The Computational Applications and Theory of Boolean Symmetric Polynomials

Justin Stoncius

8/18/2025

Abstract

SSI Abstract: This is my working draft of a paper introducing various symmetric polynomial models of computation. Much of the paper is simply a generative AI adaptation of my notes, so it may contain errors or fabrications, however, the first three sections were given sufficient human attention. I just wanted to put something together for this SSI application, it is definitely not ready for publication. I've been trying to find a good definition for the HSPSM but I keep confusing myself, one attempt is done here.

We present a novel model of computation with purely algebraic primitives built from symmetric polynomials over GF(2). With computation described entirely in terms of elementary and complete homogeneous symmetric polynomials, we further develop from this model a characterization of computational complexity in terms of polynomial degree, composition depth, and symmetry-breaking of computations within the model, allowing for an elegant natural algebraic interpretation of various computational structures and results. We establish the universality of this model and characterize the relationship between our measure and standard measures of complexity. To illustrate our points, we introduce the Homogeneous Symmetric Polynomial State Machine (HSPSM) as a concrete implementation of our model, demonstrating the remarkable discovery that the simple recurrence relation $h_{k,j}(x) = h_{k,j-1}(x) \oplus (x_j \cdot h_{k-1,j-1}(x))$ is a sufficient update rule to provide a uniform, sequential, universal model of computation.

1 Introduction

This paper introduces a computational model based on symmetric polynomials over GF(2), where complexity is characterized by the degree of symmetry-breaking required to solve a problem. Despite their applications in specific complexity proofs [**?**, 5, **?**], symmetric polynomials have not been systematically studied as computing primitives. We show that this algebraic perspective on computation yields several new insights into computational complexity.

1.1 Main Contributions

Our key contributions are:

1. A computational model based on symmetric polynomials over GF(2) that is provably universal, capable of simulating arbitrary Turing machines.
2. The Homogeneous Symmetric Polynomial State Machine (HSPSM), which implements universal computation using only the recurrence relation $h_{k,j}(x) = h_{k,j-1}(x) \oplus (x_j \cdot h_{k-1,j-1}(x))$.
3. Complexity measures specific to our model, including Symmetric Polynomial Composition Complexity (SPCC) and Degree Complexity (DC), with proven relationships to standard complexity classes.
4. A characterization of complexity classes in terms of symmetry-breaking operations and polynomial degrees, providing a natural algebraic interpretation of complexity separations.
5. A connection between binomial coefficients modulo 2 and computational complexity, demonstrating how Lucas' Theorem relates to algorithm efficiency.

1.2 Organization

The remainder of this paper develops our framework progressively: we begin with the algebraic foundations of Boolean functions and symmetric polynomials over GF(2) (Sections 2-3), then introduce computational primitives based on partial and masked operations (Section 4). The core of our contribution—the Homogeneous Symmetric Polynomial State Machine—is presented in Section 5, where we prove its computational universality. We then explore the complexity-theoretic implications (Sections 6-9), establishing connections between polynomial degree, symmetry-breaking operations, and standard complexity classes. These connections reveal a natural algebraic interpretation of why certain computational problems require greater resources than others.

2 Algebraic Foundations

We briefly review the relevant algebraic structures that form the basis of our computational model, focusing on Boolean functions and symmetric polynomials over GF(2).

2.1 Boolean Functions and Symmetric Polynomials

Definition 1 (Boolean Functions)**.** *The set of all n-variable Boolean functions, denoted $\mathcal{B}_n = \{f : \{0,1\}^n \to \{0,1\}\}$, can be represented as polynomials in the ring $\mathrm{GF}(2)[x_1, \ldots, x_n]/(x_1^2 - x_1, \ldots, x_n^2 - x_n)$, where the operations $\oplus$ and $\cdot$ correspond to polynomial addition and multiplication, respectively.*

Definition 2 (Multilinear Representation)**.** *Every Boolean function has a unique multilinear polynomial representation over $\mathrm{GF}(2)$, known as the Algebraic Normal Form, the Zhegalkin polynomial, or the Reed-Muller expansion:*

$$f(x_1, \ldots, x_n) = \bigoplus_{S \subseteq \{1,\ldots,n\}} a_S \prod_{i \in S} x_i \tag{1}$$

where $a_S \in \{0,1\}$ are the coefficients.

Definition 3 (Hamming Weight)**.** *For a binary string $x = (x_1, \ldots, x_n) \in \{0,1\}^n$, the Hamming weight $\mathrm{wt}(x)$ is the number of 1's in x: $\mathrm{wt}(x) = \sum_{i=1}^n x_i$.*

Definition 4 (Symmetric Function)**.** *A Boolean function $f \in \mathcal{B}_n$ is symmetric if for any permutation $\sigma \in S_n$ and any input $x = (x_1, \ldots, x_n) \in \{0,1\}^n$:*

$$f(x_1, \ldots, x_n) = f(x_{\sigma(1)}, \ldots, x_{\sigma(n)}) \tag{2}$$

The set of all symmetric Boolean functions on n variables is denoted $\mathcal{S}_n$.

Proposition 1. *A Boolean function $f \in \mathcal{B}_n$ is symmetric if and only if its output depends only on the $n+1$ possible Hamming weights of the input.*

Proof. ($\Rightarrow$) If f is symmetric, then it gives the same output for all inputs with the same number of 1s. A Boolean input of length n can have a Hamming weight from 0 to n, giving $n+1$ total possibilities. Therefore, f depends only on the $n+1$ possible Hamming weights.

($\Leftarrow$) If f depends only on the Hamming weight, then any permutation of the input preserves the Hamming weight and thus the output of f.

□

Our framework primarily utilizes two types of symmetric polynomials:

Definition 5 (Elementary Symmetric Polynomials)**.** *For variables $x_1, \ldots, x_n$, the k-th elementary symmetric polynomial e_k over $\mathrm{GF}(2)$ is defined as:*

$$e_k(x_1, \ldots, x_n) = \bigoplus_{1 \le i_1 < i_2 < \cdots < i_k \le n} (x_{i_1} \cdot x_{i_2} \cdot \ldots \cdot x_{i_k}) \tag{3}$$

with the convention that $e_0(x_1, \ldots, x_n) = 1$.

Definition 6 (Complete Homogeneous Symmetric Polynomials)**.** *For variables $x_1, \ldots, x_n$, the k-th complete homogeneous symmetric polynomial h_k over* GF(2) *is defined as:*

$$h_k(x_1, \ldots, x_n) = \bigoplus_{1 \le i_1 \le i_2 \le \cdots \le i_k \le n} (x_{i_1} \cdot x_{i_2} \cdot \ldots \cdot x_{i_k}) \tag{4}$$

with the convention that $h_0(x_1, \ldots, x_n) = 1$.

Remark 1. *If it is not immediately clear, the difference lies in their indexing: elementary symmetric polynomials use strictly increasing indices ($i_1 < i_2 < \cdots < i_k$), while complete homogeneous symmetric polynomials use merely non-decreasing indices ($i_1 \le i_2 \le \cdots \le i_k$), allowing repetition like multisets.*

2.2 Key Properties in GF(2)

The behavior of symmetric polynomials over GF(2) exhibits special properties that are central to our computational model:

Lemma 2 (Binomial Coefficient Relation)**.** *For an input x with Hamming weight* $\mathrm{wt}(x) = w$*, the value of $e_k(x)$ equals $\binom{w}{k} \bmod 2$.*

Proof. When exactly w variables are set to 1, e_k counts the number of k-element subsets where all variables are 1. There are $\binom{w}{k}$ such subsets. Since we work over GF(2), the result is $\binom{w}{k} \bmod 2$. □

Theorem 3 (Elementary-Homogeneous Relationship)**.** *The elementary and complete homogeneous symmetric polynomials over* GF(2) *satisfy:*

$$\bigoplus_{i=0}^{m} e_i(x_1, \ldots, x_n) h_{m-i}(x_1, \ldots, x_n) = 0 \tag{5}$$

for all $m > 0$ and any number of variables n.

Proof. It is known that elementary and complete homogeneous symmetric polynomials are, in general, related by the formula $\sum_{i=0}^{m} (-1)^i e_i h_{m-i} = 0$ for $m > 0$.

In GF(2), $-1 \equiv 1 \pmod 2$, giving us $\bigoplus_{i=0}^{m} e_i h_{m-i} = 0$. □

These properties form the foundation of our computational model. The simplifications provided by working over GF(2) enable elegant representations of computational processes using symmetric polynomials, as we will demonstrate in subsequent sections.

k	$e_k(x_1, x_2, x_3)$	$h_k(x_1, x_2, x_3)$
0	1	1
1	$x_1 + x_2 + x_3$	$x_1 + x_2 + x_3$
2	$x_1x_2 + x_1x_3 + x_2x_3$	$x_1x_2 + x_1x_3 + x_2x_3 + x_1^2 + x_2^2 + x_3^2$
3	$x_1x_2x_3$	$x_1x_2x_3 + x_1^2x_2 + x_1x_2^2 + x_1^2x_3 + x_1x_3^2 + x_2^2x_3 + x_2x_3^2 + x_1^3 + x_2^3 + x_3^3$
4	0	$x_1^2x_2^2+x_1^2x_3^2+x_2^2x_3^2+x_1^2x_2x_3+x_1x_2^2x_3+x_1x_2x_3^2+x_1^3x_2+x_1^3x_3+x_1x_2^3+x_2^3x_3+x_1x_3^3+x_2x_3^3+x_1^4+x_2^4+x_3^4$
5	0	$x_1^2x_2^2x_3+x_1^2x_2x_3^2+x_1x_2^2x_3^2+x_1^3x_2x_3+x_1x_2^3x_3+x_1x_2x_3^3+x_1^3x_2^2+x_1^2x_2^3+x_1^3x_3^2+x_1^2x_3^3+x_2^3x_3^2+x_2^2x_3^3+x_1^4x_2+x_1^4x_3+x_1x_2^4+x_2^4x_3+x_1x_3^4+x_2x_3^4+x_1^5+x_2^5+x_3^5$

Table 1: Elementary and Complete Homogeneous Symmetric Polynomials on Three Variables for $0 \leq k \leq 3$ (General Case)

k	$e_k(x_1, x_2, x_3)$	$h_k(x_1, x_2, x_3)$
0	1	1
1	$x_1 \oplus x_2 \oplus x_3$	$x_1 \oplus x_2 \oplus x_3 = e_1$
2	$x_1x_2 \oplus x_1x_3 \oplus x_2x_3$	$e_2 \oplus e_1 = (x_1x_2 \oplus x_1x_3 \oplus x_2x_3) \oplus (x_1 \oplus x_2 \oplus x_3)$
3	$x_1x_2x_3$	$e_3 \oplus e_1 = x_1x_2x_3 \oplus x_1 \oplus x_2 \oplus x_3$
4	0	$e_2 = x_1x_2 \oplus x_1x_3 \oplus x_2x_3$
5	0	$e_3 \oplus e_1 = x_1x_2x_3 \oplus x_1 \oplus x_2 \oplus x_3$

Table 2: Elementary and Complete Homogeneous Symmetric Polynomials over GF(2), demonstrating the simplification.

3 Symmetric Polynomial Operations

Having established the fundamental properties of symmetric polynomials over GF(2), we now develop their application to computation. We first show how basic Boolean operations can be implemented using symmetric polynomials, then build towards a complete computational model.

3.1 Basic Boolean Operations

Lemma 4 (Boolean Operations with Elementary Symmetric Polynomials)**.** *The basic Boolean operations can be implemented using elementary symmetric polynomials:*

$$x \wedge y = e_2(x, y) \tag{6}$$
$$x \oplus y = e_1(x, y) \tag{7}$$
$$x \vee y = e_1(x, y) \oplus e_2(x, y) \tag{8}$$
$$\neg x = 1 \oplus e_1(x) = e_0 \oplus e_1(x) \tag{9}$$

Proof. These identities follow directly from the definitions:

$$e_1(x, y) = x \oplus y \tag{10}$$
$$e_2(x, y) = x \cdot y \tag{11}$$

For the OR operation, note that $x \vee y = x \oplus y \oplus (x \cdot y) = e_1(x, y) \oplus e_2(x, y)$.
For negation, recall that $e_0 = 1$, so $e_0 \oplus e_1(x) = 1 \oplus x = \neg x$. □

Lemma 5 (Boolean Operations with Homogeneous Symmetric Polynomials)**.** *The basic Boolean operations can also be implemented using homogeneous symmetric polynomials:*

$$x \wedge y = h_2(x, y) \tag{12}$$
$$x \oplus y = h_1(x, y) \oplus h_2(x, y) \tag{13}$$
$$\neg x = 1 \oplus h_1(x) \tag{14}$$

Proof. For two variables, $h_1(x, y) = x \oplus y \oplus (x \cdot y)$ and $h_2(x, y) = x \cdot y$. Therefore:

$$h_1(x, y) \oplus h_2(x, y) = (x \oplus y \oplus (x \cdot y)) \oplus (x \cdot y) = x \oplus y \tag{15}$$

The other identities follow similarly. □

These lemmas establish that both elementary and homogeneous symmetric polynomials provide a complete basis for Boolean operations.

3.2 Compositions of Symmetric Polynomials

With basic operations established, we now formalize the concept of symmetric polynomial compositions.

Definition 7 (Symmetric Polynomial Composition)**.** *A symmetric polynomial composition C over variables $x_1, \ldots, x_n$ is defined recursively:*

1. ***Base case**: Each symmetric polynomial $e_k(x_1, \ldots, x_n)$ or $h_k(x_1, \ldots, x_n)$, where $0 \leq k \leq n$, is a symmetric polynomial composition.*
2. ***Recursive case**: If P and Q are symmetric polynomial compositions, then $P \oplus Q$ and $P \cdot Q$ are also symmetric polynomial compositions.*

Theorem 6 (Symmetric Function Representation)**.** *Any symmetric Boolean function $f : \{0, 1\}^n \to \{0, 1\}$ can be represented as a composition of elementary symmetric polynomials over* GF(2)*.*

Proof. A symmetric Boolean function depends only on the Hamming weight of its input. From our Binomial Coefficient Relation lemma, when x has Hamming weight w, $e_k(x) = \binom{w}{k} \bmod 2$.

The values of $e_0(x), e_1(x), \ldots, e_n(x)$ uniquely determine the Hamming weight of x, as the mapping $w \mapsto (\binom{w}{0} \bmod 2, \binom{w}{1} \bmod 2, \ldots, \binom{w}{n} \bmod 2)$ is injective.

Since any symmetric Boolean function is completely determined by the Hamming weight of its input, it can be expressed as a composition of elementary symmetric polynomials using $\oplus$ and $\cdot$ operations. □

Definition 8 (Size of Composition)**.** *The size of a symmetric polynomial composition, denoted $|C|$, is the number of symmetric polynomial operations (elementary or homogeneous polynomials, XOR, and AND operations) that appear in the composition.*

3.3 Circuit Model

The use of symmetric polynomials for representing Boolean functions naturally leads to a circuit model.

Definition 9 (Symmetric Polynomial Circuit)**.** *A symmetric polynomial circuit C on n input variables is a directed acyclic graph where:*

1. *Each input node is labeled with a variable x_i for $i \in \{1, \ldots, n\}$ or a constant 0 or 1.*

2. Each internal node computes either:

- *A symmetric polynomial $e_k(z)$ or $h_k(z)$, where z are the inputs to the node, or*
- *An XOR ($\oplus$) or AND ($\cdot$) of its inputs.*

3. There is a single output node.

Definition 10 (Depth of Symmetric Polynomial Circuit)**.** *The depth of a symmetric polynomial circuit is the length of the longest path from any input node to the output node.*

Theorem 7 (Boolean Circuit Simulation)**.** *For any Boolean circuit B with g gates, there exists a symmetric polynomial circuit C that computes the same function, with $|C| = O(g)$.*

Proof. We can directly translate each gate in the Boolean circuit to its corresponding implementation using symmetric polynomials:

1. For an AND gate with inputs a and b, we create a node computing $e_2(a, b)$. 2. For an XOR gate with inputs a and b, we create a node computing $e_1(a, b)$. 3. For a NOT gate with input a, we create a node computing $1 \oplus e_1(a)$.

Each gate in the Boolean circuit is replaced by exactly one node in the symmetric polynomial circuit, resulting in $|C| = O(g)$. □

Theorem 8 (Symmetric Polynomial Circuit Simulation)**.** *For any symmetric polynomial circuit C of size s with bounded degree symmetric polynomials, there exists a Boolean circuit B that computes the same function, with $O(s \cdot n^2)$ gates.*

Proof. The XOR and AND operations in C can be directly implemented in B. For a symmetric polynomial $e_k(z)$, we implement it using the definition:

$$e_k(z) = \bigoplus_{1 \le i_1 < i_2 < \cdots < i_k \le n} (z_{i_1} \cdot z_{i_2} \cdot \ldots \cdot z_{i_k}) \tag{16}$$

This requires computing $\binom{n}{k}$ products, each using $k-1$ AND gates, followed by $\binom{n}{k} - 1$ XOR gates to combine them. For constant k, this requires $O(n^k)$ gates, which is $O(n^2)$ when $k \le 2$.

The simulation of h_k follows a similar approach. The total gate count in B is thus $O(s \cdot n^2)$ for bounded degree symmetric polynomials. □

These theorems establish that symmetric polynomial circuits are computationally equivalent to Boolean circuits, with polynomial overhead in the typical case.

3.4 Masking Operations for Symmetry-Breaking

While symmetric polynomial circuits can efficiently implement symmetric functions, they cannot directly represent non-symmetric functions. To overcome this limitation, we introduce symmetry-breaking through variable masking.

Definition 11 (Bit Mask)**.** *A bit mask m of length n is a binary string $(m_1, m_2, \ldots, m_n) \in \{0,1\}^n$ that selects a subset of positions from an input.*

Definition 12 (Masked Input)**.** *For an input $x = (x_1, \ldots, x_n)$ and a mask $m = (m_1, \ldots, m_n)$, the masked input $m \odot x$ is defined as:*

$$(m \odot x)_i = m_i \cdot x_i \tag{17}$$

Definition 13 (Masked Symmetric Polynomial)**.** *For an input x, mask m, and degree k, the masked symmetric polynomial is:*

$$e_k[m](x) = e_k(m \odot x) \tag{18}$$

Similarly, for homogeneous polynomials:

$$h_k[m](x) = h_k(m \odot x) \tag{19}$$

Theorem 9 (Universality of Masked Symmetric Polynomials)**.** *Any Boolean function $f \in \mathcal{B}_n$ can be expressed as a composition of masked elementary symmetric polynomials over* GF(2)*.*

Proof. From the algebraic normal form (ANF) representation, any Boolean function can be written as

$$f(x) = \bigoplus_{S \subseteq \{1,\ldots,n\}} a_S \prod_{i \in S} x_i \tag{20}$$

where $a_S \in \{0, 1\}$ are the coefficients.

Each monomial term $\prod_{i \in S} x_i$ can be expressed as $e_{|S|}[m_S](x)$, where m_S is the mask that has 1s exactly at positions indexed by S and 0s elsewhere. This mask selects precisely the variables in the monomial.

Therefore, $f(x) = \bigoplus_{S \subseteq \{1,\ldots,n\}} a_S \cdot e_{|S|}[m_S](x)$, which is a composition of masked elementary symmetric polynomials. □

This theorem establishes that our model with masked symmetric polynomials is computationally universal, capable of expressing any Boolean function.

Definition 14 (Masked Symmetric Polynomial Circuit)**.** *A masked symmetric polynomial circuit extends the symmetric polynomial circuit definition by allowing each internal node to compute a masked symmetric polynomial $e_k[m](z)$ or $h_k[m](z)$.*

Proposition 10 (Variable Fan-in with Masks)**.** *Masked symmetric polynomials can efficiently implement gates with arbitrary fan-in:*

1. *An AND gate with k inputs can be implemented as $e_k[1^k](z)$.*
2. *An OR gate with k inputs can be implemented as $1 \oplus e_k[1^k](\neg z)$, where $\neg z$ represents the bitwise negation of inputs.*
3. *A THRESHOLD-t gate, which outputs 1 if at least t of its k inputs are 1, can be implemented as $\bigoplus_{i=t}^{k} e_i[1^k](z)$. TODO: Need to check this*

Proof. 1. The elementary symmetric polynomial $e_k[1^k](z)$ outputs 1 if and only if all k inputs are 1, which is precisely the AND function.

2. Using De Morgan's law, $z_1 \vee z_2 \vee \cdots \vee z_k = \neg(\neg z_1 \wedge \neg z_2 \wedge \cdots \wedge \neg z_k)$. The expression $e_k[1^k](\neg z)$ computes the AND of the negated inputs, and taking $1 \oplus e_k[1^k](\neg z)$ negates the result, giving us the OR function.

3. A THRESHOLD-t gate outputs 1 if at least t of its inputs are 1. This can be computed by checking if any subset of size $i \geq t$ has all 1s, which is expressed as $\bigoplus_{i=t}^{k} e_i[1^k](z)$. □

3.5 Partial Masks and Sequential Computation

We now introduce a special case of masking that enables efficient sequential computation.

Definition 15 (Partial Mask)**.** *A partial mask of length n up to position j is the mask $1^j 0^{n-j}$, which selects the first j positions of the input.*

Definition 16 (Partial Symmetric Polynomials)**.** *For a string $x = (x_1, \ldots, x_n)$, the partial elementary symmetric polynomial of degree k up to position j is:*

$$e_{k,j}(x) = e_k[1^j 0^{n-j}](x) = \bigoplus_{1 \leq i_1 < i_2 < \cdots < i_k \leq j} (x_{i_1} \cdot x_{i_2} \cdot \ldots \cdot x_{i_k}) \tag{21}$$

Similarly, the partial homogeneous symmetric polynomial of degree k up to position j is:

$$h_{k,j}(x) = h_k[1^j 0^{n-j}](x) = \bigoplus_{1 \leq i_1 \leq i_2 \leq \cdots \leq i_k \leq j} (x_{i_1} \cdot x_{i_2} \cdot \ldots \cdot x_{i_k}) \tag{22}$$

The key advantage of partial symmetric polynomials is their efficient recursive computation:

Proposition 11 (Recursive Formulation). *Partial symmetric polynomials can be computed recursively:*

$$e_{k,j}(x) = e_{k,j-1}(x) \oplus (x_j \cdot e_{k-1,j-1}(x)) \tag{23}$$
$$h_{k,j}(x) = h_{k,j-1}(x) \oplus (x_j \cdot h_{k-1,j-1}(x)) \tag{24}$$

with base cases:

$$e_{0,j}(x) = h_{0,j}(x) = 1 \quad \textit{for all } j \geq 0 \tag{25}$$
$$e_{k,0}(x) = h_{k,0}(x) = 0 \quad \textit{for all } k > 0 \tag{26}$$

Proof. For $e_{k,j}(x)$, we can partition the k-element subsets of $\{1, 2, \ldots, j\}$ into those that do not contain element j and those that do. The former contribute $e_{k,j-1}(x)$ to the sum, while the latter contribute $x_j \cdot e_{k-1,j-1}(x)$.

A similar argument applies to $h_{k,j}(x)$, partitioning the multisets of size k from $\{1, 2, \ldots, j\}$ into those that do not include element j and those that do. □

Definition 17 (Partial Masking Circuit). *A partial masking circuit is a symmetric polynomial circuit where all masks are of the form $1^j 0^{n-j}$ for some $j \in \{0, 1, \ldots, n\}$. Equivalently, it uses only partial symmetric polynomials $e_{k,j}$ and $h_{k,j}$.*

Theorem 12 (Universality of Partial Masking). *Any Boolean function $f : \{0,1\}^n \to \{0,1\}$ can be computed by a partial masking circuit.*

Proof. We prove this by showing how to simulate arbitrary masks using only partial masks and additional gates.

Given an arbitrary mask $m = (m_1, m_2, \ldots, m_n)$, we can simulate $e_k[m](x)$ as follows:

1. Create a new input sequence $y = (y_1, y_2, \ldots, y_n)$ where $y_i = m_i \cdot x_i$. This requires n AND operations.
2. Compute $e_k[1^n](y) = e_{k,n}(y)$, which is equivalent to $e_k[m](x)$.

Since we have already established that arbitrary masking can express any Boolean function, and we can simulate arbitrary masks using partial masks with additional circuitry, partial masking circuits can compute any Boolean function. □

3.6 Computational History

The sequence of partial symmetric polynomial evaluations as each input bit is processed forms what we call a computational history, which captures the evolution of computation.

Definition 18 (Computational History Sequence). *The history sequence for the k-th elementary symmetric polynomial on input x is:*

$$H_k^e(x) = (e_{k,1}(x), e_{k,2}(x), \ldots, e_{k,n}(x)) \tag{27}$$

Similarly, for homogeneous polynomials:

$$H_k^h(x) = (h_{k,1}(x), h_{k,2}(x), \ldots, h_{k,n}(x)) \tag{28}$$

Definition 19 (Full Computational History). *The full computational history of input x is the collection of all history sequences:*

$$\mathcal{H}^e(x) = \{H_k^e(x) \mid 0 \leq k \leq n\} \tag{29}$$
$$\mathcal{H}^h(x) = \{H_k^h(x) \mid 0 \leq k \leq n\} \tag{30}$$

A key property of computational histories is that they fully encode the original input:

Theorem 13 (Reversible Computation). *The original input can be recovered from its computational history:*

$$x_j = e_{1,j}(x) \oplus e_{1,j-1}(x) = h_{1,j}(x) \oplus h_{1,j-1}(x) \tag{31}$$

Proof.

$$e_{1,j}(x) \oplus e_{1,j-1}(x) = \left(\bigoplus_{i=1}^{j} x_i\right) \oplus \left(\bigoplus_{i=1}^{j-1} x_i\right) \tag{32}$$

$$= x_j \tag{33}$$

The proof for homogeneous polynomials follows similarly. □

This reversibility property establishes that computational histories fully capture the information of the original input, making them suitable as state representations in a sequential computational model.

4 Homogeneous Symmetric Polynomial State Machine

4.1 End of Current Draft Revisions

Material ahead is potentially speculative, confusing, or even incorrect.

4.2 HSPSM Definition

Building on the recursive computation of partial symmetric polynomials, we now define a formal sequential model of computation.

Definition 20 (Homogeneous Symmetric Polynomial State Machine)**.** *A Homogeneous Symmetric Polynomial State Machine (HSPSM) is defined as a tuple $M = (n, K, \delta, F)$ where:*

- *n is the input length*
- *$K \subseteq \{0, 1, \ldots, n\}$ is a set of degrees*
- *$\delta : \{0,1\}^{|K|} \times \{0,1\} \to \{0,1\}^{|K|}$ is the transition function*
- *$F \subseteq \{0,1\}^{|K|}$ is the set of accepting states*

The state of the machine after reading the prefix $x_{1:j} = (x_1, \ldots, x_j)$ is:

$$S_j = (h_{k,j}(x))_{k \in K} \tag{34}$$

The computation proceeds by updating the state according to the transition function:

$$S_{j+1} = \delta(S_j, x_{j+1}) \tag{35}$$

The machine accepts the input x if $S_n \in F$.

Lemma 14 (HSPSM Update Rule)**.** *The natural transition function for an HSPSM is:*

$$\delta((v_k)_{k \in K}, x_{j+1}) = (v_k \oplus (x_{j+1} \cdot v_{k-1}))_{k \in K} \tag{36}$$

where $v_{-1} = 0$ by convention.

Proof. This follows directly from the recursive formulation for $h_{k,j}(x) = h_{k,j-1}(x) \oplus (x_j \cdot h_{k-1,j-1}(x))$. □

4.3 Computational Universality

Theorem 15 (HSPSM Universality)**.** *The Homogeneous Symmetric Polynomial State Machine is computationally universal, capable of simulating any Turing machine.*

Proof. We establish universality through a series of steps:

Step 1: State Encoding. We first show how to encode the state of a Turing machine in the HSPSM state vector. Given a Turing machine M with state set Q, tape alphabet Γ, and head position, we encode:

- The current state $q \in Q$ using a fixed number of homogeneous polynomial values
- The tape contents within a bounded window around the head position
- The relative head position within this window

Specifically, we use $h_{2^i,n}(x)$ values to encode binary information, exploiting the fact that $h_{2^i,n}(1^n) \equiv \binom{n}{2^i} \pmod 2$, which by Lucas' Theorem corresponds to the i-th bit in the binary representation of n.

Step 2: Transition Simulation. For each transition $\delta(q, a) = (q', a', d)$ in the Turing machine's transition function, we construct a composition of homogeneous symmetric polynomials that identifies the current state and scanned symbol, updates the state encoding to q', updates the tape contents by writing a', and shifts the tape encoding according to direction d.

These compositions can be implemented using the recurrence relation:

$$h_{k,j}(x) = h_{k,j-1}(x) \oplus (x_j \cdot h_{k-1,j-1}(x)) \tag{37}$$

Step 3: Control Logic. We implement the overall control logic using polynomial compositions that select the appropriate transition based on the current state. This logic can be constructed using masked symmetric polynomial operations to conditionally apply different updates based on the current state encoding.

Step 4: Simulation Efficiency. We analyze the resources required for this simulation. Each step of the Turing machine requires:

- $O(\log |Q|)$ homogeneous polynomial degrees to encode the state
- $O(\log |\Gamma|)$ polynomial compositions per tape cell
- A constant number of mask operations for control logic

The total size of the HSPSM simulation is polynomial in the size of the Turing machine description.

Step 5: Halting Detection. Finally, we show how to detect when the simulated Turing machine has reached a halting state, by constructing a simple polynomial composition that evaluates to 1 exactly when the state encoding corresponds to a halting state. □

This construction demonstrates that the HSPSM model, using only the homogeneous symmetric polynomial recurrence relation, can simulate any Turing machine, thereby establishing its computational universality.

Corollary 16 (Cook-Levin Theorem for HSPSM)**.** *For any language $L \in NP$, there exists a polynomial-time reduction from L to the problem of determining whether a given input causes a specific HSPSM to reach an accepting configuration.*

This universality result establishes the HSPSM as a viable model of computation, with the remarkable property that all computation is performed through the single recurrence relation $h_{k,j}(x) = h_{k,j-1}(x) \oplus (x_j \cdot h_{k-1,j-1}(x))$.

4.4 Binomial Distribution of Computational Resources

The binomial structure inherent in homogeneous symmetric polynomials creates a natural distribution of computational resources that mirrors the binomial enumeration process.

Theorem 17 (Computational Resource Distribution)**.** *In a homogeneous polynomial computation of degree up to d on an input of length n, the computational effort is distributed according to the binomial distribution:*

$$\textit{Effort}(k) \propto \binom{n}{k} \text{ for } k \in \{0, 1, \ldots, \min(d, n)\} \tag{38}$$

Proof. The evaluation of $h_k(x)$ requires considering all monomials of degree k, which corresponds to selecting multisets of size k from n elements. The number of such multisets grows according to the binomial coefficient pattern, resulting in the stated distribution of computational effort. □

5 Complexity Measures

5.1 Symmetric Polynomial Composition Complexity

Definition 21 (SPCC Measure)**.** *For a language $L \subseteq \{0,1\}^*$, the Symmetric Polynomial Composition Complexity of L, denoted* $\mathrm{SPCC}(L)$*, is a function $f : \mathbb{N} \to \mathbb{N}$ where $f(n)$ represents the minimum number of symmetric polynomial operations needed in a composition to decide membership in L for inputs of length n.*

Definition 22 (Masking Complexity)**.** *For a language $L \subseteq \{0,1\}^*$, the Masking Complexity of L, denoted* $\mathrm{MC}(L)$*, is a function $f : \mathbb{N} \to \mathbb{N}$ where $f(n)$ represents the minimum number of distinct masks needed in a symmetric polynomial composition to decide membership in L for inputs of length n.*

Definition 23 (Degree Complexity)**.** *For a language $L \subseteq \{0,1\}^*$, the Degree Complexity of L, denoted* $\mathrm{DC}(L)$*, is a function $f : \mathbb{N} \to \mathbb{N}$ where $f(n)$ represents the maximum degree of any symmetric polynomial needed in a minimal composition to decide membership in L for inputs of length n.*

5.2 Relationship to Standard Complexity Classes

Theorem 18 (P Characterization)**.** *A language L is in P if and only if there exists a polynomial p such that for all n:*

$$\mathrm{SPCC}(L)(n) \leq p(n) \text{ and } \mathrm{DC}(L)(n) = O(1) \tag{39}$$

Proof. ($\Rightarrow$) If $L \in \mathrm{P}$, then there exists a polynomial-time Turing machine deciding L. By our Turing machine simulation theorem, this machine can be simulated by a symmetric polynomial composition of polynomial size. The key insight is that for a fixed-time Turing machine, the degree of the required symmetric polynomials is constant (independent of input size).

($\Leftarrow$) If L can be decided by a polynomial-size symmetric polynomial composition using constant-degree polynomials, then membership can be decided in polynomial time by directly evaluating the composition. □

Theorem 19 (NP Characterization)**.** *A language L is in NP if and only if there exist polynomials p and q such that for all n:*

$$\mathrm{SPCC}(L)(n) \leq p(n) \text{ and } \mathrm{DC}(L)(n) \leq q(n) \tag{40}$$

and there exists a polynomial r and a language $L' \in \mathrm{P}$ such that:

$$x \in L \iff \exists y \in \{0,1\}^{r(|x|)} : (x, y) \in L' \tag{41}$$

Proof. This follows from the standard definition of NP and our characterization of P. □

Theorem 20 (Homogeneous Polynomial Verification)**.** *A language L is in NP if and only if there exists a polynomial p such that for all inputs x of length n:*

$$x \in L \iff \exists k \in \{0, 1, \ldots, n\} : h_{k,n}(m_x) = 1 \tag{42}$$

where m_x is a string derived from x in polynomial time, and $h_{k,n}$ can be evaluated in polynomial time for any fixed k.

Proof. ($\Rightarrow$) If $L \in$ NP, then there exists a polynomial-time verifier V such that $x \in L$ if and only if there exists a certificate y such that $V(x, y) = 1$. We can encode the verification process as a homogeneous symmetric polynomial evaluation, where k represents the certificate. Specifically, we construct m_x such that $h_{k,n}(m_x) = 1$ if and only if k corresponds to a valid certificate for x.

($\Leftarrow$) If a language can be verified by checking $h_{k,n}(m_x) = 1$ for some k, then the verification process is in polynomial time for any fixed k. This gives us a polynomial-time verifier with k as the certificate, placing the language in NP. □

Conjecture 1 (Degree Separation)**.** *If $P \neq NP$, then there exists a language $L \in$ NP such that $\mathrm{DC}(L)(n)$ is not $O(1)$.*

6 Binomial Coefficient Structure and Complexity

The connection between binomial coefficients and our computational model provides deep insights into the nature of computational complexity.

6.1 Binomial Coefficient Growth and Complexity Classes

Theorem 21 (Binomial Coefficient Complexity)**.** *For any language L decidable by a symmetric polynomial composition using homogeneous polynomials of degree k, where k is a function of the input length n:*

1. *If $k = O(1)$, then $L \in P$*
2. *If $k = O(\log n)$, then $L \in P$*
3. *If $k = \Theta(n)$ and specifically if $k = \lfloor n/2 \rfloor$, then deciding L may require time exponential in n*

Proof. The complexity of evaluating $h_k(x)$ is dominated by the number of terms, which is $\binom{n+k-1}{k}$ for n variables and degree k.

When $k = O(1)$, this binomial coefficient is polynomial in n of degree k, specifically $O(n^k)$.

When $k = O(\log n)$, we have $\binom{n+k-1}{k} = O(n^k) = O(n^{O(\log n)}) = O(n^{\log n}) = O(2^{\log^2 n})$, which is still sub-exponential.

When $k = \lfloor n/2 \rfloor$, the central binomial coefficient $\binom{n}{\lfloor n/2 \rfloor} = \Theta(2^n/\sqrt{n})$ is exponential in n, making the computation potentially exponential. □

Lemma 22 (Binomial Degree Complexity Relation)**.** *The binomial coefficient $\binom{n}{k}$ has the following properties:*

1. *For fixed k, $\binom{n}{k} = O(n^k)$*
2. *For $k = \lfloor n/2 \rfloor$, $\binom{n}{k} = \Theta(2^n/\sqrt{n})$*
3. *For any polynomial $p(n)$, there exists n_0 such that for all $n > n_0$, $\binom{n}{\lfloor n/2 \rfloor} > p(n)$*

Proof. The first property follows from the formula $\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n(n-1)\cdots(n-k+1)}{k!}$, which is a polynomial in n of degree k for fixed k.

The second property follows from Stirling's approximation of the factorial, which gives $\binom{n}{\lfloor n/2 \rfloor} \sim \frac{2^n}{\sqrt{\pi n/2}}$.

The third property follows from the asymptotic growth rate of the central binomial coefficient being exponential, while any polynomial function grows at a slower rate. □

6.2 Binomial Enumeration and NP Verification

Theorem 23 (Binomial Enumeration)**.** *Let $\tau(x, y)$ be the $(y+1)$th smallest number with $(x+1)$ ones in its binary representation. Then after 2^n-1 steps of an enumeration process indexed by τ, the counts of operations performed for functions with indices $0, 1, \ldots, n-1$ follow the binomial distribution $\binom{n}{0}, \binom{n}{1}, \ldots, \binom{n}{n-1}$.*

Proof. The function $\tau(x, y)$ creates a mapping that associates each pair (x, y) with a unique natural number. The enumeration process follows this mapping, and after $2^n - 1$ steps, the counts of operations performed for each index follow exactly the binomial coefficient distribution. □

Theorem 24 (NP Certificate as Degree Parameter)**.** *Let L be a language in NP. Then there exists a polynomial-time computable function f such that:*

$$x \in L \iff \exists k \in \{0, 1, \ldots, |x|\} : h_k(f(x)) = 1 \tag{43}$$

Proof. Since L is in NP, there exists a polynomial-time verifier V and a polynomial p such that:

$$x \in L \iff \exists y \in \{0, 1\}^{p(|x|)} : V(x, y) = 1 \tag{44}$$

We construct $f(x)$ to encode the verification process such that $h_k(f(x)) = 1$ if and only if k encodes a certificate y for which $V(x, y) = 1$. This construction leverages the binomial distribution of homogeneous polynomial evaluations, allowing the degree parameter k to effectively serve as the certificate. □

Corollary 25 (Certificate Complexity)**.** *The complexity of verifying a language $L \in$ NP using the homogeneous polynomial model is polynomial in the input size for any fixed certificate k.*

Proof. For a fixed degree parameter k, evaluating $h_k(f(x))$ requires time polynomial in $|x|$, specifically $O(|x|^k)$. Since the verification process involves evaluating this for a specific k provided as the certificate, the verification complexity is polynomial. □

7 Symmetry-Breaking and Complexity

7.1 Symmetry-Breaking Measure

Definition 24 (Symmetry-Breaking Measure)**.** *The symmetry-breaking measure of a computational problem P, denoted SBM(P), is the minimum number of symmetry-breaking operations required to solve P.*

Theorem 26 (Masking-Complexity Correspondence)**.** *For any computational model M with complexity measure C_M, there exists a corresponding symmetric polynomial model S with symmetry-breaking measure C_S such that for any function f:*

$$C_M(f) = \Theta(C_S(f)) \tag{45}$$

where C_S measures the minimum number of mask bits required to compute f using symmetric polynomials.

Proof Sketch. We establish this correspondence by:

1. Showing that any computation in model M can be simulated using symmetric polynomials with an appropriate number of masking operations
2. Demonstrating that the number of required mask bits is proportional to the original complexity measure C_M
3. Proving that any computation using symmetric polynomials with a given number of mask bits can be simulated in the original model with proportional complexity

□

7.2 Complexity Class Characterizations

Theorem 27 (P Characterization via Symmetry-Breaking)**.** *A language L is in P if and only if it can be decided using polynomially many masking operations where the masks are computable in polynomial time.*

Theorem 28 (NP Characterization via Symmetry-Breaking)**.** *A language L is in NP if and only if it can be verified using polynomially many masking operations where one of the masks serves as a certificate.*

Theorem 29 (coNP Characterization via Symmetry-Breaking)**.** *A language L is in coNP if and only if its complement can be verified using polynomially many masking operations where one of the masks serves as a certificate.*

References

[1] Arora, S., and Barak, B. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[2] Lucas, E. Théorème sur les fonctions numériques. *Comptes Rendus de l'Académie des Sciences, Paris*, 83:1286–1288, 1878.

[3] Macdonald, I.G. *Symmetric Functions and Hall Polynomials*. Oxford University Press, 1995.

[4] Razborov, A.A. Lower bounds on the size of bounded depth circuits over a complete basis with logical addition. *Mathematical Notes*, 41(4):333–338, 1987.

[5] Smolensky, R. Algebraic methods in the theory of lower bounds for Boolean circuit complexity. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pp. 77–82, 1987.

[6] Yao, A.C. Separating the polynomial-time hierarchy by oracles. In *26th Annual Symposium on Foundations of Computer Science*, pp. 1–10, 1985.