \cdot i!$$

and also show that no such formulas exist for

$$\sum_{i=1}^{n} 1/i, \quad \sum_{i=1}^{n} 1/i^2, \quad \sum_{i=1}^{n} 2^i/i, \quad \sum_{i=1}^{n} i!$$

Indefinite summation problems of the form $S(n) - S(0) = \sum_{i=1}^{n} a(i)$, where $a(n)/a(n-1)$ is a rational function of n, can be treated by Gosper's algorithm. It will produce the result $S(n)$, if $S(n)/S(n-1)$ is a rational function of n.

2. EXAMPLE. Use Gosper's algorithm for solving the summation problem

$$S(m) - S(0) = \sum_{n=1}^{m} \frac{\prod_{j=1}^{n-1} bj^2 + cj + d}{\prod_{j=1}^{n} bj^2 + cj + c}$$

$$\text{i.e.,}\quad a(n) = \frac{\prod_{j=1}^{n-1} bj^2 + cj + d}{\prod_{j=1}^{n} bj^2 + cj + c}$$

In a first step, polynomials $p(n)$, $q(n)$, and $r(n)$ are computed so that $a(n)/a(n-1) = [p(n)q(n)]/[p(n-1)r(n)]$ and $gcd[q(n), r(n+j)] = 1$ for all $j \ge 0$. For this example it suffices to choose $p(n) = 1$, $q(n) = b(n-1)^2 + c(n-1) + d$, $r(n) = bn^2 + cn + c$. Now if $S(n)/S(n-1)$ is a rational function of n, then $S(n)$ is of the form $[q(n+1)a(n)f(n)]/p(n)$, for some polynomial $f(n)$. A bound for the degree of f can be computed, which in this case turns out to be 0. Once a bound for the degree of f is known, it can be determined as the solution to the equation $p(n) = q(n+1)f(n) - r(n)f(n-1)$. So $f = 1/(d-c)$ and therefore $S(n) - S(0)$ is equal to

$$\frac{1 - \prod_{j=1}^{n} (bj^2 + cj + d)/(bj^2 + cj + c)}{c - d}$$

C. Solution of Differential Equations

The integration problem can, of course, be considered as a problem concerning a special kind of differential equation, namely $y' = f$. There are, however, decision algorithms for more general types of differential equations. As in the case of the integration problem, one has to determine the domain in which the solutions are sought. In the following this domain will be a Liouvillian field. [*See* DIFFERENTIAL EQUATIONS, ORDINARY.]

Let F with derivation $(')$ be a differential field of characteristic zero. A differential extension E of F is Liouvillian over F if there exists a tower of differential fields $F = F_0 \subset F_1 \subset \cdots \subset F_n = E$ such that for each $1 \le i \le n$,

either $F_i = F_{i-1}(t)$ where t'/t is in F_{i-1} (t is an exponential of an integral over F_{i-1})
or $F_i = F_{i-1}(t)$ where t' is in F_{i-1} (t is an integral over F_{i-1})
or F_i is a finite algebraic extension of F_{i-1}

A function is said to be Liouvillian if it is contained in some Liouvillian extension of $C(x)$, for C a subfield of the complex numbers.

In 1979, J. J. Kovacic derived an algorithm that decides whether the second-order homogeneous linear differential equation

$$y'' + ay' + by = 0, \qquad \text{where } a, b \text{ are in } C(x) \tag{2}$$

has a Liouvillian solution, and if it does finds a basis for the space of Liouvillian solutions. As a first step in the algorithm, the change of variables $z = y \exp[(\frac{1}{2}) \int a]$ is performed. This leads to an equation of the form

$$z'' + cz = 0 \tag{3}$$

A solution z to Eq. (3) is Liouvillian if and only if the corresponding solution y to (2) is Liouvillian. The decision procedure is based on the fact that exactly one of the following cases occurs:

(a) Equation (3) has a solution of the form $\exp(\int w)$ where w is in $C(x)$.
(b) Equation (3) has a solution of the form $\exp(\int w)$ where w is algebraic over $C(x)$ of degree 2.
(c) All solutions of Eq. (3) are algebraic over $C(x)$.
(d) Equation (3) has no Liouvillian solution.

In 1981, M. F. Singer discovered an algorithm for deciding whether the nth-order homogeneous linear differential equation with algebraic function coefficients

$$y^{(n)} + c_{n-1}y^{(n-1)} + \cdots + c_1y' + c_0y = 0 \quad (4)$$

has a Liouvillian solution and if it does finds a basis for the space of Liouvillian solutions. One of the basic facts used in Singer's algorithm is the following characterization of Liouvillian solutions to an equation of the form of Eq. (4), which can be proved using a group theoretic result of Jordan.

Theorem. Let F be a differential field with an algebraically closed field of constants. There exists a constant k that depends only on n so that if Eq. (4) has a Liouvillian solution over F, then it has a solution z where z'/z is algebraic over F of degree $\leq k$.

The algorithm proceeds by reducing the question of the existence of a Liouvillian solution of Eq. (4) to a problem in elimination theory and then constructing a basis for the vector space of Liouvillian solutions.

II. Quantifier Elimination

The *elementary theory of real closed fields* is the first-order theory with the constants 0 and 1, function symbols $+$, $-$, $\cdot$, predicate symbols $=$, $>$, $\geq$, $<$, $\leq$, $\neq$ (elementary algebra), and an axiom system consisting of the axioms for a (commutative) field, axioms that relate the order relation to the arithmetic operations

$$(\forall a)(\forall b)[a > 0 \;\&\; b > 0 \rightarrow a + b > 0]$$

$$(\forall a)(\forall b)[a > 0 \;\&\; b > 0 \rightarrow a \cdot b > 0]$$

and axioms that guarantee roots of certain polynomials

$$(\forall a)(\exists b)[a = b^2 \text{ or } -a = b^2]$$

for every $n \geq 1$:

$$(\forall a_0)(\forall a_1) \cdots (\forall a_{2n})(\exists b)$$
$$[a_0 + a_1b + \ldots + a_{2n}b^{2n} + b^{2n+1} = 0]$$

Models for the elementary theory of real closed fields are the field of real numbers $\mathbb{R}$ and the field of real algebraic numbers. Every formula ϕ of elementary algebra that is valid in one model of the elementary theory of real closed fields is valid in every model of this theory. So in order to decide ϕ for an arbitrary model, it suffices to decide it for $\mathbb{R}$.

In 1951, Tarski gave a quantifier elimination algorithm for transforming any formula ϕ of the theory of real closed fields into an equivalent formula that contains no quantifiers, and he also showed how to decide whether ϕ' is true for any formula ϕ' that does not contain quantifiers and variables. Seidenberg and Cohen described other methods for quantifier elimination. However, all these methods have an extremely high computational complexity. In particular, even if one fixes the number of variables r in the formula ϕ, they have a computing time that is exponential in both m, the number of polynomials occurring in ϕ, and n, the maximum degree of these polynomials.

In 1973, G. E. Collins discovered a new method for quantifier elimination, based on cylindrical algebraic decomposition. This method has a maximum computing time of

$$c(2n)^{2^{2r+8}}m^{2^{r+6}}d^3a$$

where d is the maximum length of any integer coefficient of any polynomial in ϕ, and a is the number of occurrences of atomic formulas in ϕ. In the following, Collins's quantifier elimination method is described.

Some notation is needed for stating the problem of quantifier elimination and the algorithm for cylindrical algebraic decomposition. A standard atomic formula is an expression of the form $A \sim 0$, where A is an integral polynomial and $\sim$ is a predicate symbol. A standard formula is a formula constructed from standard atomic formulas by use of propositional connectives and quantifiers, and a standard prenex formula is a standard formula of the form $(Q_{k+1}x_{k+1}) \ldots (Q_rx_r)\phi(x_1, \ldots, x_r)$, where $\phi(x_1, \ldots, x_r)$ is a quantifier-free standard formula containing the free variables $x_1, \ldots, x_r$, $0 \leq k \leq r$, and each Q_i is either an existential or a universal quantifier. Every formula in elementary algebra can be expressed as a standard prenex formula.

Let $\mathbb{R}^r$ denote the r-dimensional Euclidean space. A nonempty connected subset of $\mathbb{R}^r$ is called a *region*. The *cylinder* over a region R, written $Z(R)$, is $R \times \mathbb{R}^1$. A *section* of $Z(R)$ is a set s of points $(a_1, \ldots, a_r, f(a_1, \ldots, a_r))$, where $(a_1, \ldots, a_r)$ ranges over R, and f is a continuous, real-valued function on R. So s is the graph of f and it is also called the f-section of $Z(R)$. A sector of $Z(R)$ is a set s' of points $(a_1, \ldots, a_r, b)$, where $(a_1, \ldots, a_r)$ ranges over R and $f_1(a_1, \ldots, a_r) < b < f_2(a_1, \ldots, a_r)$ for continuous, real-valued functions $f_1 < f_2$. The constant functions $f_1 = -\infty$ and $f_2 = +\infty$ are allowed. Then s' is called the (f_1, f_2)-sector of $Z(R)$. A decomposition of a subset X of $\mathbb{R}^r$ is a finite collection of disjoint regions whose union is X. An element of a decomposition is called a cell of the decomposition. Continuous, real-valued functions $f_1 < f_2 < \cdots < f_k$, $k \geq 0$, defined on a region R, together with $f_0 = -\infty$, $f_{k+1} = +\infty$, naturally determine a decomposition of $Z(R)$ consisting of the (f_i, f_{i+1})-sectors of $Z(R)$ for $0 \leq i \leq k$, and the f_i-sections of $Z(R)$ for $1 \leq i \leq k$. Such a decomposition is called the stack over R determined by the functions $f_1, \ldots, f_k$. A decomposition D of $\mathbb{R}^r$ is cylindrical if either (a) $r = 1$, and $D = (D_1, \ldots, D_{2\nu+1})$, where either $\nu = 0$ and $D_1 = \mathbb{R}^1$ or $\nu > 0$ and there exist real numbers $\alpha_1 < \alpha_2 < \cdots < \alpha_\nu$ such that $D_1 = (-\infty, \alpha_1)$, $D_{2i} = \{\alpha_i\}$ for $1 \leq i \leq \nu$, $D_{2i+1} = (\alpha_i, \alpha_{i+1})$ for $1 \leq i < \nu$, $D_{2\nu+1} = (\alpha_\nu, \infty)$; or (b) $r > 1$, and there is a cylindrical decomposition $D' = (D_1, \ldots, D_\mu)$ of $\mathbb{R}^{r-1}$ such that $D = (D_{1,1}, \ldots, D_{1,2\nu_1+1}, \ldots, D_{\mu,1}, \ldots, D_{\mu,2\nu_\mu+1})$ and for each $1 \leq i \leq \mu$ $(D_{i,1}, \ldots, D_{i,2\nu_i+1})$ is a stack over D_i. D' is unique and is called the induced cylindrical decomposition of $\mathbb{R}^{r-1}$. If D is determined by algebraic functions $f_1, \ldots, f_k$, then it is a cylindrical algebraic decomposition (cad). A sample point for a cell of a decomposition is a point belonging to that cell. A sample of a cad $D = (D_1, \ldots, D_\mu)$ is a tuple $s = (s_1, \ldots, s_\mu)$ such that $s_i \in D_i$ for $1 \leq i \leq \mu$. The sample is algebraic in case each s_i is an algebraic point. A sample s is cylindrical if either (1) $r = 1$ or (2) $r > 1$ and there is a cylindrical sample $s' = (s_1, \ldots, s_\mu)$ of a cad D' of $\mathbb{R}^{r-1}$ such that $s = (s_{1,1}, \ldots, s_{1,2\nu_1+1}, \ldots, s_{\mu,1}, \ldots, s_{\mu,2\nu_\mu+1})$ and the first $r - 1$ coordinates of $s_{i,j}$ are, respectively, the coordinates of s_i, for all $1 \leq i \leq \mu$, $1 \leq j \leq 2\nu_i + 1$. A sample that is both cylindrical and algebraic is called a cylindrical algebraic sample (cas).

Let A be a set of integral polynomials in r variables. A cad D of $\mathbb{R}^r$ is A-invariant if every polynomial A in A is sign-invariant on every cell of D (i.e., is either positive, negative, or zero on the whole cell). If sample points for the cells of an A-invariant cad are available, then the sign of $\mathrm{A} \in A$ on a particular cell can be determined by evaluating A at the sample point for the cell. Exact computation is essential.

A standard formula $\phi(x_1, \ldots, x_r)$ containing just the free variables $x_1, \ldots, x_r$ is a defining formula for the subset X of $\mathbb{R}^r$ if X is the set of points in $\mathbb{R}^r$ satisfying ϕ. A standard definition of the cad $D = (D_1, \ldots, D_\mu)$ is a sequence $(\phi_1, \ldots, \phi_\mu)$ such that, for $1 \leq i \leq \mu$, ϕ_i is a standard quantifier-free defining formula for D_i.

Now the problem of quantifier elimination for the elementary theory of real closed fields can be stated as:

For a given standard prenex formula $\phi^* = (Q_{k+1}x_{k+1}) \cdots (Q_r x_r)\phi(x_1, \ldots, x_r)$, find a standard quantifier-free formula ψ^* such that ψ^* is equivalent to ϕ^*.

The top-level algorithm for solving the quantifier elimination problem is given in Algorithm 1. Step (2) of the algorithm needs some further description. The algorithm for computing a cylindrical algebraic decomposition, given the input $A = (A_1, \ldots, A_m)$ and k, proceeds in three phases: the projection phase, the base phase, and the extension phase. In the projection phase, if $r \geq 2$, a list PROJ(A) (projection of A) of polynomials in $r - 1$ variables is computed, such that (E) for every PROJ(A)-invariant cad D' of $\mathbb{R}^{r-1}$ there is an A-invariant cad D of $\mathbb{R}^r$ that induces D'. This projection process is applied to PROJ(A) recursively until univariate polynomials are reached. The property (E) can be achieved by letting PROJ(A) = PROJ$_1$(A) ∪ PROJ$_2$(A), where

ALGORITHM 1. Quantifier Elimination Algorithm

$\psi^* \leftarrow \mathrm{ELIM}(\phi^*)$[a]

```
1. Extract $k$ and the list $A = (A_1, \ldots, A_m)$ of distinct nonzero polynomials occurring in $\phi$ from $\phi^*$.
2. Apply the algorithm for cylindrical algebraic decomposition to $A$ and $k$, obtaining $s$, a cas for an $A$-invariant cad $D$ of $\mathbb{R}^r$, and $\psi$, a standard definition of the cad $D'$ of $\mathbb{R}^k$ induced by $D$ if $k > 0$ or ( ) if $k = 0$.
3. Construct $\psi^*$ from $\psi$ and $s$ by evaluating the polynomials in $A$ at the sample points in $s$.
```

[a] ϕ^* is a standard prenex formula $(Q_{k+1}x_{k+1}) \cdots (Q_r x_r)\phi(x_1, \ldots, x_r)$, $0 \leq k \leq r$, and ϕ is quantifier-free; ψ^* is a standard quantifier-free formula equivalent to ϕ^*.

$$\mathrm{PROJ}_1(A) = \bigcup_{1\le i\le m}\ \bigcup_{G_i\in \mathrm{RED}(A_i)} [\{\mathrm{ldcf}(G_i)\} \cup \mathrm{PSC}(G_i, G_i')]$$

$$\mathrm{PROJ}_2(A) = \bigcup_{1\le i<j\le m}\ \bigcup_{G_i\in \mathrm{RED}(A_i)\&G_j\in \mathrm{RED}(A_j)} \mathrm{PSC}(G_i, G_j).$$

Here ldcf(F), the leading coefficient of F, is the coefficient of x_r^k, where x_r^k is the highest power of x_r in F; red(F), the reductum of F, is $F - \mathrm{ldcf}(F)x_r^k$, $\mathrm{RED}(F) = \{\mathrm{red}^k(F) | 0 \le k \le \deg(F)\ \&\ \mathrm{red}^k(F) \neq 0\}$, where $\mathrm{red}^0(F) = F$, $\mathrm{red}^{k+1}(F) = \mathrm{red}[\mathrm{red}^k(F)]$, and $\mathrm{PSC}(F, G) = \{\mathrm{psc}_j(F, G) | 0 \le j \le \min[\deg(F), \deg(G)]\ \&\ \mathrm{psc}_j(F, G) \neq 0\}$, where $\mathrm{psc}_j(F, G)$ is the jth principal subresultant coefficient of F and G, that is, the coefficient of x_r^j in $S_j(F, G)$, the jth subresultant of F and G (see Section V), for integral polynomials F and G in the variables $x_1, \ldots, x_r$. In Arnon *et al.* (1984) the following theorem is proved:

> For a list A of integral polynomials in r variables, $r \ge 2$, if R is a PROJ(A)-invariant region in $\mathbb{R}^{r-1}$, then every polynomial A in A is either delineable on R [i.e., the set of real zeros of A in $Z(R)$ consists of finitely many disjoint sections of $Z(R)$] or identically zero on R, and for every $A_1, A_2 \in A$, any A_1-section and any A_2-section of $Z(R)$ are either disjoint or identical.

In the base phase, after $r - 1$ applications of PROJ, the real roots of the polynomials under consideration are isolated (see Section VI). These roots determine a $\mathrm{PROJ}^{r-1}(A)$-invariant cad D of $\mathbb{R}^1$, and algebraic sample points can be constructed. A standard definition of D is either ( ) if $k = 0$ or is determined by the signs of the polynomials on the cells of D.

In the extension phase, a cas for a $\mathrm{PROJ}^{r-l-1}(A)$-invariant cad D is constructed from the cas for a $\mathrm{PROJ}^{r-1}(A)$-invariant cad D', for $1 \le l \le r - 1$. This is done by isolating the real roots of the polynomials $B(a_1, \ldots, a_l, x_{l+1})$, where $B \in \mathrm{PROJ}^{r-l-1}(A)$ and $(a_1, \ldots, a_l)$ is a sample point for a cell in D'. These roots also allow one to extend the standard definition of D' to a standard definition of D. In order for the arithmetic to be exact, the computations are carried out in an algebraic extension of $\mathbb{Q}$. Algorithms for computing a primitive element γ of a multiple extension $\mathbb{Q}(\alpha, \beta)$ are used, so that all the arithmetic can be carried out in $\mathbb{Q}(\gamma)$ for some algebraic number γ (see Buchberger *et al.* (1983), chapter on computing in algebraic extensions). The algorithm for computing a cad is given in Algorithm 2.

ALGORITHM 2. Algorithm for Cylindrical Algebraic Decomposition

$(s, \psi) \leftarrow \mathrm{CAD}(A, k)$[a]

1. [$r = 1$]. If $r > 1$ then go to (2). Isolate the real roots $\alpha_1, \ldots, \alpha_n$ of the irreducible factors (see Section VI) of the nonzero elements of A. Construct a cas s for an A-invariant cad D. If $k = 0$ then set ψ to ( ). Otherwise, if $n = 0$, then set ψ to ("0 = 0"). Otherwise use the signs of the polynomials in A in the cells of D to construct a standard definition ψ of D. Exit.
2. [$r > 1$]. If $k = r$ then set k' to $k - 1$, otherwise set k' to k. Call CAD recursively with the input PROJ(A) and k', obtaining outputs s' and ψ'. Since s' is a cas for a PROJ(A)-invariant cad D', the real roots of A are delineable on each cell of D'. Let $s' = (s_1', \ldots, s_p')$, $s_j' = (s_{j,1}', \ldots, s_{j,r-1}')$. Construct a cas s for an A-invariant cad D of $\mathbb{R}^r$ by isolating the real roots of $A(s_{j,1}', \ldots, s_{j,r-1}', x_r)$ for every A $\in A$, $1 \le j \le p$. If $k < r$ then set ψ to ψ', otherwise use the real roots of the derivations of the polynomials $A(s_{j,1}', \ldots, s_{j,r-1}', x_r)$ to extend ψ' to a standard definition ψ of D. Rolle's theorem is essential here.

[a] A is a list of $m \ge 0$ integral polynomials in r variables, $0 \le k \le r$; s is a cas for some A-invariant cad D of $\mathbb{R}^r$; ψ is a standard definition of the cad D^* of $\mathbb{R}^k$-induced by D if $k > 0$; and ψ is the null list if $k = 0$.

Consider $(\exists y)(x^2 + y^2 - 4 < 0\ \&\ y^2 - 2x + 2 < 0)$ as an example for an input formula ϕ^* to the algorithm ELIM. In step (1), k is recognized to be 1 and $A = (y^2 + x^2 - 4, y^2 - 2x + 2)$. In step (2), the algorithm CAD is called with the inputs $A = (y^2 + x^2 - 4, y^2 - 2x + 2)$ and $k = 1$. The number of variables in A is 2, so step (2) in CAD is executed. PROJ(A) is $\{(-x^2 - 2x + 6)^2, (x^2 - 4)^2, (-2x + 2)^2, 4(x^2 - 4), 4(-2x + 2), 2, 1\}$. At this point it is realized that all the relevant information about the real roots of the polynomials in PROJ(A) can be gathered from the polynomials in $B = \{x^2 + 2x - 6, x^2 - 4, x - 1\}$. So CAD is called recursively with the inputs B and 1. Now $r = 1$ (base phase) and the real roots of the polynomials in B are $-1 - \sqrt{7} < -2 < 1 < -1 + \sqrt{7} < 2$. A cas for a B-invariant cad of $\mathbb{R}^1$ is $t = (-4, -1 - \sqrt{7}, -3, -2, 0, 1, \frac{3}{2}, -1 + \sqrt{7}, \frac{9}{5}, 2, 3)$ and a standard definition for this cad is

$\psi = (B_1 > 0 \;\&\; B_2 > 0 \;\&\; B_3 < 0,$ $B_1 = 0 \;\&\; B_2 > 0 \;\&\; B_3 < 0,$ $B_1 = 0 \;\&\; B_2 > 0 \;\&\; B_3 < 0,$
$B_1 < 0 \;\&\; B_2 = 0 \;\&\; B_3 < 0,$ $B_1 < 0 \;\&\; B_2 < 0 \;\&\; B_3 < 0,$ $B_1 < 0 \;\&\; B_2 < 0 \;\&\; B_3 = 0,$
$B_1 < 0 \;\&\; B_2 < 0 \;\&\; B_3 > 0,$ $B_1 = 0 \;\&\; B_2 < 0 \;\&\; B_3 > 0,$ $B_1 > 0 \;\&\; B_2 < 0 \;\&\; B_3 > 0,$
$B_1 > 0 \;\&\; B_2 = 0 \;\&\; B_3 > 0,$ $B_1 > 0 \;\&\; B_2 > 0 \;\&\; B_3 > 0).$

That finishes the recursive call of CAD.

The extension of t to a cas for an A-invariant cad of $\mathbb{R}^2$ yields $s = [(-4, 0), (-1 - \alpha, 0), (-3, 0), (-2, -1), (-2, 0), (-2, 1), (0, -3), (0, -2), (0, 0), (0, 2), (0, 3), (1, -2), (1, -\beta), (1, -1), (1, 0), (1, 1), (1, \beta), (1, 2), (\frac{3}{2}, -2), (\frac{3}{2}, -\alpha/2), (\frac{3}{2}, -\frac{6}{5}), (\frac{3}{2}, -1), (\frac{3}{2}, 0), (\frac{3}{2}, 1), (\frac{3}{2}, \frac{6}{5}), (\frac{3}{2}, \alpha/2), (\frac{3}{2}, 2), (\alpha, -2), (\alpha, -\gamma), (\alpha, 0), (\alpha, \gamma), (\alpha, 2), (\frac{9}{5}, -2), (\frac{9}{5}, -\delta), (\frac{9}{5}, -1), (\frac{9}{5}, -\varepsilon), (\frac{9}{5}, 0), (\frac{9}{5}, \varepsilon), (\frac{9}{5}, 1), (\frac{9}{5}, \delta), (\frac{9}{5}, 2), (2, -2), (2, -\zeta), (2, -1), (2, 0), (2, 1), (2, \zeta), (2, 2), (3, -3), (3, -2), (3, 0), (3, 2), (3, 3)]$, where $\alpha = \sqrt{7}$, $\beta = \sqrt{3}$, $\gamma = \sqrt{2}\sqrt{\sqrt{7} - 2}$, $\delta = 2\sqrt{2}/\sqrt{5}$, $\varepsilon = \sqrt{19}/5$, $\zeta = \sqrt{2}$ (see Fig. 3). Further, $k < r$, so ψ does not have to be extended. That finishes the call of CAD. In step (3) of the algorithm ELIM, the polynomials in A are evaluated at the sample points in s. Only the cylinders over the cells defined by ψ_7, ψ_8, and ψ_9 contain sample points that satisfy $A_1 < 0$ and $A_2 < 0$. Thus a standard quantifier-free formula equivalent to ϕ^* is $\psi^* = (\psi_7 \text{ or } \psi_8 \text{ or } \psi_9)$. In Fig. 1 the set of zeros of the polynomials in A is shown, the sample points for $\mathbb{R}^1$ are indicated as lines parallel to the y-axis and the sample points for $\mathbb{R}^2$ as points.

As is obvious from the above example, efficiency can be gained by combining neighboring cells, in which the polynomials involved have the same signs, into clusters of cells. Only one sample point is necessary for the whole cluster.

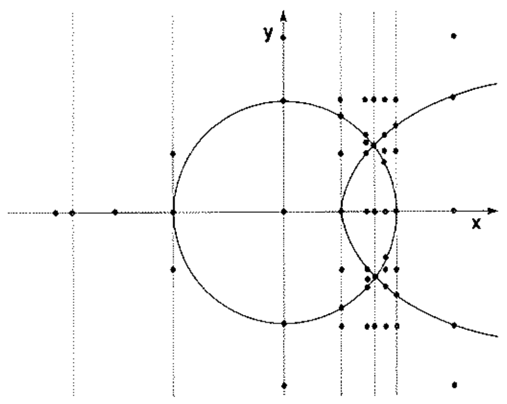

FIG. 1. The $(y^2 + x^2 - 4,\ y^2 - 2x + 2)$-invariant cad of $\mathbb{R}^2$.

Clustering algorithms are discussed in Arnon *et al.* (1984).

III. Simplification

A. Canonical Simplification and Decision of Equivalences

Let T be a decidable set of objects and $\sim$ an equivalence relation on T. The problem is to decide, for arbitrary a and b in T, whether $a \sim b$. Related to that is the problem of simplification of elements of T. If a canonical simplifier for $\sim$ were available, that is, a function S from T to T such that

$$S(a) \sim a \quad \text{and}$$
$$\text{if} \quad a \sim b \quad \text{then} \quad S(a) = S(b)$$

then one could decide $a \sim b$ by reducing a and b to their normal forms $S(a)$ and $S(b)$, respectively, and checking whether $S(a) = S(b)$.

This problem of deciding equivalences or simplifying elements appears in many situations. For instance, given the polynomial equations

$$\begin{aligned} 4x^2 + xy^2 - z + \tfrac{1}{4} &= 0 \\ 2x + y^2z + \tfrac{1}{2} &= 0 \\ -x^2z + \tfrac{1}{2}x + y^2 &= 0 \end{aligned} \tag{5}$$

one might be interested in simplifying the polynomial

$$f(x, y, z) = 2x + y^2 - \tfrac{9912}{497}z^6 + \tfrac{168}{497}z^5 - \tfrac{1043}{994}z^4 - \tfrac{129{,}493}{1988}z^3 - \tfrac{249{,}004}{1988}z^2 + \tfrac{1{,}067{,}573}{1988}z - \tfrac{153{,}426}{994}$$

Or, phrased as a decision problem, one might be interested in the question

$$f(x, y, z) = 0? \tag{6}$$

Let T be the free algebra of terms over a (countable) set of function symbols F and a (countable) set of variables V. Then the problem is to decide whether two terms are equal with respect to a given equational theory (i.e., a set of term equations) E. Two terms are equal if and only if they can be proved equal in the equational calculus, which assumes all the equations in E as axioms, and uses the rules of reflexivity, symmetry, transitivity, substitution, and replacement of subterms. As an example consider the term algebra T over the function symbols newq, add, app, with arity 0, 2, 2, respectively, and the countable set of variables $\{w, x, y, z, w_1, x_1, y_1, z_1, \ldots\}$. Let E be the equational theory

$$\begin{aligned} \text{app}(x, \text{newq}) &= x \\ \text{app}[x, \text{add}(y, z)] &= \text{add}[\text{app}(x, y), z] \qquad (7) \\ \text{app}[\text{app}(x, y), z] &= \text{app}[x, \text{app}(y, z)] \end{aligned}$$

Given this equational theory E, one might ask whether

$$\text{app}\{x, \text{app}[\text{add}(y, z), w]\} = \text{app}(\text{add}\{\text{app}[\text{app}(x, \text{newq}), y], z\}, w) \qquad (8)$$

In many cases (for instance in the ones above) one can associate a reduction relation $\rightarrow$ with the given equivalence $\sim$ so that $\leftrightarrow^*$ (the reflexive, symmetric, and transitive closure of $\rightarrow$) is equal to $\sim$. If, additionally, $\rightarrow$ has the termination property (i.e., there are no infinite chains of the form $t_1 \rightarrow t_2 \rightarrow t_3 \cdots$) and the Church–Rosser property (i.e., $t_1 \leftrightarrow^* t_2$ implies that there exists an s such that $t_1 \rightarrow {}^*s \leftarrow^* t_2$), then every $\sim$-equivalence class $[t]$ contains exactly one $\rightarrow$-irreducible element nf(t), the normal form of t. Then nf(t) is reached after finitely many applications of the reduction $\rightarrow$. Thus $t_1 \sim t_2$ if and only if nf(t_1) = nf(t_2), that is, $\sim$ can be decided by reducing t_1 and t_2 to normal forms and then testing for equality.

For checking the Church–Rosser property of a reduction relation $\rightarrow$, the following facts are useful:

a. $\rightarrow$ has the Church–Rosser property if and only if $\rightarrow$ is confluent (i.e., $t_1 \leftarrow^* s \rightarrow^* t_2$ implies that there is a u such that $t_1 \rightarrow^* u \leftarrow^* t_2$)

b. If $\rightarrow$ has the termination property, then $\rightarrow$ is confluent if and only if $\rightarrow$ is locally confluent (i.e., $t_1 \leftarrow s \rightarrow t_2$ implies that there is a u such that $t_1 \rightarrow^* u \leftarrow^* t_2$).

For a more detailed introduction to the decision problem for equational theories and simplification, the reader is referred to the chapter on simplification in Buchberger *et al.* (1983) and to Huet and Oppen (1980).

In the approach to the decision problem for $\sim$ that is presented here, one tries to construct a reduction relation $\rightarrow$ that has both the termination property and the Church–Rosser property.

B. Gröbner Bases and Canonical Simplification of Polynomials with Respect to Polynomial Side Relations

Let K be a field, $T = K[x_1, \ldots, x_n]$, the ring of polynomials over K in the indeterminates $x_1, \ldots, x_n$. Every set F of polynomials in $K[x_1, \ldots, x_n]$ generates an ideal id(F), the set of all polynomials of the form

$$h_1 \cdot f_1 + \cdots + h_m \cdot f_m$$

where m is an arbitrary nonnegative integer, $h_1, \ldots, h_m \in K[x_1, \ldots, x_n]$ and $f_1, \ldots, f_m \in F$. Then F is called a basis of the ideal. According to Hilbert's basis theorem, every ideal in $K[x_1, \ldots, x_n]$ has a finite basis, so without loss of generality one can restrict F to be finite.

Every basis $F = \{f_1, \ldots, f_m\}$ generates an equivalence relation $\sim_F : f \sim_F g$ if and only if $f - g \in \text{id}(F)$, or in other words,

$$f - g = h_1 \cdot f_1 + \cdots + h_m \cdot f_m \qquad (9)$$

where m is an arbitrary nonnegative integer, $h_1, \ldots, h_m \in K[x_1, \ldots, x_n]$. Obviously $\sim_F = \sim_{F'}$ if F and F' generate the same ideal. Of course, Eq. (9) could easily be solved or shown to be unsolvable if an upper bound for the coefficients $h_1, \ldots, h_m$ (say, for their degrees) were known. In 1926, G. Hermann computed such bounds. However, they turn out to be double exponential in the number of variables, which creates a nearly insurmountable problem in actual computation.

Alternatively, take the approach outlined in the previous section. As the first step, choose a linear ordering $>$ on the power products in $x_1, \ldots, x_n$ that satisfies the properties

$$x_1^0 \cdots x_n^0 < p \qquad \text{for all} \quad p \neq x_1^0 \cdots x_n^0$$

$$\text{if } p_1 < p_2 \text{ then } p_1 \cdot q < p_2 \cdot q \text{ for all } p_1, p_2, q$$

These two properties immediate imply that $>$ does not admit infinitely decreasing chains, that is, $>$ has the termination property. Using the ordering $>$ one can decompose every polynomial $f_i \in F$ into a leading term (consisting of a leading power product, lpp, and a leading coefficient, ldcf) and a reductum red,

$$f_i = \text{ldcf}(f_i) \cdot \text{lpp}(f_i) + \text{red}(f_i)$$

such that ldcf(f_i) $\neq 0$ and every power product that occurs in red(f_i)—that is, that has a coefficient different from 0 in red(f_i)—is smaller than

$\mathrm{lpp}(f_i)$ w.r.t. $>$. This leads to the rule system

$$\begin{aligned}\mathrm{lpp}(f_1) &\to -[1/\mathrm{ldcf}(f_1)] \cdot \mathrm{red}(f_1)\\ &\vdots\\ \mathrm{lpp}(f_m) &\to -[1/\mathrm{ldcf}(f_m)] \cdot \mathrm{red}(f_m)\end{aligned}$$

and to a reduction relation $\to_F$, where $f \to_F g$ if and only if there is an index i $(1 \le i \le m)$ such that a multiple of $\mathrm{lpp}(f_i)$, say $p \cdot \mathrm{lpp}(f_i)$, occurs in f with coefficient a and

$$g = f - a \cdot p \cdot \mathrm{lpp}(f_i) + a \cdot p \cdot [-(1/\mathrm{ldcf}(f_i)] \cdot \mathrm{red}(f_i)$$

So, whenever the left-hand side of one of these rules occurs in a polynomial f, then one can substitute the corresponding right-hand side for it. Further, $\leftrightarrow_F^* = \sim_F$, so if it can be established that $\to_F$ has the termination property and the Church–Rosser property, then $\sim_F$ can be decided by computing normal forms. In this special case the termination property is not a problem. In fact, $\to_F$ terminates for every ideal basis F (in every reduction step a term is replaced by lower terms w.r.t. a terminating order relation). A basis that has the Church–Rosser property is called a Gröbner basis. In 1965, B. Buchberger showed that

> F is a Gröbner basis if and only if for every critical pair (f, g) there exists an h such that $f \to_F^* h \leftarrow_F^* g$.

This theorem reduces the check for local confluence (which involves infinitely many test cases) to a confluence check on the finitely many critical pairs. A critical pair consists of the two reduction results of a term that can be reduced by two different rules. More precisely, let (1) $p_1 \to r_1$ and (2) $p_2 \to r_2$ be two rules derived from polynomials in F, let f be the result of reducing the least common multiple of p_1 and p_2 by rule (1), and let g be the result of reducing the least common multiple of p_1 and p_2 by rule (2); then, (f, g) is a critical pair of F.

Buchberger's theorem immediately leads to an algorithm for testing whether a given F is a Gröbner basis: for every one of the finitely many critical pairs (f, g), check whether $\mathrm{nf}(f) = \mathrm{nf}(g)$. If $\mathrm{nf}(f) \ne \mathrm{nf}(g)$ for some critical pair (f, g), then this inadequacy can be resolved by adding the polynomial $\mathrm{nf}(f) - \mathrm{nf}(g)$ to F. This procedure stops for all inputs F, and it is called the Gröbner basis algorithm, the completion algorithm for bases of polynomial ideals (see Algorithm 3).

Application of the Gröbner basis algorithm to the polynomials in Eq. (5) leads, after reducing the polynomials in the basis with respect to each other, to the following system of algebraic equations:

$$\begin{aligned}
&z^7 - \tfrac{1}{2}z^6 + \tfrac{1}{16}z^5 + \tfrac{13}{4}z^4 + \tfrac{75}{16}z^3 - \tfrac{171}{8}z^2 + \tfrac{133}{8}z - \tfrac{15}{4} = 0\\
&y^2 - \tfrac{191{,}886}{497}z^6 + \tfrac{318}{497}z^5 - \tfrac{4197}{1988}z^4 - \tfrac{251{,}555}{1988}z^3\\
&\qquad - \tfrac{481{,}837}{1988}z^2 + \tfrac{1{,}407{,}741}{1988}z - \tfrac{297{,}833}{994} = 0 \qquad (10)\\
&2x + \tfrac{9276}{497}z^6 - \tfrac{150}{497}z^5 + \tfrac{2111}{1988}z^4 + \tfrac{61{,}031}{944}z^3\\
&\qquad + \tfrac{232{,}833}{1988}z^2 - \tfrac{170{,}084}{497}z + \tfrac{144{,}407}{994} = 0
\end{aligned}$$

ALGORITHM 3. Gröbner Basis Algorithm

$G \leftarrow$ Gröbner basis(F)[a]

```
$G \leftarrow F$;
$C \leftarrow$ set of critical pairs of $G$;
while $C \ne \{\ \}$ do
  {choose $(f, g)$ from $C$;
  delete $(f, g)$ from $C$;
  if $\mathrm{nf}(f) \ne \mathrm{nf}(g)$
    then
      {add $\mathrm{nf}(f) - \mathrm{nf}(g)$ to $G$;
      increase $C$ by the critical pairs derived from
      $\mathrm{nf}(f) - \mathrm{nf}(g)$}};
```

[a] F and G are finite sets of polynomials, $\mathrm{id}(F) = \mathrm{id}(G)$, G is a Gröbner basis

So, indeed, Eq. (6) holds.

Besides solving the equivalence problem modulo a polynomial ideal, a Gröbner basis also provides solutions to a number of other problems, like determining whether an ideal is zero-dimensional, effective computation in the residue class ring modulo a polynomial ideal, and the word problem for commutative semigroups. In addition, if the ordering $>$ is lexicographic, then the construction of a Gröbner basis is a major step towards an exact solution of a system of algebraic equations. So, for instance, the variables in Eq. (10) are separated.

C. Complete Rewrite Rule Systems and Canonical Simplification of First-Order Terms with Respect to Equational Theories

An idea very similar to Gröbner basis construction can be used for attacking the decision problem for equational theories over first order terms. As a basis F for a polynomial ideal determines a set of pairs of polynomials f and g such that $f \sim_F g$, an equational theory E determines a set of pairs of terms s and t such that $s =_E t$. Again, the first step is to give a direction to the equations in E, usually by choosing a well-founded ordering $>$. For instance, from Eq. (7) one would get the rewrite rules

$$\begin{aligned} &\mathrm{app}(x, \mathrm{newq}) \to x \\ &\mathrm{app}[x, \mathrm{add}(y, z)] \to \mathrm{add}[\mathrm{app}(x, y), z] \quad (11) \\ &\mathrm{app}[\mathrm{app}(x, y), z] \to \mathrm{app}[x, \mathrm{app}(y, z)] \end{aligned}$$

which determine a reduction relation $\to_E$ on the first-order terms. This reduction relation satisfies $\leftrightarrow_E^* = =_E$. The goal is to complete the rewrite rule system by adding new rules that leave $\leftrightarrow_E^*$ invariant but transform $\to_E$ into a reduction relation that has the termination property and the Church–Rosser property. Proving that $\to_E$ has the termination property can be very complicated.

In 1967, D. E. Knuth and P. B. Bendix proved a theorem similar to Buchberger's theorem, namely,

> $\to_E$ is locally confluent if and only if for every critical pair (s_1, s_2) there exists a t such that $s_1 \to_E^* t \leftarrow_E^* s_2$.

The construction of critical pairs in this case is more complicated than in the polynomial case. Let (1) $l_1 \to r_1$ and (2) $l_2 \to r_2$ be two rewrite rules. Let t be a subterm of l_1 that is unifiable with l_2 (their variables might have to be renamed so that they become different), say by the unifier σ. Then a critical pair (s_1, s_2) results from reducing $\sigma(l_1)$ by rule (1) and by rule (2), respectively.

The theorem of Knuth and Bendix suggests the following algorithm for testing whether a terminating reduction relation $\to_E$ has the Church–Rosser property: for every one of the finitely many critical pairs (s, t) check whether $\mathrm{nf}(s) = \mathrm{nf}(t)$. If $\mathrm{nf}(s) \neq \mathrm{nf}(t)$ for some critical pair and the equation $\mathrm{nf}(s) =_E \mathrm{nf}(t)$ can be transformed into a rewrite rule $l \to r$ such that the termination property is preserved, then one can resolve this deficiency by adding the new rewrite rule $l \to r$ to the system. That is essentially the Knuth–Bendix completion procedure for rewrite rule systems over first-order terms. Contrary to the polynomial case, it might happen that a newly derived equation cannot be transformed into a rewrite rule, in which case the procedure stops unsuccessfully, or the completion procedure might run forever, generating infinitely many new rewrite rules. So a very important issue, which is still not resolved, is to determine whether the Knuth–Bendix procedure stops on a given equational theory E.

Given the equational theory of Eq. (7), the Knuth–Bendix procedure produces the rewrite rule system

$$\begin{aligned} &\mathrm{app}(x, \mathrm{newq}) \to x \\ &\mathrm{app}[x, \mathrm{add}(y, z)] \to \mathrm{add}[\mathrm{app}(x, y), z] \\ &\mathrm{app}[\mathrm{app}(x, y), z] \to \mathrm{app}[x, \mathrm{app}(y, z)] \\ &\mathrm{app}[x, \mathrm{app}(\mathrm{newq}, y)] \to \mathrm{app}(x, y) \\ &\mathrm{app}\{x, \mathrm{app}[\mathrm{add}(y, z), w]\} \to \\ &\qquad \mathrm{app}\{\mathrm{add}[\mathrm{app}(x, y), z], w\} \quad (12) \end{aligned}$$

So Eq. (8) holds modulo the equational theory E, since both sides reduce to the same term with respect to the complete rewrite rule system of Eq. (12).

D. Canonical Simplification in Other Domains

Rational expressions in $x_1, \ldots, x_n$ over an integral domain R represent elements in the quotient field of $R[x_1, \ldots, x_n]$. Two expressions f_1/g_1 and f_2/g_2 are equivalent (represent the same element in the quotient field) iff $f_1 \cdot g_2 = f_2 \cdot g_2$. If greatest common divisors can be computed in $R[x_1, \ldots, x_n]$, then there is a canonical simplifier S for this equivalence, which, for every rational expression, produces an equivalent expression whose numerator and denominator are relatively prime and whose denominator is a monic polynomial. The simplifier S uses rational arithmetic and the computation of greatest common divisors (see Sections V and VII).

Radical expressions are built from the rationals $\mathbb{Q}$ together with the variables $x_1, \ldots, x_n$, function symbols $+, -, \cdot, /$, and radicals, that is, rational powers s^r, where s is a radical expression and $r \in \mathbb{Q}$ (s in s^r is called a radicand). For two radical expressions s and t, $s \sim t$ iff s and t denote the same meromorphic function. Caviness and Fateman have developed a canonical simplifier for unnested radical expressions, an unnested radical expression being an expression s such that no radicand in s contains another radical. Factorization of polynomials plays a major role in the simplification algorithm. Basically, an extension field K of $\mathbb{Q}$ is constructed such that the expression s can be viewed as denoting an element of K. In a second step, s is simplified using rational arithmetic in K.

We have mentioned a few examples of domains that allow canonical simplification. In general, however, domains with canonical simplifiers are quite rare. An example of a domain for which no canonical simplifier exists is the class $R2$, which consists of terms built from the rationals $\mathbb{Q}$, the variable x, π, and the function symbols $+$, $\cdot$, sin, and abs. Two expressions s and t in $R2$ are equivalent iff s and t denote the same function on $\mathbb{R}$.

E. Noncanonical Simplification

Canonical simplification may not be possible in the domain to which an expression belongs, or the emphasis may not be put on using simplification for deciding equivalences, but on generating "intelligible" output. For situations like that, most of the currently available computer algebra systems give the user the freedom to choose from a number of simplification procedures the one which is most appropriate for the problem at hand, or even to teach the system the kind of simplification he or she wants. Substitution and pattern matching are two widely used techniques in such noncanonical simplifiers.

IV. Computational Group Theory

Computational group theory probably started with Todd and Coxeter's (1936) method for coset enumeration. By now, it has evolved into a sizable collection of algorithms and program systems of its own. These methods and program systems are in most cases quite distinct from those used in other parts of computer algebra. It is therefore not feasible to present an overview in this limited space. The interested reader may wish to consult Atkinson (1984) and the chapter on computing with groups and their character tables in Buchberger *et al.* (1983) for algorithms dealing with finitely presented groups, permutation groups, matrix groups, determination of the character table of finite group, and computation of Galois groups.

The following example of the Todd–Coxeter method for determining the cardinality of a group given by a finite presentation should give the flavor of the computational methods used in group theory. Let $\{A, B \mid ABA = 1, BAB = 1\}$ be a finite presentation of a group G, that is, G is the quotient of F by R, where F is the free group generated by A and B, and R is the congruence generated by the relations $ABA = 1$ and $BAB = 1$. The goal is to enumerate the elements of G and also produce a multiplication table for G. Various tables are used in the process: a multiplication table, and a relator table for each of the relators. The rows in the relator tables show the effect of multiplication of each element by the relator, the overall effect being identity. To get started, one defines $2 = 1A$. This information, together with $2A^{-1} = 1$, is stored in the multiplication table and the according entries are also made in the relator tables, leading to the multiplication table:

	A	A^{-1}	B	B^{-1}
1	**2**			
2		1		

and the relator tables:

$$\begin{array}{ccccccc} & A & & B & & A & \\ 1 & & 2 & & & & 1 \\ 2 & & & & 1 & & 2 \end{array} \qquad \begin{array}{ccccccc} & B & & A & & B & \\ 1 & & & & & & 1 \\ 2 & & & & & & 2 \end{array}$$

No further entries can be made into the tables. So the next definition $3 = 1A^{-1}$ is introduced. Entering this information into the tables causes the first row in the relator table for ABA to be closed, producing the deduction $2B = 3$, which is denoted by the subscript 1. This deduction is recorded in the multiplication tables and the relator tables, leading to the deduction $1B = 2$. Finally all the tables close and the procedure stops, with the multiplication table:

	A	A^{-1}	B	B^{-1}
1	**2**	**3**	2	3
2	3	1	3	1
3	1	2	1	2

and the relator tables:

$$\begin{array}{ccccccc} & A & & B & & A & \\ 1 & & 2 & & 3 & & 1 \\ & & & {}_1 & & & \\ 2 & & 3 & & 1 & & 2 \\ & & & {}_4 & & & \\ 3 & & 1 & & 2 & & 3 \\ & & & & & {}_3 & \end{array} \qquad \begin{array}{ccccccc} & B & & A & & B & \\ 1 & & 2 & & 3 & & 1 \\ 2 & & 3 & & 1 & & 2 \\ & & & & & {}_2 & \\ 3 & & 1 & & 2 & & 3 \end{array}$$

So the group G has cardinality 3.

It has been recently recognized that completion procedures such as the ones discussed in Section III can be used instead of the Todd–Coxeter method. The completion procedure terminates whenever the given group is finite, but also in some other cases.

V. Polynomial Greatest Common Divisors

For many of the problems in computer algebra—for instance, indefinite integration, indefinite summation, or cylindrical algebraic decomposition—it is essential to be able to compute the greatest common divisor (gcd) of multivariate integral polynomials. The ring $\mathbb{Z}[x_1, \ldots, x_\nu]$ does not allow division with quotient and remainder, and it is not a Euclidean ring. That means no grading function deg exists such that for every A and every nonzero B, A can be written as $A = QB + R$ and $R = 0$ or $\deg(R) < \deg(B)$. However, letting deg denote the degree in the main variable x_ν, for every A and every nonzero B with $m = \deg(A) \geq n = \deg(B)$, it is possible to find polynomials Q and R such that

$$\mathrm{ldcf}(B)^{m-n+1}A = QB + R$$
$$\text{and} \qquad R = 0 \qquad \text{or} \qquad \deg(R) < \deg(B)$$

(see the algorithm for pseudo-division in Section VII). Q is the pseudo-quotient, pquot, and R the pseudo-remainder, prem, of A and B.

Throughout this section the isomorphism $\mathbb{Z}[x_1, \ldots, x_\nu] \sim (\mathbb{Z}[x_1, \ldots, x_{\nu-1}])[x_\nu]$ is used and polynomials in $\mathbb{Z}[x_1, \ldots, x_\nu]$ are considered as polynomials in the main variable x_ν with coefficients in $\mathbb{Z}[x_1, \ldots, x_{\nu-1}]$.

A. Polynomial Remainder Sequences

For an arbitrary unique factorization domain U, two nonzero polynomials $A(x)$ and $B(x)$ in $U[x]$ are called similar, $A(x) \sim B(x)$, if there are nonzero coefficients of similarity a and b in U such that $aA(x) = bB(x)$. A polynomial remainder sequence (prs) is a sequence $F_1, F_2, \ldots, F_k$ of nonzero polynomials in $U[x]$, where $k \geq 2$, $\deg(F_1) \geq \deg(F_2)$, $F_i \sim \mathrm{prem}(F_{i-2}, F_{i-1})$ for $3 \leq i \leq k$, and $\mathrm{prem}(F_{k-1}, F_k) = 0$. Pseudo-division yields a unique result, and this implies that the prs beginning with F_1 and F_2 is unique up to similarity. Because of $\gcd(F_1, F_2) \sim \gcd(F_2, F_3) \sim \cdots \sim \gcd(F_{k-1}, F_k) \sim F_k$, the last element in a prs is equal to $\gcd(F_1, F_2)$ to within similarity.

A polynomial $A \in U[x]$ is primitive if its coefficients are relatively prime. Every $A \in U[x]$ can be written uniquely (up to multiplication by unit elements) as $A = \mathrm{cont}(A) \cdot \mathrm{pp}(A)$, where $\mathrm{cont}(A)$, the content of A, is the gcd of the coefficients of A, and $\mathrm{pp}(A)$, the primitive part of A, is a primitive polynomial.

So the gcd of two polynomials A and B in $U[x]$ is the product of $\gcd[\mathrm{cont}(A), \mathrm{cont}(B)]$ and the primitive part of the last element of a prs starting with $\mathrm{pp}(A)$ and $\mathrm{pp}(B)$. An algorithm is given in Alg. 4. The efficiency of step (2) in PRSGCD crucially depends on the method used for computing a prs. Choosing $F_i = \mathrm{prem}(F_{i-2}, F_{i-1})$ leads to the Euclidean prs algorithm. This algorithm suffers from an exponential coefficient growth and is therefore impractical even for univariate polynomials over $\mathbb{Z}$. As an example, take $F_1 = x^8 + x^6 - 3x^4 - 3x^3 + 8x^2 + 2x - 5$, $F_2 = 3x^6 + 5x^4 - 4x^2 - 9x + 21$. The Euclidean prs algorithm produces the prs $F_1, F_2, F_3 = -15x^4 + 3x^2 - 9$, $F_4 = -5265x^2 - 10{,}125x + 19{,}845$, $F_5 = 114{,}727{,}286{,}250x - 151{,}313{,}062{,}500$, $F_6 = 1{,}426{,}857{,}992{,}535{,}543{,}750$. So $\gcd(F_1, F_2) = 1$. The coefficients in the inputs as well as the result are very small, yet the intermediary results show an enormous coefficient growth. The effect is much worse for multivariate polynomials.

In the primitive prs algorithm, one chooses $F_i = \mathrm{pp}[\mathrm{prem}(F_{i-2}, F_{i-1})]$. This produces the prs $F_1, F_2, F_3 = 5x^4 - x^2 + 3$, $F_4 = 13x^2 + 25x - 49$, $F_5 = 4663x - 6150$, $F_6 = 1$. The coefficients are kept as small as possible, but the price that has to be paid is the gcd computations in the coefficient domain, which may be very time-consuming.

Exploiting the advantages and avoiding the limitations of both the Euclidean and the primitive prs algorithm, one will try to factor out as much as possible of the content without actually computing the content, that is, one chooses $\beta_i \cdot F_i = \mathrm{prem}(F_{i-2}, F_{i-1})$ for a suitable β_i. The best algorithm along these lines is Collins's subresultant prs algorithm. Some notation is necessary for stating the algorithm. Let $A(x) = a_m x^m + \cdots + a_1 x + a_0$ and $B(x) = b_n x^n + \cdots + b_1 x + b_0$ be

ALGORITHM 4. Algorithm for Computing the gcd of Polynomials in $U[x]$ by Polynomial Remainder Sequences

```
G ← PRSGCD(F_1, F_2)^a

1. Compute $d = \gcd[\mathrm{cont}(F_1), \mathrm{cont}(F_2)]$.
2. Compute a prs $F_1' = \mathrm{pp}(F_1)$, $F_2' = \mathrm{pp}(F_2)$, ..., $F_k'$.
3. Set $G$ to $d \cdot \mathrm{pp}(F_k')$.
```

[a] F_1, F_2 are nonzero polynomials in $U[x]$, $\deg(F_1) \geq \deg(F_2)$, and $G = \gcd(F_1, F_2)$.

two nonzero polynomials in $U[x]$ with $m = \deg(A) \geq n = \deg(B)$. Let $M(A, B)$ be the Sylvester matrix of A and B, such that

$$M(A, B) = \left[\begin{array}{ccccccccc} a_m & a_{m-1} & & \cdots & a_1 & a_0 & 0 & \cdots & 0 \\ 0 & a_m & a_{m-1} & & \cdots & a_1 & a_0 & \cdots & 0 \\ & & & & \vdots & & & & \\ 0 & \cdots & 0 & a_m & a_{m-1} & & \cdots & a_1 & a_0 \\ \hline b_n & b_{n-1} & \cdots & b_1 & b_0 & 0 & & \cdots & 0 \\ 0 & b_n & b_{n-1} & \cdots & b_1 & b_0 & & \cdots & 0 \\ & & & & \vdots & & & & \\ 0 & \cdots & 0 & b_n & b_{n-1} & & \cdots & b_1 & b_0 \end{array}\right] \begin{array}{l} \left.\vphantom{\begin{array}{c}a\\a\\a\\a\end{array}}\right] n \text{ rows} \\ \left.\vphantom{\begin{array}{c}a\\a\\a\\a\end{array}}\right] m \text{ rows} \end{array}$$

Let $M(A, B)_{i,j}$ be the matrix that results from $M(A, B)$ by eliminating the last j rows of coefficients of A, the last j rows of coefficients of B, and the last $2j + 1$ columns except the $(m + n - i - j)$-th, for $0 \leq i \leq j \leq n - 1$. The jth subresultant of A and B is the polynomial

$$S_j(A, B) = \sum_{i=0}^{j} \det[M(A, B)_{i,j}] \cdot x^i$$

for $0 \leq j \leq n - 1$. The zeroth subresultant is what is usually called the resultant of A and B.

Let F_1, F_2 be two nonzero polynomials in $U[x]$, and $F_1, F_2, \ldots, F_k$ a prs in $U[x]$. Let n_i be the degree of F_i, for $1 \leq i \leq k$, and $\delta_i = n_i - n_{i+1}$, for $1 \leq i \leq k - 1$. The fundamental theorem of subresultants states that there are γ_i and θ_i, $1 \leq i \leq k$, in the quotient field of U such that for $i = 3, \ldots, k$, $S_{n_{i-1}-1}(F_1, F_2) = \gamma_i F_i$, $S_j(F_1, F_2) = 0$ for $n_{i-1} - 1 > j > n_i$, $S_{n_i}(F_1, F_2) = \theta_i F_i$, and $S_j(F_1, F_2) = 0$ for $n_k > j \geq 0$. In order to get $\gamma_i = 1$, it suffices to choose

$$\beta_3 = (-1)^{\delta_1+1},$$

$$\beta_i = (-1)^{\delta_{i-2}+1} \operatorname{ldcf}(F_{i-2}) h_{i-2}^{\delta_{i-2}} \qquad \text{for } i = 4, \ldots, k$$

where

$$h_2 = \operatorname{ldcf}(F_2)^{\delta_1}$$

$$h_i = \operatorname{dcf}(F_i)^{\delta_{i-1}} h_{i-1}^{1-\delta_{i-1}} \qquad \text{for } i = 3, \ldots, k - 2$$

A proof of the fundamental theorem of subresultants and a detailed introduction to the theory of subresultants can be found in the chapter on generalized polynomial remainder sequences in Buchberger *et al.* (1983). Starting with the polynomials $F_1 = x^8 + x^6 - 3x^4 - 3x^3 + 8x^2 + 2x - 5$, $F_2 = 3x^6 + 5x^4 - 4x^2 - 9x + 21$, the subresultant prs algorithm produces the prs F_1, F_2, $F_3 = -15x^4 + 3x^2 - 9$, $F_4 = -65x^2 - 125x + 245$, $F_5 = 9326x - 12{,}300$, $F_6 = 260{,}708$.

B. Modular GCD Algorithms

In 1971 W. S. Brown and G. E. Collins independently discovered a modular algorithm for computing the gcd of polynomials in $\mathbb{Z}[x_1, \ldots, x_\nu]$. In the following, Brown's version of the modular algorithm for the univariate case is described.

Algorithm 5 shows how the gcd computation for arbitrary polynomials in $\mathbb{Z}[x]$ can be reduced to the gcd computation for primitive polynomials. So let F_1, F_2 be primitive polynomials in $\mathbb{Z}[x]$. Let G, H_1, H_2 denote the gcd and the cofactors of F_1 and F_2. Furthermore let f_1, f_2, g, h_1, h_2 be the leading coefficients of F_1, F_2, G, H_1, H_2, respectively. Setting $\mathbf{g} = \gcd(f_1, f_2)$, $\mathbf{G} = (\mathbf{g}/g) \cdot G$, $\mathbf{F}_1 = \mathbf{g} \cdot F_1$, $\mathbf{F}_2 = \mathbf{g} \cdot F_2$, $\mathbf{H}_1 = g \cdot H_1$, $\mathbf{H}_2 = g \cdot H_2$, implies that $\mathbf{F}_1 = \mathbf{G} \cdot \mathbf{H}_1$, $\mathbf{F}_2 = \mathbf{G} \cdot \mathbf{H}_2$, and furthermore $\operatorname{pp}(\mathbf{G}) = G$, $\operatorname{ldcf}(\mathbf{G}) = \mathbf{g}$, $\operatorname{ldcf}(\mathbf{H}_1) = g \cdot h_1 = f_1$, $\operatorname{ldcf}(\mathbf{H}_2) = g \cdot h_2 = f_2$. Then f_1, f_2, and $\mathbf{g}$ can be computed at the beginning, whereas g can be determined

ALGORITHM 5. Reduction of the gcd Computation for Integral Polynomials to the gcd Computation for Primitive Integral Polynomials

```
$(G', H'_1, H'_2) \leftarrow \mathrm{GCD}(F'_1, F'_2)^a$

1. Compute $c_1 = \mathrm{cont}(F'_1)$, $c_2 = \mathrm{cont}(F'_1)$, and
   $c = \gcd(c_1, c_2)$.
2. Set $F_1 \leftarrow F'_1/c_1$, $F_2 \leftarrow F'_2/c_2$.
3. Compute the gcd $G$ of the polynomials $F_1$ and
   $F_2$ and the corresponding cofactors $H_1$ and $H_2$.
4. $G' \leftarrow c \cdot G$, $H'_1 \leftarrow (c_1/c) \cdot H_1$,
   $H'_2 \leftarrow (c_2/c) \cdot H_2$.
```

[a] F'_1 and F'_2 are polynomials in $\mathbb{Z}[x]$; $G' = \gcd(F'_1, F'_2)$; H'_1 and H'_2 are the cofactors, i.e., $F'_1 = H'_1 \cdot G'$, $H'_2 \cdot G'$.

only at the end of the computation. Further, $d = \deg(G)$ is unknown at the beginning of the computation, but obviously $d \leq \min[\deg(F_1), \deg(F_2)]$.

Now a prime p_1 is chosen that divides neither f_1 nor f_2. Let $g^{\sim(1)} = \mathbf{g} \bmod p_1$. The polynomials $F_1^{\sim(1)} = \mathbf{F}_1 \bmod p_1$, $F_2^{\sim(1)} = \mathbf{F}_2 \bmod p_1$, $G^{\sim(1)} = g^{\sim(1)} \cdot \gcd(F_1^{\sim(1)}, F_2^{\sim(1)})$, $H_1^{\sim(1)} = F_1^{\sim(1)}/G^{\sim(1)}$, $H_2^{\sim(1)} = F_2^{\sim(1)}/G^{\sim(1)}$ in $\mathbb{Z}_{p_1}[x]$ are computed using a gcd algorithm for $\mathbb{Z}_{p_1}[x]$. It is assumed that this gcd algorithm returns a monic gcd (leading coefficient is 1). So $\text{ldcf}(G^{\sim(1)}) = g^{\sim(1)}$. Since G divides F_1 and F_2 in $\mathbb{Z}[x]$, G_{p_1} divides $F_1^{\sim(1)}$ and $F_2^{\sim(1)}$ and therefore G_{p_1} divides $G^{\sim(1)}$ in $\mathbb{Z}_{p_1}[x]$, where $G_{p_1} = G \bmod p_1$. Furthermore, $\deg(G_{p_1}) = \deg(G)$, because p_1 neither divides f_1 nor f_2 and thus also does not divide g. So $\deg(G^{\sim(1)}) \geq \deg(G_{p_1}) = \deg(G) = d$. If, additionally, $\deg(G^{\sim(1)}) = d$, then $G^{\sim(1)} = a \cdot (\mathbf{G} \bmod p_1)$ for some $a \in \mathbb{Z}_{p_1}$. But $a = 1$ because of $\text{ldcf}(\mathbf{G} \bmod p_1) = g^{\sim(1)}$, and therefore $G^{\sim(1)} \equiv \mathbf{G} \bmod p_1$ and $H_1^{\sim(1)} \equiv \mathbf{H}_1 \bmod p_1$, $H_2^{\sim(1)} \equiv \mathbf{H}_2 \bmod p_1$.

This process is repeated for several primes $p_2, \ldots, p_n$. Only those primes are kept for which $\deg(G^{\sim(i)})$ is minimal. Instead of storing all the quadruples $(p_i, G^{\sim(i)}, H_1^{\sim(i)}, H_2^{\sim(i)})$, the Chinese remainder algorithm (see Section VII) is used to compute the uniquely determined polynomials G^*, H_1^*, H_2^* with coefficients smaller than $q/2$ such that $G^* \equiv G^{\sim(i)} \bmod p_i$, $H_1^* \equiv H_1^{\sim(i)} \bmod p_i$, $H_2^* \equiv H_2^{\sim(i)} \bmod p_i$, for all $1 \leq i \leq n$, where $q = p_1 \cdot \cdots \cdot p_n$. Those primes p_i, which neither divide f_1 nor f_2 but for which $\deg(G^{\sim(i)}) > d$, are called unlucky primes. Brown proves that every unlucky prime is a divisor of a certain subresultant of F_1 and F_2, so there are only finitely many of them.

As soon as $\deg(G^{\sim(1)}) = \cdots = \deg(G^{\sim(n)}) = d$ is reached, one has $G^* \equiv \mathbf{G} \bmod q$, $H_1^* \equiv \mathbf{H}_1 \bmod q$, $H_2^* \equiv \mathbf{H}_2 \bmod q$. If $q > \max(\mu^*, \mu)$, where $\mu^*/2$ is an upper bound for the absolute values of the coefficients of $G^* \cdot H_1^*$ and $G^* \cdot H_2^*$ and $\mu/2$ is an upper bound for the coefficients of $\mathbf{g} \cdot F_1$ and $\mathbf{g} \cdot F_2$, then $G^* \cdot H_1^* \equiv \mathbf{F}_1 \bmod q$ and $q/2$ is an upper bound for the absolute values of the coefficients of $G^* \cdot H_1^*$ and $\mathbf{F}_1$. Therefore $G^* \cdot H_1^* = \mathbf{F}_1$. Under the same conditions one gets $G^* \cdot H_2^* = \mathbf{F}_2$. So $G^* = \mathbf{G}$, $H_1^* = \mathbf{H}_1$, $H_2^* = \mathbf{H}_2$. The final results are $G = \text{pp}(\mathbf{G})$, $H_1 = \mathbf{H}_1/g$, $H_2 = \mathbf{H}_2/g$. The algorithm is given in Algorithm 6.

The following example illustrates the modular gcd algorithm. Consider the two polynomials $F_1 = x^5 - x^4 - 3x^2 - 3x + 2$, $F_2 = x^4 - 2x^3 - 3x^2 + 4x + 4$. So $f_1 = f_2 = \mathbf{g} = 1$, $e = \min[\deg(F_1), \deg(F_2)] = 4$, $\mu = 2 \cdot 1 \cdot 4 = 8$.

ALGORITHM 6. Modular gcd Algorithm for Univariate Primitive Integral Polynomials

$(G, H_1, H_2) \leftarrow \text{MODGCD}(F_1, F_2)$[a]

```
1. $f_1 \leftarrow \text{ldcf}(F_1)$, $f_2 \leftarrow \text{ldcf}(F_2)$, $\mathbf{g} \leftarrow \gcd(f_1, f_2)$.
2. $n \leftarrow 0$, $e \leftarrow \min[\deg(F_1), \deg(F_2)]$.
3. $\mu \leftarrow 2\,\mathbf{g} \cdot \max\{|\phi| \mid \phi$ is a coefficient of $F_1$ or $F_2\}$.
4. Choose a new prime $p$ that does not divide $f_1$ and $f_2$.
5. $g^{\sim} \leftarrow \mathbf{g} \bmod p$, $F_1^{\sim} \leftarrow g^{\sim}F_1 \bmod p$, $F_2^{\sim} \leftarrow g^{\sim}F_2 \bmod p$.
6. Compute $\gcd(F_1^{\sim}, F_2^{\sim})$ and set $G^{\sim} \leftarrow g^{\sim} \cdot \gcd(F_1^{\sim}, F_2^{\sim})$, $H_1^{\sim} \leftarrow F_1^{\sim}/G^{\sim}$, $H_2^{\sim} \leftarrow F_2^{\sim}/G^{\sim}$. All the computations are done in $\mathbb{Z}_p[x]$.
7. If $\deg(G^{\sim}) = 0$ then set $G \leftarrow 1$, $H_1 \leftarrow F_1$, $H_2 \leftarrow F_2$ and exit. If $\deg(G^{\sim}) > e$ then go to (4). If $\deg(G^{\sim}) < e$ then set $n \leftarrow 0$, $e \leftarrow \deg(G^{\sim})$.
8. $n \leftarrow n + 1$.
9. If $n = 1$ then set $q \leftarrow p$, $G^* \leftarrow G^{\sim}$, $H_1^* \leftarrow H_1^{\sim}$, $H_2^* \leftarrow H_2^{\sim}$. Otherwise use the Chinese remainder algorithm with the moduli $q$ and $p$ in order to extend $(q, G^*, H_1^*, H_2^*)$ by $(p, G^{\sim}, H_1^{\sim}, H_2^{\sim})$.
10. If $q < \mu$ then go to (4). Choose $\mu^*/2$ to be an upper bound for the absolute values of the coefficients of $G^*H_1^*$ and $G^*H_2^*$. If $q < \mu^*$ then go to (4).
11. $G \leftarrow \text{pp}(G^*)$, $g \leftarrow \text{ldcf}(G)$, $H_1 \leftarrow H_1^*/g$, $H_2 \leftarrow H_2^*/g$.
```

[a] F and F_2 are primitive polynomials in $\mathbb{Z}[x]$; $G = \gcd(F_1, F_2)$, $F_1 = H_1 \cdot G$, $F_2 = H_2 \cdot G$.

As the first prime, choose $p = 2$. $F_1^{\sim} = x^5 + x^4 + x^2 + x$, $F_2^{\sim} = x^4 + x^2$. Then $\gcd(F_1^{\sim}, F_2^{\sim}) = x^3 + x$. Therefore $G^{\sim} = x^3 + x$, $H_1^{\sim} = F_1^{\sim}/G^{\sim} = x^2 + x + 1$, $H_2^{\sim} = F_2^{\sim}/G^{\sim} = x$, $e = 3$, $q = 2$, $G^* = x^3 + x$, $H_1^* = x^2 + x + 1$, $H_2^* = x$. Since q is less than μ, choose the next prime $p = 3$: $F_1^{\sim} = x^5 - x^4 - 1$, $F_2^{\sim} = x^4 + x^3 + x + 1$. Then $\gcd(F_1^{\sim}, F_2^{\sim}) = x^2 - x + 1$. Therefore $G^{\sim} = x^2 - x + 1$, $H_1^{\sim} = F_1^{\sim}/G^{\sim} = x^3 - x - 1$, $H_2^{\sim} = F_2^{\sim}/G^{\sim} = x^2 - x + 1$. $\deg(G^{\sim}) < e$, so the prime 2 is eliminated and $e = 2$, $q = 3$, $G^* = x^2 - x + 1$, $H_1^* = x^3 - x - 1$, $H_2^* = x^2 - x + 1$. Since q is less than μ, choose the next prime $p = 5$: $F_1^{\sim} = x^5 - x^4 + 2x^2 + 2x + 2$, $F_2^{\sim} = x^4 - 2x^3 + 2x^2 - x - 1$. Then $\gcd(F_1^{\sim}, F_2^{\sim}) = x^2 - x - 2$. Therefore $G^{\sim} = x^2 - x - 2$, $H_1^{\sim} = F_1^{\sim}/G^{\sim} = x^3 + 2x - 1$, $H_2^{\sim} = F_2^{\sim}/G^{\sim} = x^2 - x - 2$. Extending (q, G^*, H_1^*, H_2^*) by $(p, G^{\sim}, H_1^{\sim}, H_2^{\sim})$ yields $(q, G^*, H_1^*, H_2^*) = (15, x^2 - x - 2, x^3 + 2x - 1, x^2 - x - 2)$. Now $q \geq \mu$ and $q \geq \mu^*$, so the result is $G = \text{pp}(G^*) = x^2 - x - 2$, $H_1 = x^3 + 2x - 1$, $H_2 = x^2 - x - 2$.

The algorithm described in this section can be generalized to a gcd algorithm for multivariate polynomials over $\mathbb{Z}$. The polynomials are considered as polynomials in the main variable x_ν with coefficients in $\mathbb{Z}[x_1, \ldots, x_{\nu-1}]$. Instead of prime numbers p one has to use irreducible polynomials in $\mathbb{Z}[x_1, \ldots, x_{\nu-1}]$. The algorithm MODGD and its generalization to multivariate polynomials do not take advantage of the sparseness of the input polynomials. Algorithms that take sparseness into account are Moses and Yun's EZ-GCD algorithm, which is based on p-adic arithmetic and Hensel lifting, and Zippel's probabilistic algorithm, which is currently the default gcd algorithm used in Macsyma.

VI. Factorization and Real Root Isolation

A. Factorization of Univariate Polynomals over a Finite Field

This section deals with the factorization of a polynomial $A(x)$ in $\mathbb{Z}_p[x]$, where p is a prime number. In a first step the factorization of $A(x)$ is reduced to the factorization of square-free polynomials, a square-free polynomial being a polynomial with no factor of multiplicity greater than 1. Let $A(x)$ be a nonzero polynomial in $\mathbb{Z}_p[x]$. If $A(x)$ has a factor of multiplicity greater than 1, then it can be written as

$$A(x) = B(x)^2C(x).$$

The derivation of $A(x)$ is

$$A'(x) = 2B(x)B'(x)C(x) + B(x)^2C'(x)$$

so $A(x)$ and $A'(x)$ have a nonconstant greatest common divisor. On the other hand, if $A(x)$ is square-free,

$$A(x) = \prod_{i=1}^{r} A_i(x)$$

where the $A_i(x)$, $1 \le i \le r$, are pairwise different prime factors, then

$$A'(x) = A_1'(x) \prod_{i=2}^{r} A_i(x) + \cdots + A_r'(x) \prod_{i=1}^{r-1} A_i(x)$$

so $A(x)$ and $A'(x)$ are relatively prime. Thus a nonzero polynomial $A(x)$ in $\mathbb{Z}_p[x]$ is square-free if and only if $\gcd[A(x), A'(x)] = 1$. Using this fact, one first computes $D(x) = \gcd[A(x), A'(x)]$. If $D(x) = 1$, then $A(x)$ is already square-free. If $D(x)$ is a proper factor of $A(x)$, then the factorization of $A(x)$ reduces to the factorization of $D(x)$ and $A(x)/D(x)$. Finally, if $D(x) = A(x)$, then $A'(x) = 0$, that is, the coefficient a_k of x^k in $A(x)$ is different from 0 only if k is a multiple of p. So, using the basic arithmetic in $\mathbb{Z}_p$, $A(x)$ can be written as $A(x) = B(x^p) = [B(x)]^p$. The factorization of $A(x)$ reduces to the factorization of $B(x)$. Iterating this process sufficiently often will lead to square-free polynomials in $\mathbb{Z}_p[x]$.

The problem of factoring square-free polynomials is solved by Berlekamp's algorithm. Let $A(x)$ be a square-free polynomial in $\mathbb{Z}_p[x]$ of degree n, and let r be the number of prime factors of $A(x)$, $A(x) = P_1(x) \cdots P_r(x)$. By the Chinese remainder theorem for polynomials (Section VII), for every r-tuple $(s_1, \ldots, s_r)$ in $\mathbb{Z}_p$ there exists a uniquely determined polynomial $V(x)$ in $\mathbb{Z}_p[x]$ such that

$$(*) \qquad V(x) \equiv s_i[\text{mod } P_i(x)]$$
$$\text{for} \quad 1 \le i \le r, \qquad \text{and}$$
$$\deg(V) < \deg(P_1) + \cdots + \deg(P_r) = \deg(A)$$

If $r \ge 2$ and $s_1 \ne s_2$, then $P_1(x)$ is a divisor of $\gcd[A(x), V(x) - s_1]$, but $P_2(x)$ is not. If $V(x)$ satisfies the condition (*) then $V(x)^p \equiv s_i^p = s_i \equiv V(x)$ [mod $P_i(x)$] for $1 \le i \le r$, so $V(x)$ satisfies

$$(**) \quad V(x)^p \equiv V(x) \ [\text{mod } A(x)] \text{ and } \deg(V) < \deg(A)$$

Every solution of (*) for some $(s_1, \ldots, s_r)$ is also a solution of (**). On the other hand, $V(x)^p - V(x) = [V(x) - 0][V(x) - 1] \ldots [V(x) - (p-1)]$, so if $V(x)$ satisfies (**) then $A(x)$ divides $V(x)^p - V(x)$ and therefore every irreducible factor $P_i(x)$ divides one of the relatively prime factors $V(x) - s$. Thus, every solution of (**) is also a solution of (*) for some $(s_1, \ldots, s_r)$. So there are exactly p^r solution of (**).

The solutions of (**) form a vector space. If Q is the (n, n)-matrix whose kth row $(q_{k,0} \cdots q_{k,n-1})$ is determined by $x^{p(k-1)} \equiv q_{k,n-1}x^{n-1} + \cdots + q_{k,1}x + q_{k,0}$[mod $A(x)$], $0 \le k \le n-1$, then

$(v_0, \ldots, v_{n-1}) \cdot Q = (v_0, \ldots, v_{n-1})$ if and only if

$$V(x) = \sum_{j=0}^{n-1} v_j x^j = \sum_{j=0}^{n-1}\sum_{k=0}^{n-1} v_k \cdot q_{k,j} \cdot x^j$$
$$\equiv \sum_{k=0}^{n-1} v_k x^{pk} = V(x^p) = V(x)^p[\text{mod } A(x)]$$

So $V(x) = v_{n-1}x^{n-1} + \cdots + v_1x + v_0$ solves (**) if and only if $(v_0, \ldots, v_{n-1}) \cdot Q = (v_0, \ldots, v_{n-1})$, and therefore the solutions of (**) are the solutions of $v(Q - I) = 0$.

The Berlekamp algorithm is given in Algorithm 7. As an example, consider the problem of factoring $A(x) = x^5 + x^3 + 2x^2 + x + 2$ in $\mathbb{Z}_3[x]$: $A'(x) = 2x^4 + x + 1$, so $\gcd[A(x), A'(x)] = 1$ and $A(x)$ is a square-free polynomial. Next the

ALGORITHM 7. Berlekamp Algorithm

```
FACTORS ← Berlekamp($A(x)$, $p$)[a]

1. Let $n$ be the degree of $A$. For $0 \le k \le n-1$
   compute the entries $(q_{k,0} \ldots q_{k,n-1})$ of the $k$th row of
   $Q$ from $x^{pk-1} \equiv q_{k,n-1}x^{n-1} + \cdots + q_{k,0}[\mathrm{mod}\ A(x)]$.
2. Transform the matrix $Q - I$ into the triangular
   form $T$ by elementary operations. From $T$ read off
   the rank $n - r$ of $Q - I$ and $r$ linearly independent
   solution vectors $v^{[1]}, \ldots, v^{[r]}$ of (**) [let $v^{[1]}$ be the
   trivial solution $(1, 0, \ldots, 0)$]. So there are $p^r$ solutions
   of (**) and $r$ irreducible factors of $A(x)$.
3. If $r = 1$ then $A(x)$ is irreducible, so set FAC-
   TORS ← [$A(x)$] and exit. Otherwise compute
   $\gcd[A(x), V^{[2]}(x) - s]$ for $0 \le s \le p-1$ and add the
   resulting factors to the list FACTORS. If FACTORS
   has fewer than $r$ elements, compute $\gcd[V^{[k]}(x) - s,$
   $F(x)]$, for $F(x) \in$ FACTORS, $0 \le s \le p-1$, $k =$
   $3, \ldots, r$, and add the resulting factors to FACTORS,
   until FACTORS contains exactly $r$ elements.
```

[a] $A(x)$ is a square-free polynomial in $\mathbb{Z}_p[x]$, and FACTORS is the list of prime factors of $A(x)$.

powers x^0, x^3, x^6, x^9, x^{12} are reduced modulo $A(x)$, yielding the entries of the matrix Q. The triangularized form of $Q - I$ is T.

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 2 & 1 & 2 \\ 0 & 1 & 1 & 2 & 2 \\ 2 & 0 & 2 & 1 & 1 \end{bmatrix} \qquad T = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

So $r = 2$ and $A(x)$ has two prime factors. Further, $v^{[1]} = (1, 0, 0, 0, 0)$, $v^{[2]} = (0, 0, 2, 1, 0)$, and $\gcd[A(x), V^{[2]}(x)] = x^2 + x + 2$, $\gcd[A(x), V^{[2]}(x) + 1] = x^3 + 2x^2 + 1$. The prime factors of $A(x)$ are $P_1(x) = x^2 + x + 2$ and $P_2(x) = x^3 + 2x^2 + 1$.

B. Factorization of Univariate Integral Polynomials

The problem is to factor a polynomial $A(x)$ in $\mathbb{Z}[x]$ into its content and all its prime irreducible polynomial factors. The content of A is readily available as the gcd of the coefficients of A. The algorithm in Algorithm 8 can be used to achieve a square-free factorization of $A(x)$, that is, a representation of $A(x)$ in the form

$$A(x) = \prod_{i=1}^{r} [A_i(x)]^i$$

where the $A_i(x)$ are relatively prime square-free polynomials and $A_r(x)$ is nonconstant. So the problem is reduced to factoring a primitive square-free polynomial, which can be done by the Berklekamp–Hensel algorithm given in Algorithm 9. In the worst case the Berlekamp–Hensel algorithm is exponential in n, the degree of the input polynomial $A(x)$. On the average, however, the algorithm performs quite well. In step (4) of the Berlekamp–Hensel algorithm, the Hensel Lemma is used.

The Hensel lemma (for univariate polynomials) is as follows. Let $A(x)$ be an integral polynomial and p a prime number. Let $A(x) \equiv F(x)G(x)$ (mod p), where $F(x)$ and $G(x)$ are relatively prime in $\mathbb{Z}_p[x]$. Then for every natural number k there are polynomials $F^{(k)}(x)$ and $G^{(k)}(x)$ in $\mathbb{Z}_{p^k}[x]$ such that $A(x) \equiv F^{(k)}(x)G^{(k)}(x)$ (mod p^k) and $F^{(k)}(x) \equiv F(x)(\mathrm{mod}\ p)$, $G^{(k)}(x) \equiv G(x)(\mathrm{mod}\ p)$.

An algorithm for computing the polynomials $F^{(k)}(x)$ and $G^{(k)}(x)$ can be found in the chapter on polynomial factorization in Buchberger *et al.* (1983).

D. R. Musser has generalized the Hensel lemma to obtain a multivariate factorization algorithm. In 1982, A. K. Lenstra, H. W. Lenstra, and L. Lovász detected an algorithm for factoring univariate polynomials in polynomial time,

ALGORITHM 8. Square-Free Factorization Algorithm for Univariate Integral Polynomials

```
FACTORS ← Squarefree($A(x)$)[a]

1. Set $A_1(x) \leftarrow A(x)$,   $A_2 \leftarrow \gcd[A_1(x), A_1'(x)]$,   $G_1(x) \leftarrow A_1(x)/A_2(x)$,
       $A_3 \leftarrow \gcd[A_2(x), A_2'(x)]$,   $G_2(x) \leftarrow A_2(x)/A_3(x)$,   $F_1(x) \leftarrow G_1(x)/G_2(x)$,
       $k \leftarrow 2$.
2. while $G_k(x) \ne 1$ do
     {$A_{k+2} \leftarrow \gcd(A_{k+1}(x), A_{k+1}'(x))$, $G_{k+1}(x) \leftarrow A_{k+1}(x)/A_{k+2}(x)$,
     $F_k(x) \leftarrow G_k(x)/G_{k+1}(x)$, $k \leftarrow k + 1$}.
```

[a] $A(x)$ is a primitive integral polynomial. FACTORS = $(F_1, \ldots, F_r)$, such that the F_i, $1 \le i \le r$, are square-free, relatively prime polynomials, and $A(x) = F_1(x) \ldots F_r(x)^r$.

ALGORITHM 9. Berlekamp–Hensel Algorithm

```
FACTORS ← Berlekamp–Hensel($A(x)$)[a]

1. Choose a prime $p$ that does not divide ldcf($A$) and so that $A(x)$ mod $p$ is square-free.
2. Call the Berlekamp algorithm to compute $U_1(x), \ldots, U_r(x)$ in $\mathbb{Z}_p[x]$ such that
   ldcf($U_1$) $\equiv$ ldcf($A$)(mod $p$), ldcf($U_2$) $= \cdots =$ ldcf($U_r$) $= 1$, and $U_1(x) \cdot \cdots \cdot U_r(x)$ is a
   complete factorization of $A(x)$modulo $p$.
3. Compute a bound $b(A)$ for the absolute values of the coefficients in the factors of $A$,
   for instance $b(A) = 2^n(a_0^2 + \ldots + a_n^2)^{1/2}$.
4. [quadratic Hensel lifting] Set $q \leftarrow p$.
   for $k = 1, 2, \ldots$ until $q \geq 2b(A)$ do
      {$q \leftarrow q^2$; compute polynomials $U_i^{(k)}(x)$ in $\mathbb{Z}_q[x]$ such that $U_1^{(k)}(x) \cdot \cdots \cdot$
      $U_r^{(k)}(x) \equiv A(x)$ (mod $q$), ldcf($U_1^{(k)}$) $\equiv$ ldcf($A$)(mod $q$), and $U_i^{(k)}(x) \equiv U_i(x)$
      (mod $p$)}.
5. [trial factor combinations] $H(x) \leftarrow A(x)$; $C \leftarrow \{2, \ldots, r\}$; $s \leftarrow 0$;
   while $C \neq \{\ \}$ do
     {for $m = 1, \ldots, |C|$ do
        for all $\{i_1, \ldots, i_m\} \subseteq C$ do
          test whether $F(x) =$ pp[ldcf($H$)$U_{i_1}^{(k)}(x) \ldots U_{i_m}^{(k)}(x)$ mod $p^{2^k}$] divides $A(x)$,
          where $k$ is the number of iterations in step (4) and the coefficients are
          balanced around 0 before taking the primitive part. If so, then leave
          both for loops.
     $s \leftarrow s + 1$; $F_s(x) \leftarrow F(x)$; $H(x) \leftarrow H(x)/F(x)$; delete the subset of $C$, which
     determines $F$, from $C$};
   $s \leftarrow s + 1$; $F_s(x) \leftarrow H(x)$.
```

[a] $A(x) = a_nx^n + \cdots + a_1x + a_0$ is a primitive square-free integral polynomial, FACTORS $= (F_1, \ldots, F_s)$ such that the F_i, $1 \leq i \leq s$, are primitive irreducible polynomials, and $A(x) = F_1(x) \ldots F_s(x)$.

and E. Kaltofen showed how to reduce the problem of factoring multivariate polynomials to univariate factorization, also in polynomial time. In practice, however, the Berlekamp–Hensel approach is still preferred, and a variant of it is used as the standard multivariate factorization algorithm in Macsyma. References for factoring polynomials over algebraic extensions of $\mathbb{Q}$ can be found in the chapter on polynomial factorization in Buchberger *et al.* (1983).

C. Real Root Isolation

Let $A(x) = a_nx^n + \cdots + a_1x + a_0$ be an integral polynomial and ε a positive rational number. The problem is to compute a sequence of disjoint intervals of length less than ε, each containing exactly one real zero of A and together containing all real zeros of A.

First, using the square-free factorization algorithm in the previous section, A is written as

$$A(x) = \text{cont}(A) \prod_{i=1}^{r} [A_i(x)]^i$$

where $A_i(x)$ is a primitive square-free polynomial for $1 \leq i \leq r$. The problem reduces to the problem of isolating the real roots of the primitive square-free polynomials $A_i(x)$, $1 \leq i \leq r$.

There are various algorithms for isolating the real roots of a primitive square-free polynomial. A survey, including Kronecker's algorithm, Sturm's algorithm, and an algorithm using Rolle's theorem, can be found in the chapter on real zeros of polynomials in Buchberger (1983). The most efficient algorithm seems to be the modified Uspenski algorithm, whose original version was given by Uspenski in 1948 and modified (eliminating the exponential computing time) by Collins and Akritas in 1976. It suffices to isolate the positive zeros of $A(x)$, since the negative zeros of $A(x)$ are the positive zeros of $A(-x)$ and $A(0) = 0$ if and only if $a_0 = 0$.

Some notation is useful at this point. Let p(A) be the number of positive zeros of the polynomial A, multiplicities counted. If $(u_1, \ldots, u_s)$ is a sequence of real numbers, $(u_1', \ldots, u_t')$ the subsequence of all nonzero numbers, then var$(u_1, \ldots, u_s) = |\{i \mid 1 \leq i < t, u_i'u_{i+1}' < 0\}|$, the sign variation

ALGORITHM 10. Modified Uspenski Algorithm

```
$L \leftarrow$ Modified-Uspenski$(A(x))$[a]

1. Let $b'(A)$ be a bound for the absolute values of the roots of $A$, for instance,
   $b'(A) = 1 + (\max\{a_{n-1}, \ldots, a_0\})/|a_n|$. Let $s^k \geq b'(A)$.
2. If $k \geq 0$ set $B(x) = A(2^k x)$, otherwise set $B(x) = 2^{-kn}A(2^k x)$. [1 is now a root
   bound for $B$.]
3. Call Roots01 to compute the list $L'$ of isolating intervals for the real zeros
   of $B$ in $(0, 1)$.
4. Let $L$ be the list containing $(2^k c, 2^k d)$ for every $(c, d)$ in $L'$.

$L \leftarrow$ Roots01$(A(x))$[b]

1. Set $A^* = (x + 1)^n A[1/(x + 1)]$ [The zeros of $A$ in $(0, 1)$ are transformed onto
   the zeros of $A^*$ in $(0, \infty)$].
   If var$(A^*) = 0$ then set $L \leftarrow ( \ )$ and exit.
   Otherwise, if var$(A^*) = 1$ then set $L \leftarrow [(0, 1)]$ and exit.
2. If $A(\frac{1}{2}) = 0$ then include $(\frac{1}{2}, \frac{1}{2})$ in $L$ and replace $A$ by $A/(2x - 1)$.
3. Set $A' \leftarrow 2^n A(x/2)$ [the zeros of $A$ in $(0, \frac{1}{2})$ are transformed onto the zeros of
   $A'$ in $(0, 1)$]. Apply Roots01 recursively to $A'$ with result $L'$. Replace every inter-
   val $(c', d')$ in $L'$ by $(c'/2, d'/2)$ in $L$.
4. Set $A'' \leftarrow A'(x + 1)$ [the zeros of $A$ in $(\frac{1}{2}, 1)$ are transformed onto the zeros
   of $A''$ in $(0, 1)$]. Apply Roots01 recursively to $A''$ with result $L''$. Replace every
   interval $(c'', d'')$ in $L''$ by $[(c'' + 1)/2, (d'' + 1)/2]$ in $L$.
```

[a] $A(x) = a_n x^n + \cdots + a_0$ is a primitive square-free integral polynomial and L is a list of isolating intervals for the positive real zeros of A.
[b] $A(x) = a_n x^n + \cdots + a_0$ is a square-free integral polynomial. L is a list of isolating intervals for the real zeros of A in $(0, 1)$.

of A. For a polynomial A, let d_A, the seminorm of A, be the sum of the absolute values of the coefficients of A.

By Descartes's rule, var(A) = var$(a_n, \ldots, a_0)$ = p$(A) + m$ for some even nonnegative integer m. So if var$(A) = 0$, then p$(A) = 0$, and if var$(A) = 1$ then p$(A) = 1$. If A is a nonzero real polynomial with no zeros in the right half-plane, then var$(A) = 0$. Let A be a nonzero real polynomial of degree $n \geq 2$. Let $T(A)(x)$ be $2^n A(x/2)$ or $2^n A[(x + 1)/2]$. Then the polynomial $T^k(A)$, $k = O[nL(d_A)]$, has at most one zero α_n in $(0, 1)$ and for all other zeros $\alpha_1, \ldots, \alpha_{n-1}$, real or complex, $|\alpha_i| > r_n = O(n^2)$. For the polynomial $A^*(x) = (x + 1)^n T^k(A)[1/(x + 1)]$, one has var$(A^*) = 1$. These observations are the basis for the modified Uspenski algorithm given in Algorithm 10.

VII. Arithmetic in Basic Domains

This chapter deals with arithmetic in some of the basic domains of computer algebra. These are the integers $\mathbb{Z}$, the rationals $\mathbb{Q}$, the integers modulo some natural number m, p-adic numbers, algebraic extensions of $\mathbb{Q}$, polynomials over these domains, and power series. In contrast to the approach in numerical mathematics, all these domains are represented exactly. So the computational problems, although similar to the ones considered in numerical mathematics, have quite a different flavor. In the previous sections, many applications of the arithmetic in these basic domains have been described for which it is absolutely essential to compute exact results. Only a brief account can be given here. For more details the reader is referred to the chapter on arithmetic in algebraic domains in Buchberger *et al.* (1983) and to the appropriate chapters in Knuth (1981).

A. The Integers

The elements of $\mathbb{Z}$ are represented in a positional number system with a base or radix β (≥ 2). A positive integer a is represented by the unique sequence $a_{(\beta)} = (a_0, \ldots, a_{n-1})$ of nonnegative β-digits, such that $a_{n-1} > 0$ and $a = \sum_{k=0}^{n-1} a_i\beta^i$. Nonpositive β-digits are used for representing negative integers. The empty sequence represents the integer 0. For $a \neq 0$, represented as $(a_0, \ldots, a_{n-1})$, n is called the β-length of a, written as $L_\beta(a)$. Let $L_\beta(0) = 1$. $L_\beta(a)$ is essentially the β-logarithm of $|a|$, and since $L_\beta(a)/L_\gamma(a)$ is constant for different bases β and γ, one sometimes just speaks of $L(a)$, the length of a.

A careful analysis of the propagation of carries shows that the sum of two integers a and b can be computed in an average computing time proportional to $\min[L(a), L(b)]$.

The "classical" algorithm for multiplying two integers a and b has a computing time proportional to $L(a) \cdot L(b)$. For long integers (a realistic tradeoff point seems to be 200 decimal digits), an algorithm attributed to A. Karatsuba may be used. The basic idea for Karatsuba's algorithm is to bisect the two integers a and b into two parts each, $a = A_1\beta^k + A_0$, $b = B_1\beta^k + B_0$, where k is about half the maximum length of a and b. Then the product $c = a \cdot b$ can be computed as

$$c = (A_1B_1)\beta^{2k} + [(A_1 + A_0)(B_1 + B_0) - A_1B_1 - A_0B_0]\beta^k + (A_0B_0)$$

That means one of the four multiplications can be replaced by additions and shifting operations. The computing time for Karatsuba's algorithm is proportional to $\max[L(a), L(b)]^{\log_2 3}$. A. Schönhage and V. Strassen have devised an even faster algorithm with computing time $mL(m)L[L(m)]$, where m is the maximum of the lengths of the multiplicands. The practicality of this algorithm still has to be determined.

Often it is necessary to compute the greatest common divisor of two integers a and b together with integers u and v such that $\gcd(a, b) = a \cdot u + b \cdot v$. The extended Euclidean algorithm does precisely that (see Algorithm 11). In 1938, D. H. Lehmer improved the Euclidean algorithm. His idea was to consider short integers a', b', a'', b'' such that $a'/b' < a/b < a''/b''$ (one can use approximately rounded versions of the first few digits of a and b) and carry out the Euclidean algorithm on (a', b') and (a'', b'') instead of on (a, b). As long as the intermediary results are the same, one can be sure that they are identical to the intermediary results for (a, b). If a discrepancy occurs, new values for a', b', a'', and b'' can be computed from the last correct intermediary result.

ALGORITHM 11. Extended Euclidean Algorithm

```
(c, u, v) ← Extended-Euclidean(a, b)[a]

1. $(u_1, u_2, u_3) \leftarrow (\text{sign}(a), 0, |a|)$;
   $(v_1, v_2, v_3) \leftarrow (0, \text{sign}(b), |b|)$;
2. while $v_3 \neq 0$ do
       {let $q$ be the integer quotient of $u_3$ divided by
       $v_3$;
       $(t_1, t_2, t_3) \leftarrow (u_1, u_2, u_3) - (v_1, v_2, v_3) \cdot q$;
       $(u_1, u_2, u_3) \leftarrow (v_1, v_2, v_3)$;
       $(v_1, v_2, v_3) \leftarrow (t_1, t_2, t_3)$}.
3. Set $c \leftarrow u_3$, $u \leftarrow u_1$, $v \leftarrow u_2$.
```

[a] a and b are integers. $c = \gcd(a, b) = a \cdot u + b \cdot v$.

B. The Rational Numbers

In $\mathbb{Q}$ one is essentially interested in efficient algorithms for addition and multiplication. In order to compute the sum (or product) of the two rational numbers a/b and c/d, one can of course compute $(ab + bc)/bd$ (or ac/bd) and reduce the result to lowest terms, that is, eliminate the gcd of the numerator and denominator. In 1956, P. Henrici gave a more efficient method for addition and multiplication. The idea is not to do one gcd calculation involving the relatively large numerator and denominator of the final result, but to do several gcd calculations on smaller intermediary results. In a similar way one gets Henrici algorithms for addition and multiplication in $\mathbb{Q}(x)$ from the basic operations in $\mathbb{Z}[x]$.

C. Integers Modulo m

There are two natural choices for representatives of the residue classes of $\mathbb{Z}$ modulo a positive integer m, namely $\{0, 1, \ldots, m - 1\}$ and $\{a : -m/2 < a \leq m/2\}$. Let H_m denote the function that returns the representative of the residue class to which its input belongs. The sum, difference, and product of two residue classes with representatives a and b can be computed as

$$a +_m b = H_m(a + b) \qquad a -_m b = H_m(a - b)$$
$$a \cdot_m b = H_m(a \cdot b).$$

Division of a by b is defined only if there is a unique c such that $a = bc$, and then c is the quotient a/b. An efficient way of computing a/b is to use the extended Euclidean algorithm with inputs m and b, which produces $\gcd(m, b) = m \cdot u + b \cdot v$. Then a/b is defined if and only if $\gcd(m, b) = 1$ and then $c = a \cdot H_m(v)$.

A typical application for modular arithmetic is that one wants to replace a computation in $\mathbb{Z}$ (with possibly huge intermediary results) by a number of computations with respect to relatively prime moduli. So it is essential to be able to combine the partial results with respect to the various moduli and thus get the desired result in $\mathbb{Z}$. This leads to the so called Chinese remainder problem:

ALGORITHM 12. Chinese Remainder Algorithm

```
$r \leftarrow$ Chinese-rem2($r_1, m_1, r_2, m_2$)[a]

$c \leftarrow m_1^{-1} \bmod m_2$ (use the extended Euclidean algorithm); $r_1' \leftarrow r_1 \bmod m_1$;
$s \leftarrow [(r_2 - r_1' \bmod m_2) \cdot c] \bmod m_2$; $r \leftarrow r_1' + s \cdot m_1$.

$r \leftarrow$ Chinese-rem($n, (r_1, \ldots, r_n), (m_1, \ldots, m_n)$)[b]

$M' \leftarrow m_1$; $r \leftarrow r_1 \bmod m_1$;
for $k = 2$ to $n$ do
   {$r \leftarrow$ Chinese-rem2($r, M', r_k, m_k$); $M' \leftarrow M' \cdot m_k$}.
```

[a] r_1, m_1, r_2, m_2 are integers, $m_1 \neq 0$, $m_2 \neq 0$, $\gcd(m_1, m_2) = 1$. $r \equiv r_i$ for $1 \leq i \leq 2$.
[b] $r_1, \ldots, r_n$ are integers, $m_1, \ldots, m_n$ are nonzero, relatively prime integers. $r \equiv r_i$ (mod m_i) for $1 \leq i \leq n$.

For given integers $r_1, \ldots, r_n$ and nonzero, relatively prime integers $m_1, \ldots, m_n$, find an integer r such that $r \equiv r_i (\bmod\ m_i)$ for $1 \leq i \leq n$.

An algorithm for solving the Chinese remainder problem is given in Algorithm 12. For generalizations and details, the reader is referred to Lipson (1981).

D. Polynomials

Whenever R is a ring with effective basic arithmetic operations, then so is $R[x_1, \ldots, x_n]$, the ring of polynomials over R in n indeterminates. There are essentially four different representations of polynomials. In a sparse representation, $A(x_1, \ldots, x_n)$ is represented as a list containing the nonzero coefficients in A together with the corresponding exponents. In a dense representation, all the coefficients of A, down from the leading coefficient, are recorded. Both sparse and dense polynomials can be represented recursively—that is, a polynomial in $R[x_1, \ldots, x_n]$ is represented as a polynomial in the indeterminate x_n over $R[x_1, \ldots, x_{n-1}]$—or distributively—that is, as a list of exponent vectors and coefficients in R.

For dense polynomials, fast multiplication algorithms have been developed. They are based on the Karatsuba method or the fast Fourier transform. The tradeoff points for these fast algorithms, however, are rather high.

For univariate polynomials over a field, there is the well-known division algorithm that, for given polynomials $A(x)$ and $B(x)$, yields a unique quotient $Q(x)$ and remainder $R(x)$ such that $A(x) = Q(x)B(x) + R(x)$ and $R(x)$ is the zero polynomial or $\deg(R) < \deg(B)$. There is no corresponding division algorithm for $\mathbb{Z}[x]$. For most applications, however—for instance, for the computation of polynomial remainder sequences—it is sufficient to have a pseudo-division, which yields a unique pseudo-quotient $Q(x)$ and pseudo-remainder $R(x)$ such that $\mathrm{ldcf}(B)^{m-n+1}A(x) = Q(x)B(x) + R(x)$ and $R(x)$ is the zero polynomial or $\deg(R) < \deg(B)$, where $m = \deg(A) \geq \deg(B) = n$. Pseudo-division can be carried out in $\mathbb{Z}[x]$.

If R is a field, then $R[x]$ is a Euclidean domain, so the Chinese remainder algorithm (Algorithm 12) can be employed for solving the problem:

For given polynomials $A_1(x), \ldots, A_n(x)$ and nonzero, relatively prime polynomials $P_1(x), \ldots, P_n(x)$, find a polynomial $A(x)$ such that $A(x) \equiv A_i(x)[\bmod\ P_i(x)]$ for $1 \leq i \leq n$.

A solution to this Chinese remainder problem for polynomials is needed for the proof of the Berlekamp algorithm.

VIII. Program Systems for Computer Algebra

Very early in the history of computer algebra, program systems were constructed to carry out the newly developed algorithms. Many of these early systems were programmed in FORTRAN, notable among them Brown's ALPAK (1963) and Collins's PM (1966). Today there is a large number of computer algebra systems. The majority of them are devoted to special applications like high-energy physics, celestial mechanics, or general relativity. The most influential among the general-purpose systems probably are Macsyma, Reduce, SAC/ALDES, and mu-Math, every one for different reasons. All of them have originated more or less between 1970 and 1980.

Macsyma, developed at MIT under J. Moses, is certainly the most comprehensive system. Most of the algorithms presented in the previous chapters are implemented in some way in Macsyma. It has only recently become available to a large number of users. From the early beginnings, Reduce has always put a high emphasis on portability and it is probably the full-fledged computer algebra system with the widest distribution. Reduce has been developed at the University of Utah under Hearn. Collins's SAC/ALDES system is the best documented of the major computer algebra systems. Its algorithms form a huge "library" of algebraic programs, which are frequently used as a basis for new implementations. Stoutemyer of the University of Hawaii was the first to write a computer algebra system for a microprocessor. His mu-Math is able to perform most of the symbolic calculations for high-school mathematics. New computer algebra systems are currently developed, among them Scratchpad, SMP, and Maple.

The following is a conversation with Macsyma. The lines starting with (c ...) are input and the ones starting with (d ...) are responses of the system. Comments are placed between /* and */. The symbol % refers to the expression in the previous line.

```
(c1) /* Macsyma is able to handle arbitrarily long integers */

     (13*x+43)^3 * (214*x-25*y)^2 * (17*x+27*y+12*x*y)^3;

Time= 150 msec.

                        3               2                       3
(d1)         (13 x + 43)  (214 x - 25 y)  (12 x y + 27 y + 17 x)

(c2) expand(%);

Time= 4233 msec.

                 6  5                5  5                 4  5
(d2) 2372760000 x  y  + 39561210000 x  y  + 272845462500 x  y

                 3  5                 2  5                    5
 + 996174871875 x  y  + 2030599884375 x  y  + 2191214075625 x y

                 5                7  4                 6  4
 + 978085175625 y  - 40621651200 x  y  - 667203685200 x  y

                  5  4                   4  4                   3  4
 - 4525668693000 x  y  - 16222173247125 x  y  - 32402893073625 x  y

                   2  4                     4                 8  3
 - 34195733596575 x  y  - 14897323986075 x y  + 173860667136 x  y

                  7  3                   6  3                   5  3
 + 2726150259456 x  y  + 17516626173540 x  y  + 58917553200945 x  y

                    4  3                    3  3                   2  3
 + 109157302390365 x  y  + 105328551365091 x  y  + 41201957969451 x  y

                 8  2                   7  2                   6  2
 + 738907835328 x  y  + 10412748356400 x  y  + 58018090580645 x  y

                    5  2                    4  2                    3  2
 + 159575990863065 x  y  + 216419302250655 x  y  + 115702069522163 x  y

                  8                     7                     6
 + 1046786100048 x  y + 12627113405192 x  y + 56583575068488 x  y

                    5                     4                   8
 + 111396993677592 x  y + 81054958674248 x  y + 494315658356 x

                  7                   6                   5
 + 4905132302148 x  + 16224668384028 x  + 17888736936236 x

(c3) /* multivariate integral polynomials can be factored and their
     gcd computed */

     factor(%);

Time= 18866 msec.

                        3               2                       3
(d3)         (13 x + 43)  (25 y - 214 x)  (12 x y + 27 y + 17 x)

(c4) (13*x+43)^4 * (214*x-25*y) * (5*x^2-2*x+1);

Time= 116 msec.
```

```
                            4     2
(d4)             (13 x + 43)  (5 x  - 2 x + 1) (214 x - 25 y)

(c5) expand(%);

Time= 650 msec.

                6               5                4                3
(d5) - 3570125 x  y - 45807450 x  y - 216180575 x  y - 432498300 x  y

              2                                           7              6
 - 267504075 x  y + 67580950 x y - 85470025 y + 30560270 x  + 392111772 x

               5               4               3              2
 + 1850505722 x  + 3702185448 x  + 2289834882 x  - 578492932 x  + 731623414 x

(c6) gcd(d2,d5);

Time= 6550 msec.

             3           2                                    4            3
(d6) (54925 x  + 545025 x  + 1802775 x + 1987675) y - 470158 x  - 4665414 x

                                                              2
                                                  - 15431754 x  - 17014498 x

(c7) factor(%);

Time= 683 msec.

                                     3
(d7)                      (13 x + 43)  (25 y - 214 x)

(c8) /* the resultant and discriminant computed in the following lines are
     needed in the example in Chapter II */

     y^2 + x^2 - 4;

Time= 16 msec.

                                   2    2
(d8)                              y  + x  - 4

(c9) y^2 - 2*x + 2;

Time= 33 msec.

                                  2
(d9)                             y  - 2 x + 2

(c10) resultant(d8,d9,y);

Time= 1016 msec.

                                  2           2
(d10)                           (x  + 2 x - 6)

(c11) poly_discriminant(d8,y);

Time= 66 msec.

                                           2
(d11)                              16 - 4 x

(c12) /* algebraic equations can be solved */

      (x-b-a^2)*(x-b-2*a)*(x+3*a-b^2);

Time= 83 msec.

                                           2        2
(d12)              (x - b - 2 a) (x - b - a ) (x - b  + 3 a)

(c13) expand(%);

Time= 666 msec.

       3    2  2        2    2  2      2      3      2  2          2      2
(d13) x  - b  x  - 2 b x  - a  x  + a x  + 2 b  x + a  b  x + 2 a b  x + b  x

    2                  3        2      4    2  3        3      3  2        2
 + a  b x - 4 a b x - a  x - 6 a  x - b  - a  b  - 2 a b  - 2 a  b  + 3 a b

      3       2       4
 + 3 a  b + 6 a  b + 6 a
```

```
(c14) solve(%,x);

Time= 8266 msec.

                                          2                     2
(d14)                 [x = b + 2 a, x = b  - 3 a, x = b + a ]

(c15) /* in some cases closed form solutions to indefinite summation
      problems can be computed */

      simpsum : true$

Time= 0 msec.

(c16) sum(i^3 + 3^i,i,0,n);

Time= 566 msec.

                               n + 1        4      3    2
                              3      - 1   n  + 2 n  + n
(d16)                         ---------- + --------------
                                  2              4

(c17) sum(i^2,i,1,4) * sum(1/i^2,i,1,inf);

Time= 766 msec.

                                            2
(d17)                                  5 %pi

(c18) /* derivatives can be computed */

      x^x^x;

Time= 0 msec.

                                           x
                                          x
(d18)                                    x

(c19) diff(%,x);

Time= 100 msec.

                            x
                           x    x                             x - 1
(d19)                     x   (x  log(x) (log(x) + 1) + x      )

(c20) /* the result is checked by integration */

      integrate(%,x);

Time= 15800 msec.

                                            x log(x)
                                 log(x) %e
(d20)                          %e

(c21) /* simplification yields the original formula */

      radcan(%);

Time= 800 msec.

                                           x
                                          x
(d21)                                    x

(c22) /* the following examples should illustrate some of the features
      of Macsyma's integration package */

      1/(x^3+a*x^2+x);

Time= 33 msec.

                                          1
(d22)                               -------------
                                     3      2
                                    x  + a x  + x

(c23) integrate(%,x);

       2
Is  a   - 4  positive or negative?

positive;
```

```
Time= 1083 msec.

                                   2
                   2 x - sqrt(a  - 4) + a
             a log(----------------------)
                                   2               2
                   2 x + sqrt(a  - 4) + a    log(x  + a x + 1)
(d23)      - ---------------------------- - ----------------- + log(x)
                              2                    2
                    2 sqrt(a  - 4)

(c24) %e^x^(1/2);

Time= 33 msec.

                                      sqrt(x)
(d24)                               %e

(c25) integrate(%,x);

Time= 1250 msec.

                                              sqrt(x)
(d25)                       2 (sqrt(x) - 1) %e

(c26) x/(x^2-1);

Time= 33 msec.

                                        x
(d26)                                 ------
                                        2
                                       x  - 1

(c27) integrate(%,x);

Time= 133 msec.

                                         2
                                   log(x  - 1)
(d27)                              -----------
                                        2

(c28) (log(x)-1)/(log(x)^2 - x^2);

Time= 83 msec.

                                   log(x) - 1
(d28)                             ------------
                                     2        2
                                  log (x) - x

(c29) integrate(%,x);

Time= 2216 msec.

                        log(log(x) + x)   log(log(x) - x)
(d29)                   --------------- - ---------------
                               2                 2

(c30) /* definite integration from 0 to a */

      1/(a^2-x^2)^(1/2);

Time= 33 msec.

                                         1
(d30)                              -------------
                                        2    2
                                   sqrt(a  - x )

(c31) integrate(%,x,0,a);

Is  a  positive or negative?

positive;

Time= 31933 msec.

                                        %pi
(d31)                                   ---
                                         2

(c32) 1/(x^2-a^2);
```

```
Time= 50 msec.

                                                 1
(d32)                                         -------
                                               2    2
                                              x  - a

(c33) integrate(%,x,a,b);

Is  (b - a) (b + a)  positive or negative?

positive;
INTEGRAL IS DIVERGENT


Time= 5766 msec. so far

(c34) /* definite integration from 0 to infinity */

      %e^(-x)*x^n;

Time= 66 msec.

                                              n   - x
(d34)                                        x  %e

(c35) integrate(%,x,0,inf);

Is  n + 1  positive or negative?

positive;

Is  n  positive or negative?

positive;

Is  n  an integer?

yes;

Time= 38983 msec.

(d35)                                           n!

(c36) /* Macsyma has a package for solving ordinary differential
      equations. The solution contains a parameter %c */

      x^2*'diff(y,x) + 3*x*y = sin(x)/x;

Time= 83 msec.

                                     2 dy             sin(x)
(d36)                               x  -- + 3 x y = ------
                                       dx                x

(c37) ode2(%,y,x);

Time= 2033 msec.

                                         %c - cos(x)
(d37)                                y = -----------
                                              3
                                             x

(c38) (1+x^2)*'diff(y,x,2) - 2*y = 0;

Time= 50 msec.

                                              2
                                      2      d y
(d38)                               (x  + 1) --- - 2 y = 0
                                               2
                                             dx

(c39) ode2(%,y,x);

Time= 6483 msec.

                           2        atan(x)       x                  2
(d39)         y = %k1 (x  + 1) (------- + ---------) + %k2 (x  + 1)
                                      2          2
                                             2 x  + 2

(c40) 'diff(y,x,2) - 2*'diff(y,x,1) - 2*y = 0;
```

```
Time= 66 msec.

                                 2
                                d y      dy
(d40)                           --- - 2 -- - 2 y = 0
                                  2      dx
                                dx

(c41) ode2(%,y,x);

Time= 1200 msec.
                             (2 sqrt(3) + 2) x         (2 - 2 sqrt(3)) x
                             -----------------         -----------------
                                     2                         2
(d41)          y = %k1 %e                        + %k2 %e

(c42) quit();
```

Bibliography

Arnon, D. S., Collins, G. E., and McCallum, S. (1984). *SIAM J. Comput.* **13**(4), 865–889.

Atkinson, M. D., ed. (1984). "Computational Group Theory." Academic Press, New York.

Buchberger, B., Collins, G. E., and Loos, R. (1983). "Computer Algebra—Symbolic and Algebraic Computation," 2nd ed. Springer-Verlag, Wien-New York.

Huet, G., and Oppen, D. C. (1980). Equations and Rewrite Rules: A Survey, *In* "Formal Language Theory" (R. V. Book, ed.). Academic Press, New York. 349–405.

Journal of Symbolic Computation, Academic Press, 1985–.

Special Volume on Decision Methods for Real Closed Fields, Journal of Symbolic Computation, Academic Press, 1986.

Knuth, D. E. (1981). "The Art of Computer Programming," 2nd ed., Vol. 2. Addison-Wesley, Reading, Massachusetts.

Lipson, J. D. (1981). "Elements of Algebra and Algebraic Computing." Addison-Wesley, Reading, Massachusetts.

Pavelle, R., ed. (1985). "Applications of Computer Algebra." Kluwer Academic Publishers, Boston-Dordrecht-Lancaster.