Formalizing the Wu-Ritt Characteristic Set Method in Lean 4

Yuxuan Xiao[1,2], Hao Shen[2], Junyu Guo[3], Dingkang Wang[2], and Lihong Zhi[2]

[1] School of Mathematics, Shandong University, Shandong, 250100, China
yuxuanxiao@mail.sdu.edu.cn
[2] State Key Laboratory of Mathematical Sciences, Academy of Mathematics and Systems Science, University of Chinese Academy of Sciences, Beijing, 100190, China shenhao24@amss.ac.cn, {dwang, lzhi}@mmrc.iss.ac.cn
[3] Institute of Logic and Cognition, Department of Philosophy, Sun Yat-sen University, 510275, Guangdong, China
guojy228@mail2.sysu.edu.cn

Abstract. We formalize the Wu-Ritt characteristic set method in Lean 4 for triangular decomposition of multivariate polynomial systems. The development covers polynomial orders, initials, pseudo-division, triangular and ascending sets, basic and characteristic sets, and algorithms for characteristic-set computation and zero decomposition, with termination and correctness proofs. In particular, we verify the well-ordering principle and the zero decomposition theorem, providing a machine-checked foundation for certified polynomial system solving and geometry theorem proving.

Keywords: Wu-Ritt characteristic set method, ascending sets, pseudo-division, characteristic sets, Lean

1 Introduction

The Wu–Ritt characteristic set method [4] introduced by Wu Wen-Tsun [6], is a fundamental tool for solving systems of multivariate polynomial equations and for automated theorem proving in elementary geometry. In this paper, we present a formalization of the Wu-Ritt characteristic set method in Lean 4 [3], following Wu's mathematical and algorithmic treatment [7]. Our development formalizes the basic concepts of the theory, including pseudo-division, pseudo-remainder with respect to a polynomial or a triangular set, ascending sets, basic sets, and characteristic sets. We verify not only the main algorithms of the method but also the central statements of the Wu–Ritt theory. In particular, we formalize the well-ordering principle that connects a polynomial system with its characteristic set, and we prove the correctness of the zero decomposition algorithm, showing that it expresses the zero set of the original system as a finite union of zero sets of triangular sets, away from the zeros of the corresponding initials.

In Coq [4], a formalization of Wu's simple method was previously carried out for proving geometric theorems [1]. Our work provides a comprehensive formalization of the characteristic set method in Lean 4. It builds on Mathlib's algebraic infrastructure and provides a basis for certified symbolic computation and automated geometry theorem proving in the Lean ecosystem [5].

2 Preliminaries

Let R be a commutative ring with identity, and let σ be a finite set of indices equipped with a linear order $<$. We write $R[X_i]_{i\in\sigma}$ for the polynomial ring under consideration, ordering the variables by declaring

$$X_i < X_j \quad \text{whenever } i < j.$$

[4] : The terminology "Wu-Ritt" is used here to emphasize Wu Wen-Tsun's algorithmic development of the characteristic set method, built on Ritt's earlier theory of characteristic sets. In particular, we follow Wu's terminology and treatment in [6].

Let $p \in R[X_i]_{i\in\sigma}$ be a polynomial. We denote the greatest variable appearing in p by X_{m_p} and call it the *main variable* of p, where $m_p \in \sigma$ is the index of that variable. We define the *main degree* of p, denoted by $\deg(p)$, to be the degree of p with respect to its main variable, namely $\deg_{m_p}(p)$. If p is a constant polynomial, then its main variable is undefined, and we define $\deg(p) = 0$.

Definition 1 (Polynomial Order). *Let p and q be two non-zero multivariate polynomials with main variables X_{m_p}, X_{m_q}, respectively. We say p is less than q, denoted by $p \prec q$, if either $m_p < m_q$ or $m_p = m_q$ and $\deg(p) < \deg(q)$.*

The order $\prec$ can be viewed as the lexicographic order on $(m_p, \deg(p))$, comparing first the main-variable index and then the degree. Unlike Gröbner-basis monomial orders, which compare monomials to define leading terms and reductions [2], it compares whole polynomials for our triangular/pseudo-division procedure. Thus, it is neither induced by nor intended to extend such monomial orders; apart from `MvPolynomial`, the two formalizations are largely independent.

```
def order (p : MvPolynomial σ R) : WithBot σ ×ₗ ℕ := (p.vars.max, p.mainDegree)
```

We say that p and q have the same ordering, denoted by $p \sim q$, if neither $p \prec q$ nor $q \prec p$ holds. Equivalently,

$$p \sim q \quad \iff \quad m_p = m_q \ \wedge \ \deg(p) = \deg(q).$$

We write $p \preceq q$ if $p \prec q$ or $p \sim q$. The relation $\preceq$ defines a preorder on $R[X_i]_{i\in\sigma}$, and $\sim$ is the corresponding equivalence relation. Hence $\preceq$ induces a partial order on the quotient $R[X_i]_{i\in\sigma}/\sim$.

```
instance : Preorder (MvPolynomial σ R) where
  le := InvImage (· ≤ ·) order
  le_refl := fun _ ↦ by rw [InvImage]
  le_trans := fun _ _ _ ↦ le_trans

theorem lt_def : p < q ↔ p.vars.max < q.vars.max ∨
    p.vars.max = q.vars.max ∧ p.mainDegree < q.mainDegree

instance : Setoid (MvPolynomial σ R) :=
  AntisymmRel.setoid (MvPolynomial σ R) (· ≤ ·)
```

Definition 2 (Initial). *Let p be a non-constant polynomial with main variable X_i. Writing p as a polynomial in X_i, $p = c_d X_i^d + c_{d-1} X_i^{d-1} + \cdots + c_0$, where each $c_k \in R[X_j]_{j<i}$ and $c_d \neq 0$, we call c_d the initial of p, and denote it by $\mathrm{init}(p)$ or simply I_p. We define the initial of the zero polynomial to be 0. If p is a nonzero constant, then we set $I_p = 1$.*

```
def initialOf (p : MvPolynomial σ R) (i : σ) : MvPolynomial σ R :=
  Σ s ∈ p.support with s i = p.degreeOf i, monomial (s.erase i) (p.coeff s)

def initial (p : MvPolynomial σ R) : MvPolynomial σ R :=
  if p = 0 then 0 else
    match p.vars.max with
    | ⊥ => 1
    | some m => p.initialOf m
```

```
theorem initialOf_eq_leadingCoeff [DecidableEq σ] {p : MvPolynomial σ R} {i : σ} :
    p.initialOf i = rename Subtype.val
      (optionEquivLeft R {b // b ≠ i} (rename (Equiv.optionSubtypeNe i).symm p)).leadingCoeff
```

The *initial product* of polynomials $p_1, p_2, \ldots, p_r$ is defined to be the product of their initials $\prod_{i=1}^{r} I_{p_i}$.

Example 1. Consider $p = 5xy^2 + x^3y, q = y^2 + x^2y, r = x^5 \in \mathbb{Z}[x, y]$ with $x < y$. Then $r \prec p$ and $p \sim q$, $I_p = \mathrm{init}_y(p) = 5x$, $I_q = 1$, and $I_r = 1$.

Definition 3 (Reduce). *Let p be a non-constant polynomial with main variable X_{m_p}. A polynomial q is said to be reduced with respect to p if the degree of q in X_{m_p} is strictly smaller than the main degree of p. By convention, the zero polynomial is reduced with respect to every polynomial. If p is a constant polynomial, then no nonzero polynomial is reduced with respect to p.*

```
def reducedTo (q p : MvPolynomial σ R) : Prop :=
  if q = 0 then True
  else
    match p.vars.max with
    | ⊥ => False
    | some m => q.degreeOf m < p.degreeOf m
```

Let PS be a set of polynomials. We say that a polynomial q is *reduced with respect to* PS if it is reduced with respect to every polynomial in PS.

3 Triangular Set

Definition 4 (Triangular Set). *A set of polynomials S is called a triangular set if all its elements are nonzero and can be arranged as a sequence*

$$s_1, s_2, \ldots, s_r \in R[X_i]_{i \in \sigma},$$

whose main variables $X_{m_1}, X_{m_2}, \ldots, X_{m_r}$ satisfy

$$m_1 < m_2 < \cdots < m_r.$$

```
structure TriangularSet (σ R : Type*) [CommSemiring R] [LinearOrder σ] where
  /-- The underlying list of non-zero polynomials -/
  toList : List (MvPolynomial σ R)
  /-- No polynomial in the list is zero -/
  zero_notMem' : 0 ∉ toList
  /-- The main variables of successive polynomials are strictly increasing -/
  isChain_max_vars' : toList.IsChain (·.vars.max < ·.vars.max)

instance : FunLike (TriangularSet σ R) ℕ (MvPolynomial σ R) where
  coe S n := S.toList[n]?.getD 0
  coe_injective' := ...

def length (S : TriangularSet σ R) : ℕ := S.toList.length
```

Especially, our formalization allows s_1 to be a nonzero constant. In this case, m_1 is undefined, while $m_2 < \cdots < m_r$.

When S is a triangular set, we always write it as $s_1, \ldots, s_r$ so that the main variables of the s_i are strictly increasing with i.

Definition 5 (Ordering of Triangular Sets). *Let $S = (s_1, \ldots, s_r)$ and $S' = (s'_1, \ldots, s'_{r'})$ be two triangular sets. We say that S is less than S', and write $S \prec S'$, if one of the following conditions holds:*

1. *there exists $1 \le k \le \min(r, r')$ such that*

$$s_1 \sim s'_1,\ s_2 \sim s'_2,\ \ldots,\ s_{k-1} \sim s'_{k-1},\ \text{while } s_k \prec s'_k.$$

2. *$r > r'$ and $s_i \sim s'_i$ for all $1 \le i \le r'$.*

```
-- The indices here are 0-based.
def order (S : TriangularSet σ R) : Lex (ℕ → WithTop (WithBot σ ×ₗ ℕ)) :=
  fun i ↦ if i < S.length then WithTop.some (S i).order else ⊤
```

```
theorem lt_def : S < T ↔ (∃ k < S.length, S k < T k ∧ ∀ i < k, S i ≈ T i) ∨
    (T.length < S.length ∧ ∀ i < T.length, S i ≈ T i)
```

As in Definition 1, we extend the relations $\preceq$ and $\sim$ to triangular sets. The relation $\preceq$ is a preorder on triangular sets, and it induces a partial order on the quotient TriangularSet$/\sim$.

The polynomial order is well-founded whenever σ is well-ordered. For triangular sets, however, well-foundedness requires the stronger assumption that σ is **finite**.

Proposition 1 (Well-ordering property of triangular sets) *If σ is finite, then there is no infinite strictly decreasing sequence of triangular sets.*

```
instance [Finite σ] : WellFoundedLT (TriangularSet σ R)
```

This well-ordering property is a key ingredient in proving the termination of the algorithms for computing characteristic sets and zero decompositions.

Definition 6 (Ascending Set). *A triangular set $AS = \{as_1, \ldots, as_r\}$ is called an* ascending set *if, for every pair i, j with $i < j$, the polynomial as_j is reduced with respect to as_i.*

```
class AscendingSetTheory (σ R : Type*) [CommSemiring R] [DecidableEq R] [LinearOrder σ] where
  /-- The reduction relation used to define the ascending property. -/
  protected reducedTo' : MvPolynomial σ R → MvPolynomial σ R → Prop
  decidableReducedTo : DecidableRel reducedTo' := by infer_instance
  /-- A key property linking the ascending set structure to the initial.
  If 'S' is an ascending set, the initial of any non-constant element in 'S'
  must be reduced with respect to 'S'. -/
  protected initial_reducedToSet_of_max_vars_ne_bot : ∀ {S : TriangularSet σ R} {i : ℕ},
    (∀ {i j}, i < j → j < S.length → reducedTo' (S j) (S i)) →
    (S i).vars.max ≠ ⊥ → (S i).initial.reducedToSet S

def isAscendingSet (S : TriangularSet σ R) : Prop :=
  ∀ {i j}, i < j → j < S.length → AscendingSetTheory.reducedTo' (S j) (S i)
```

The essential property of an ascending set AS is that, for every non-constant polynomial $p \in AS$, its initial I_p is reduced with respect to AS itself. This property plays a crucial role in the termination proof of the zero decomposition algorithm.

Definition 7 (Basic Set). *A basic set BS of a polynomial set PS is the minimal ascending set contained in PS.*

```
class HasBasicSet (σ R : Type*) [CommSemiring R] [DecidableEq R] [LinearOrder σ]
    extends AscendingSetTheory σ R where
  /-- Computes a Basic Set from a list of polynomials. -/
  basicSet : List (MvPolynomial σ R) → TriangularSet σ R
  basicSet_isAscendingSet (l : List (MvPolynomial σ R)) : (basicSet l).isAscendingSet
  basicSet_subset (l : List (MvPolynomial σ R)) : ∀ {c}, c ∈ basicSet l → c ∈ l
  basicSet_minimal (l : List (MvPolynomial σ R)) :
      ∀ {S}, S.isAscendingSet → (∀ {p}, p ∈ S → p ∈ l) → basicSet l ≤ S
  /-- Order reduction property: appending a reduced element strictly decreases the basic set order.
  Crucial for proving termination of zero decomposition. -/
  basicSet_append_lt_of_exists_reducedToSet : ∀ {l1 l2 : List (MvPolynomial σ R)},
      (∃ p ∈ l2, p ≠ 0 ∧ p.reducedToSet (basicSet l1)) → basicSet (l2 ++ l1) < basicSet l1
```

The next proposition is used to show that, throughout the algorithm, the order of the basic set strictly decreases.

Proposition 2 *Let p be a polynomial reduced with respect to the basic set of PS. Then the basic set of $PS \cup \{p\}$ is strictly smaller than that of PS under the prescribed ordering.*

```
theorem basicSet_append_lt_of_exists_reducedToSet
    (h : ∃ p ∈ l2, p ≠ 0 ∧ p.reducedToSet l1.basicSet) : (l2 ++ l1).basicSet < l1.basicSet
```

4 Polynomial Pseudo-Division

Theorem 3 (Pseudo-Remainder) *Let $f, g \in R[X_i]_{i\in\sigma}$, and suppose that f is a non-constant polynomial with initial $I = \mathrm{init}(f)$. Then there exist $s \in \mathbb{N}$, $q \in R[X_i]_{i\in\sigma}$, and $r \in R[X_i]_{i\in\sigma}$ such that*

$$I^s \cdot g = q \cdot f + r,$$

where r is reduced with respect to f. The polynomial r is called the pseudo-remainder *of g with respect to f, denoted by*

$$r = \mathrm{Remdr}(g/f).$$

The following procedure gives an algorithm for computing s, q, and r.

Require: polynomial g, nonzero polynomial f with main variable X_m
Ensure: pseudo-remainder r of g divided by f
1: $s := 0$, $q := 0$, $r := g$
2: $I := \mathrm{init}_m(f)$
3: $d_f := \deg_m(f)$
4: $d_r := \deg_m(r)$
5: **while** $d_r \geq d_f$ **do**
6: $\quad t := \mathrm{init}_m(r) \cdot X_m^{d_r - d_f}$
7: $\quad r := I \cdot r - t \cdot f$
8: $\quad q := I \cdot q + t$
9: $\quad s := s + 1$
10: $\quad d_r := \deg_m(r)$
11: **end while**
12: **return** r

The key step is Step 6, where r is multiplied by I, while f is multiplied by $\mathrm{init}_m(r)$, so that the leading term of r is canceled. The factor $X_m^{d_r - d_f}$ aligns the degrees of r and f in the main variable X_m. Consequently, the degree of r strictly decreases at each iteration.

The general `pseudo` function is defined over a field: it automatically detects the main variable and also handles constant divisors. Hence, from this point on, we assume that R is a field.

```
variable {R σ : Type*} [Field R] [DecidableEq R] [LinearOrder σ] (g f : MvPolynomial σ R)

def pseudo : PseudoResult (MvPolynomial σ R) :=
  if f = 0 then ⟨0, 0, g⟩
  else
    match f.vars.max with
    | ⊥ => ⟨0, (f.coeff 0)⁻¹ · g, 0⟩
    -- pseudoOf implements the pseudo-division algorithm described above
    | some m => g.pseudoOf m f

theorem pseudo_equation : f.initial ^ (g.pseudo f).exponent * g = (g.pseudo f).quotient * f + (g.pseudo
     f).remainder
```

This allows us to define the pseudo-remainder with respect to a triangular set.

Theorem 4 (Pseudo-remainder with respect to a triangular set) *Let $S = (s_1, \ldots, s_k)$ be a triangular set and let $g \in R[X_i]_{i \in \sigma}$. Then there exist*

$$r, q_1, \ldots, q_k \in R[X_i]_{i \in \sigma}$$

and $e_1, \ldots, e_k \in \mathbb{N}$ such that

$$\left(\prod_{i=1}^{k} \mathrm{init}(s_i)^{e_i} \right) g = \sum_{i=1}^{k} q_i s_i + r,$$

where r is reduced with respect to S. The polynomial r is called the pseudo-remainder *of g with respect to S and is denoted by*

$$r = \mathrm{Remdr}(g/S).$$

Proof. This remainder can be computed by recursively performing pseudo-division, starting from the last element of S. First, we perform pseudo-division of g by s_k, obtaining e_k, q_k, and r_k. We then perform pseudo-division of r_k by s_{k-1}, obtaining e_{k-1}, q_{k-1}, and r_{k-1}. Continuing in this way, we eventually obtain a final remainder, r_0. This polynomial r_0 is the pseudo-remainder of g with respect to S.

```
def setPseudo.go (f : ℕ → MvPolynomial σ R) (fuel : ℕ) (es : List ℕ)
    (qs : List (MvPolynomial σ R)) (r : MvPolynomial σ R) : SetPseudoResult (MvPolynomial σ R) :=
  if fuel = 0 then ⟨es, qs, r⟩
  else
    let p := r.pseudo (f (fuel - 1))
    let es' := p.exponent :: es
```

```
    let qs' := p.quotient :: qs.map (· * (f (fuel - 1)).initial ^ p.exponent)
    let r' := p.remainder
    go f (fuel - 1) es' qs' r'

def setPseudo (S : TriangularSet σ R) : SetPseudoResult (MvPolynomial σ R) :=
  setPseudo.go S S.length [] [] g

theorem setPseudo_equation : letI result := g.setPseudo S
    (Π i : Fin result.exponents.length, (S i).initial ^ result.exponents[i]) * g
    = (Σ i : Fin result.quotients.length, result.quotients[i] * S i) + result.remainder
```

To decouple the algorithm from the correctness proofs, the notion of remainder is defined as a proposition, not tied to a specific implementation:

```
def IsRemainder (r g f : MvPolynomial σ R) : Prop :=
  r.reducedTo f ∧ ∃ (s : ℕ) (q : MvPolynomial σ R), f.initial ^ s * g = q * f + r

def IsSetRemainder (r g : MvPolynomial σ R) (S : TriangularSet σ R) : Prop := r.reducedToSet S ∧
  ∃ (es : List ℕ) (qs : List (MvPolynomial σ R)), (es.length = S.length ∧ qs.length = S.length) ∧
    (∏ i : Fin es.length, (S i).initial ^ es[i]) * g = (Σ i : Fin qs.length, qs[i] * S i) + r
```

5 Characteristic Set

For a polynomial set PS, we write $\mathrm{Zero}(PS)$ for its vanishing set, that is, the set of all points at which every polynomial in PS evaluates to zero. We also use the abbreviation

$$\mathrm{Zero}(CS/IP) := \mathrm{Zero}(CS) \setminus \mathrm{Zero}(IP),$$

namely the set of common zeros of CS at which the initial product IP does not vanish. In Lean, these are defined as $MvPolynomial.vanishingSet$ and $MvPolynomial.vanishingSet'$, which coincide with the zero locus of the ideal generated by the set.

Definition 8 (Characteristic Set). *A triangular set CS is a* characteristic set *of PS if* $\forall p \in PS,\ \mathrm{Remdr}(p/CS) = 0$ *and* $\mathrm{Zero}(PS) \subseteq \mathrm{Zero}(CS)$.

```
def TriangularSet.IsCharacteristicSet {α : Type*} [Membership (MvPolynomial σ R) α] (K : Type*)
    [CommSemiring K] [Algebra R K] (CS : TriangularSet σ R) (a : α) : Prop :=
  (∀ g ∈ a, (0 : MvPolynomial σ R).IsSetRemainder g CS) ∧ vanishingSet K a ⊆ vanishingSet K CS
```

Theorem 5 (Well-ordering Principle) *Let $CS = (c_1, \ldots, c_r)$ be a characteristic set of PS, and let $IP = \prod_{i=1}^{r} I_{c_i}$ be the initial product of CS. Then* $\mathrm{Zero}(CS/IP) \subseteq \mathrm{Zero}(PS)$, *and consequently* $\mathrm{Zero}(CS/IP) = \mathrm{Zero}(PS/IP)$. *Moreover,*

$$\mathrm{Zero}(PS) = \mathrm{Zero}(CS/IP) \cup \left(\bigcup_{p \in CS} \mathrm{Zero}(PS \cup \{I_p\}) \right). \tag{1}$$

```
theorem vanishingSet_diff_initialProd_subset
    (h : (∀ g ∈ PS, (0 : MvPolynomial σ R).IsSetRemainder g CS)) :
    vanishingSet K CS \ vanishingSet' K (initialProd CS.toFinset) ⊆ vanishingSet K PS

theorem vanishingSet_diff_initialProd_eq (h : CS.IsCharacteristicSet K PS) :
    vanishingSet K CS \ vanishingSet' K (initialProd CS.toFinset) =
      vanishingSet K PS \ vanishingSet' K (initialProd CS.toFinset)

theorem vanishingSet_decomposition (h : CS.IsCharacteristicSet K PS) : vanishingSet K PS =
      vanishingSet K CS \ vanishingSet' K (initialProd CS.toFinset) ∪
      (∪ p ∈ CS, vanishingSet K PS ∩ vanishingSet' K p.initial)
```

Theorem 6 *A characteristic set CS can be constructed in a finite number of steps using the following algorithm:*

Require: $PS = (p_1, \ldots, p_s)$
Ensure: a Characteristic set $CS = (c_1, \ldots, c_t)$ for PS,

```
PS' := PS
repeat
    RS := ∅
    BS := BasicSet(PS')
    for each {p} in PS' do
        r := Remdr(p, BS)
        if r ≠ 0 then
            RS := RS ∪ {r}
        end if
    end for
    PS' := PS ∪ RS ∪ BS
until RS = ∅
return CS := BS
```

```
def characteristicSet.go [Finite σ] (l₀ l : List (MvPolynomial σ R))
    : TriangularSet σ R :=
  let BS := l.basicSet.val
  let lBS := BS.toList
  let RS : List _ := ((l \ lBS).map fun p ↦ (p.setPseudo BS).remainder).filter (· ≠ 0)
  if RS = [] then BS
  else go l₀ (l₀ ++ RS ++ lBS)
  termination_by l.basicSet

def characteristicSet [Finite σ] (l) : TriangularSet σ R := characteristicSet.go l l
```

Theorem 7 (Zero Decomposition Theorem) *For a polynomial system PS, there exists a finite collection of triangular sets*

$$\mathcal{ZD} = \{CS_j\}_j$$

such that

$$\mathrm{Zero}(PS) = \bigcup_j \mathrm{Zero}(CS_j / IP_j), \tag{2}$$

where IP_j denotes the initial product of CS_j. Moreover, for every j and every $p \in PS$,

$$\mathrm{Remdr}(p / CS_j) = 0.$$

```
def zeroDecomposition [Finite σ] (l : List (MvPolynomial σ R)) : List (TriangularSet σ R) :=
  let CS := l.characteristicSet
  let subDecomp := (CS.toList.filter fun p ↦ p.vars.max ≠ ⊥).attach.map
    fun ⟨p, _⟩ ↦ zeroDecomposition (p.initial :: CS.toList ++ l)
  CS :: subDecomp.flatten
  termination_by l.basicSet
  decreasing_by ...
```

```
theorem vanishingSet_eq_zeroDecomposition_union :
    vanishingSet K l = ∪ CS ∈ l.zeroDecomposition,
      vanishingSet K CS \ vanishingSet' K (initialProd CS.toFinset)

theorem zero_isSetRemainder_of_mem_zeroDecomposition :
    ∀ CS ∈ l.zeroDecomposition, ∀ g ∈ l, (0 : MvPolynomial σ R).IsSetRemainder g CS
```

For brevity, we omit the proof of Theorem 7 and provide an example illustrating the zero-decomposition procedure.

Example 2. Consider $PS = \{x^2 + y^2 - 1,\ xy\}$ over the rational number field, where $x < y$. The characteristic set of PS can be taken as $CS = \{x^3 - x,\ xy\}$. The initials of the two polynomials in CS are 1 and x, so the only nontrivial initial is x. Thus (1) gives

$$\mathrm{Zero}(PS) = \mathrm{Zero}(CS/\{x\}) \cup \mathrm{Zero}(PS \cup \{x\}).$$

Since $x \neq 0$ on the first branch, $xy = 0$ gives $y = 0$, and $x^3 - x = 0$ gives $x^2 - 1 = 0$. Hence $\mathrm{Zero}(CS/\{x\}) = \mathrm{Zero}(x^2 - 1, y)$. On the second branch, $PS \cup \{x\}$ has characteristic set $\{x, y^2 - 1\}$, whose initials are all 1, so the recursion stops. Therefore

$$\mathrm{Zero}(PS) = \mathrm{Zero}(x^2 - 1, y) \cup \mathrm{Zero}(x, y^2 - 1) = \{(\pm 1, 0), (0, \pm 1)\}.$$

6 Conclusions

We formalized the Wu-Ritt characteristic set method in Lean 4, including polynomial orders, pseudo-division, triangular and ascending sets, characteristic sets, and zero decomposition. We proved termination and correctness, including the well-ordering principle and the zero-set decomposition theorem.

The development consists of about 3000 lines of `sorry`-free Lean 4 code based on Mathlib 4 (v4.29.0), and is available under the Apache-2.0 license at `https://github.com/WuProver/lean_characteristic_set`. Since `MvPolynomial` is noncomputable, our current focus is on correctness proofs. Future work includes executable extraction and extensions to differential polynomials.

Acknowledgments. This work was supported by the National Key R&D Program of China 2023YFA1009401 and the Strategic Priority Research Program of Chinese Academy of Sciences under Grant XDA0480502. Junyu Guo is encouraged and supported by his supervisor, Xishun Zhao, to work on this project.

References

1. Génevaux, J.D., Narboux, J., Schreck, P.: Formalization of Wu's simple method in Coq. In: Jouannaud, J.P., Shao, Z. (eds.) Certified Programs and Proofs. pp. 71–86. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
2. Guo, J., Shen, H., Liu, J., Zhi, L.: Formalizing Gröbner basis theory in Lean (2026), `https://arxiv.org/abs/2602.12772`
3. de Moura, L., Ullrich, S.: The Lean 4 theorem prover and programming language. In: Automated Deduction – CADE 28: 28th International Conference on Automated Deduction, Virtual Event, July 12–15, 2021, Proceedings. p. 625–635. Springer-Verlag, Berlin, Heidelberg (2021). `https://doi.org/10.1007/978-3-030-79876-5_37`
4. The Coq Development Team: The Coq Proof Assistant Reference Manual, Version 8.3. LogiCal Project (2010)
5. The mathlib Community: The Lean mathematical library. In: Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs. p. 367–381. CPP 2020, Association for Computing Machinery, New York, NY, USA (2020). `https://doi.org/10.1145/3372885.3373824`
6. Wu, W.T.: Basic principles of mechanical theorem proving in elementary geometries. Journal of Automated Reasoning **2**(3), 221–252 (1986)
7. Wu, W.T.: Mathematics mechanization: mechanical geometry theorem-proving, mechanical geometry problem-solving, and polynomial equations-solving. Kluwer Academic Publishers (2001)