On the Number of Inequivalent Monotone Boolean Functions of 9 Variables

Bartłomiej Pawelski

Abstract—The problem of counting all inequivalent monotone Boolean functions of nine variables is considered. We solve the problem using known algorithms and deriving new ones when necessary. We describe methods to count fixed points in sets of all monotone Boolean functions under a given permutation of input variables. With these techniques as a basis, we tabulate the cardinalities of these sets for nine variables. By applying Burnside’s lemma and the numbers obtained, we calculate the number of inequivalent monotone Boolean functions of 9 variables, which equals $789,204,635,842,035,040,527,740,846,300,252,680$.

Index Terms—Boolean functions, monotone Boolean functions, Dedekind number, inequivalent monotone Boolean functions.

I. INTRODUCTION

Let $B$ denote the set of two bits $\{0, 1\}$ and let $B^n$ denote the set of $n$-element sequences of $B$. A Boolean function is any function $f : B^n \rightarrow B$. There are $2^n$ elements in $B^n$ and $2^{2^n}$ Boolean functions of $n$ variables.

We have a partial order in $B$: $0 \leq 0, 0 \leq 1$, and $1 \leq 1$. This partial order induces a partial order on $B^n$. There are $2^n$ elements in $B^n$, $x \leq y$ if and only if $x_i \leq y_i$ for all $i$.

A Boolean function is said to be monotone if for any $x, y \in B^n$, when $x \leq y$, it follows that $f(x) \leq f(y)$.

We define $D_n$ as the set of all monotone Boolean functions of $n$ variables. Let $d_n$ represent the cardinality of $D_n$, also known as the $n$-th Dedekind number. Dedekind numbers are listed in the On-Line Encyclopedia of Integer Sequences (OEIS) sequence A003182 (see Table I). The term $d_n$ also corresponds to the number of simple games with $n$ players in minimal winning form, the number of antichains of subsets of an $n$ set, the number of Sperner families and the cardinality of a free distributive lattice on $n$ generators [18].

An important and recent result is the calculation of $d_9$, which was independently achieved in 2023 by Jäkel [10] and Van Hirtum, De Causmaecker, Goemaere, Kenter, Riebler, Lass, and Plessl [9].

Monotone Boolean functions are instrumental in the design and study of nonlinear digital filters, like stack filters, which are widely used in image processing and other applications where noise sources are non–Gaussian or non–additive [17]. Other practical applications of monotone Boolean functions include, among others, machine learning algorithms which are not heuristic [13], optimization of consensus maximization algorithms for robust fitting in computer vision [20], game theory [8], and cryptography [6].

Two monotone Boolean functions are said to be equivalent if the first function can be obtained from the second function through any permutation of input variables. Let $R_n$ represent the set of all equivalence classes of $D_n$, and let $r_n$ denote the cardinality of this set, also known as the number of inequivalent monotone Boolean functions of $n$ variables. The values of $r_n$ are listed in the OEIS sequence A003182 (see Table I). Note that the value of $r_9$, first reported in this paper, has subsequently been included in the OEIS. There are at most $n!$ permutations of $n$ variables, so each equivalence class has at most $n!$ Boolean functions. Therefore, a lower bound of $r_n$ is given by $\frac{n!}{n}$. In 1985 and 1986, Liu and Hu [11], [12] calculated $r_n$ for values of $n$ up to 7. Independently, $r_7$ was calculated by Stephen and Yusun [19]. In 2021, the author calculated $r_8$ [14], and this result was independently reported in 2022 by Carić and Živković [5]. The recent paper by Szepietowski [16] contributes to the topic by systematizing knowledge about studying monotone Boolean functions and counting fixed points of permutations acting on $D_n$. Notably, one of the algorithms from Szepietowski’s paper has been applied in the current study (see Section III-C).

In this paper, our primary contribution is the computation of $r_9$. Our work focuses on counting the fixed points in $D_9$ under permutations of the cycle types of its input variables. The OEIS sequence A000041, which is well-known for enumerating the number of integer partitions, also provides information on the number of different cycle types as $n$ increases. Given this sequence, there are 29 cycle types for $n = 9$ (see Table V). Then, we apply a specialized version of Burnside’s lemma, tailored specifically for these cycle types, similarly to the approaches found in [5], [11], [12], and [14].

The computational complexity and diversity of these cycle types, we leverage existing algorithms where suitable and introduce new ones when demanded by specific computational challenges:

1) Algorithm III-A, corresponding to [14, Algorithm 1].
2) Algorithm III-B, a partially new algorithm, evolved from [14, Algorithm 2].
3) Algorithm III-C, corresponding to [16, Theorem 4].
4) Algorithms III-D and III-E: new algorithms, evolved from [14, Algorithm 3].
The methodology is further elaborated upon in Section III. Our efforts culminate in the calculation of
\[ r_9 = 789204635842035040527740846300252680, \]
marking the first-ever computation of this number.

II. Preliminaries

A. Integer Representation of a Boolean Function

In the context of the n-dimensional Boolean space, denoted as \( B^n \), we can consider an array composed of \( 2^n \) potential inputs for a given Boolean function. Each of these possible input combinations is associated with a specific output bit.

In this manner, we represent any Boolean function of \( n \) variables using a binary vector of length \( 2^n \), which is called a truth table representation of a Boolean function (also see [2], [3, Chapter II]). Note that a Boolean function of up to six inputs for a given Boolean function. Each of these possible inputs for a given Boolean function. Each of these possible

| \( n \) | \( d_n \) | \( r_n \) |
|---|---|---|
| 0 | 2 | 2 |
| 1 | 3 | 3 |
| 2 | 6 | 5 |
| 3 | 20 | 10 |
| 4 | 168 | 30 |
| 5 | 7581 | 210 |
| 6 | 7828354 | 16393 |
| 7 | 2414682040998 | 490013048 |
| 8 | 56130437228675759707788 | 139219548889993358 |
| 9 | 2863865776829841128469151667598498812366 | 789204635842035040527740846300252680 |

Consider a poset \( P = (X, \leq) \). A downset of \( P \) is defined as a subset \( S \subseteq X \) such that if \( x \in S \), then all elements in \( X \) satisfying \( y \leq x \) also belong to \( S \). The set \( D_n \) corresponds to the set of all downsets of \( B^n \). As a result, every element of \( D_n \) can be associated with a specific downset of \( B^n \).

An incidence matrix of a poset, also referred to as an array in Szeptycki’s paper [16, Section 3] is a binary matrix that represents the partial order relation between elements in the poset. For a poset \( P = (X, \leq) \), its incidence matrix \( M \) has rows and columns indexed by elements in \( X \). The entry \( M(x, y) = 1 \) if \( x \leq y \), and 0 otherwise.

Consider two posets \( (X, \leq) \) and \( (Y, \leq) \). The Cartesian product \( X \times Y \) is the poset with the relation \( \leq \) defined by \( (a, b) \leq (c, d) \) if and only if \( a \leq c \) and \( b \leq d \). For two disjoint posets \( (X, \leq) \) and \( (Y, \leq) \), by \( X + Y \) we denote the disjoint union (sum) with the order defined as follows: \( a \leq b \) iff \( (a, b \in X \text{ and } a \leq b) \) or \( (a, b \in Y \text{ and } a \leq b) \).

Let us recall the function \( f : X \rightarrow Y \) is monotone if for any elements \( x, y \in X \) such that \( x \leq y \), we have \( f(x) \leq f(y) \).

We denote the set of all monotone functions from \( X \) to \( Y \) by \( Y^X \). A partial order on \( Y^X \) can be defined as follows: given two functions \( f, g \in Y^X \), we say that \( f \leq g \) if and only if \( f(x) \leq g(x) \) for all \( x \in X \).

Two posets are said to be isomorphic if their order structures are analogous. Formally, posets \( P = (X, \leq) \) and \( P' = (X', \leq') \) are isomorphic, if there exists a bijection \( f : X \rightarrow X' \) such that for all elements \( x_1, x_2 \in P \), \( x_1 \leq x_2 \) if and only if \( f(x_1) \leq' f(x_2) \). We use the equals sign \( = \) to denote an isomorphic relationship between posets.

In the current study, we use the following lemmas:

Lemma 3: [16] For three posets \( R, S, T \):

1. If \( S \) and \( T \) are disjoint, then \( R^{S+T} = R^S \times R^T \).
2. \( R^{S \times T} = (R^S)^T \) and also \( R^{S \times T} = (R^T)^S \).

Lemma 4: [16]

(a) \( B^{k+m} = B^k \times B^m \).
(b) \( D_{k+m} = (D_k)B^m \).

These lemmas, in various formulations, are standard tools in the literature for counting monotone Boolean functions. For example, Wiedemann [21] used the isomorphism \( D_8 = (D_6)^2 \) to count the monotone Boolean functions of eight variables.
C. Permutations Acting on Variables of a Boolean Function

Let $S_n$ represent the set of all permutations of the set $\{1, 2, \ldots, n\}$. In this context, we treat $S_n$ as the set of all permutations of the $n$ variables within a Boolean function. The action of a permutation $\pi$ on a Boolean function $f$ essentially permutes the input variables of $f$. For a given Boolean function $f$ and a permutation $\pi$, we denote the resulting function as $\pi(f)$.

Each permutation $\pi \in S_n$ can be represented as a product of disjoint cycles. The cycle type of $\pi$ is defined as the tuple of lengths of its disjoint cycles arranged in increasing order. For example, the type of permutation $\pi = (12)(34)(5678)$ is $(2, 2, 4)$, and its total length is 8.

Given a set $X$ and a permutation $\pi$ acting on $X$, the orbit of an element $x \in X$ under the action of $\pi$ is the set of all images of $x$ obtained by applying $\pi$ repeatedly. Formally, the orbit of $x$ is:

$$\text{Orb}_\pi(x) = \{\pi^j(x) \mid j \in \mathbb{N}\}$$

where $\pi^j(x)$ denotes the result of applying the permutation $\pi$ $j$ times to $x$.

An element $x$ is said to be a fixed point of $\pi$ if it remains unchanged under the action of $\pi$. The set $\Phi_n(\pi)$ contains all fixed points of $\pi$ acting on $D_n$. In this context, let $e$ represent the identity permutation, where none of the variables is swapped. This is shown in the first row of Table II, where each element of $B^3$ remains unchanged.

**Example 5:** As shown in Table III, the Boolean function of three variables represented by the integer 85 (in binary form as 01010101) is transformed under permutation $\pi = (12)$. After the permutation, the binary representation becomes 00110110, which corresponds to the integer 51. We therefore have $\pi(85) = 51$. This transformation allows us to illustrate two significant insights:

1) 51 and 85 are equivalent Boolean functions, illustrating that the application of a permutation on variables can yield a different, but equivalent, Boolean function;
2) neither 51 nor 85 are fixed points in $D_3$ under the permutation $\pi = (12)$, illustrating that under certain permutations, original functions do not remain invariant.

A simple example of a fixed point in $D_3$ under $\pi = (12)$ is the function represented by the integer 255, corresponding to the binary word 11111111. Indeed, this function remains invariant under any permutation, which can be observed upon examining its binary representation.

### Table II

| $e$ | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| $(12)$ | 000 | 010 | 001 | 011 | 100 | 110 | 101 | 111 |

### Table III

| $e$ | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 |
|-----|---|---|---|---|---|---|---|---|
| $(12)$ | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |

By applying Burnside’s lemma, $r_n$ can be calculated as follows [5], [14], [16]:

$$r_n = \frac{1}{n!} \sum_{i=1}^{k} \mu_i \phi_i,$$

where:

- $\phi_i(\pi) = |\Phi_n(\pi)|$,
- $k$ is the number of different cycle types in $S_n$,
- $i$ is the index of a cycle type,
- $\mu_i$ is the number of permutations $\pi \in S_n$ with cycle type $i$,
- $\pi_i$ is a representative permutation $\pi \in S_n$ with cycle type $i$.

The first application of Burnside’s lemma for calculating $r_n$ found in the literature was by Liu and Hu [11], [12]. They used it in 1985 and 1986 to determine $r_n$ for values of $n$ up to 7. Computing $\phi(\pi)$ for all cycle types in $S_n$ (excluding the identity permutation) requires significantly less computational effort compared to calculating $d_\pi$, making it feasible with our available resources.

### III. Methodology

To efficiently calculate $\phi(\pi)$ for all cases, we employ several algorithms, the details of which are described in this section.

#### A. Algorithm III-A

Let $B^n(\pi)$ denote the poset of orbits of $B^n$ under $\pi \in S_n$, where two orbits $C_1$ and $C_2$ are in the relation $C_1 \preceq \pi C_2$ if and only if for every $c_1 \in C_1$, there exists a $c_2 \in C_2$ such that $c_1 \preceq c_2$. An illustrative representation of this can be seen in Figure 2. Note that this figure includes two doublet orbits: $\{001, 010\}$ and $\{101, 110\}$.

**Lemma 6:** $\Phi_n(\pi) = B^n(\pi)$

**Proof:** Recall that $\Phi_n(\pi)$ represents the set of fixed points of $\pi$ acting on $D_n$. Given an orbit in $B^n(\pi)$, a function in $\Phi_n(\pi)$ determines a specific bit value (either 0 or 1) for the entire orbit. This assignment of bit values to orbits can be represented as a function from $B^n(\pi)$ to $B$, where each orbit is mapped to a single bit value. The set of all possible ways to assign bit values to orbits in $B^n(\pi)$ thus forms the set $B^{B^n(\pi)}$. Since every function in $\Phi_n(\pi)$ corresponds uniquely to such an assignment, and vice versa, we establish a one-to-one correspondence between $\Phi_n(\pi)$ and $B^{B^n(\pi)}$. 

#### Table III

| $e$ | 001 | 010 | 100 | 011 | 101 | 000 | 110 | 111 |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| $(12)$ | 011 | 101 | 110 | 101 | 110 | 000 | 110 | 111 |

Fig. 1. $B^3$ and $B^3$ under $\pi = (12)$ on Hasse diagrams.
The essence of this algorithm is to count $\phi_n(\pi)$ by generating all downsets of $B_n(\pi)$ recursively and exploiting the fact proven in Lemma 6, that $\phi_n(\pi) = |B^B(\pi)|$. For a pseudocode of Algorithm III-A, please refer to [14, Algorithm 1].

A significant limitation of this approach, however, is that it requires storing all elements of the set rather than simply counting them. This constraint makes the application of this algorithm highly dependent on the available main memory. For reference, our calculations were performed on a machine with 128 GB of main memory, and we were able to compute up to $\phi((123)(456789)) = 218542866$. Hence, in practice, all sets of fixed points with cardinalities smaller than about two hundred million can be computed using the resources at our disposal. The number 218542866 represents the upper limit of our computational capabilities in the context of hardware limitations, and not limitations of the method or the algorithm itself.

**B. Algorithm III-B**

In order to count the fixed points in $D_n$ under permutations containing at least one disjoint 1–cycle, we can leverage the isomorphism delineated by the subsequent lemma:

**Lemma 7:** Let $k$ denote the number of 1–cycles in a permutation $\pi$. Then, $\Phi_{n+k}(\pi) = \Phi_n(\pi)^{B^k}$.

**Proof:** Start by recalling the definition $\Phi_{n+k}(\pi) = B^{B^{n+k}(\pi)}$. Using Lemma 4(a), we can express the term $B^{B^{n+k}(\pi)}$ as $B^{B^{n}(\pi) \times B^k}$, given that the posets $B^n(\pi)$ and $B^k$ are disjoint and the last $k$ variables are not affected by $\pi$.

Next, by applying Lemma 3(2), $B^n(\pi) \times B^k$ can be expressed as $(B^n(\pi)B^k = \Phi_n(\pi)B^k)^{\times k}$. $\square$

In [14, Section 3.2], we outline the simplest version of the algorithm that uses this isomorphism for the case where $k = 1$. Ćirić and Živković elaborate on this algorithm for the situation where $k = 2$ [5, Theorem 3.1]. Several studies in the literature provide detailed descriptions of algorithms for counting monotone Boolean functions. Notably, in the context of our work, these algorithms are applicable to cases involving the identity permutation. By Lemma 7, these algorithms can be adapted to our needs through modification of the input set. For example, Fidytek et al. [7] describe algorithms for $k \in \{1, 2, 4\}$, while a Campo [4, Section 4.1] presents algorithms for $k \in \{1, 2, 3, 4\}$.

In order to calculate $r_9$, we extend the application of Lemma 7 to handle cases up to $k = 4$. To illustrate our approach, for example, to calculate $\phi_9((12)(34))$, we first generate the set $\Phi_6((12)(34))$ using our implementation of Algorithm III-A, and then apply Algorithm III-B with $k = 3$.

In the calculation of $r_8$, there is only one case where the algorithms referenced above (detailed in [14]) did not work due to computational resource limitations: specifically, $\phi_8((12)(34)(56)(78))$. In the calculation of $r_9$, we have three cases: $\phi_9((12)(34)(56)(78))$, $\phi_9((12)(34)(56)(789))$, and $\phi_9((123)(456)(789))$.

**C. Calculation of $\phi_9((12)(34)(56)(789))$**

Let $k \in \{2, 3\}$. Let $\tau$ represent a permutation where all cycle lengths are coprime to $k$. Define $\epsilon$ as the permutation obtained from the composition of $\tau$ and a single-cycle permutation of length $k$, ensuring that these permutations are disjoint. Let $P_\epsilon$ denote a special type of poset, known as a chain, where each pair of $n$ elements is comparable.

**Lemma 8:** [1, 16] For any $n, m \in \mathbb{N}$, the number of functions from $(B^n)^{P_m}$ is equal to the sum of elements of the $(m − 1)$–power of the incidence matrix of $B^n$.

**Corollary 9:** For $k \in \{2, 3\}$, as poset $B^2((12))$ is isomorphic to the chain $P_3$ and poset $B^3((123))$ is isomorphic to the chain $P_3$, we have $B^{n+k}(\epsilon) = B^n(\tau)^{P_3+1}$.

**Lemma 8** indicates that the number of functions from a given poset to another poset, being a chain, can be calculated by raising its incidence matrix to the appropriate power and summing all its elements. This provides a computationally efficient technique for dealing with some complex poset structures. In Corollary 9, we use this lemma leaning on two isomorphisms: $B^2((12)) = P_3$ and $B^3((123)) = P_3$. This sets the stage for the subsequent corollary, first applied by Szeptykowski [16].

**Corollary 10:** For $k \in \{2, 3\}$, $\phi_{n+k}(\epsilon)$ is equal to the sum of elements of the $(k − 1)$–power of the incidence matrix of $(B^n(\tau))$.

We use this property to compute

$$\phi_9((12)(34)(56)(789)) = 807900672006$$

by summing all the elements of the third power of the incidence matrix of $\Phi_6((12)(34)(56))$, which has $8600 \times 8600$ elements. Notably, using Algorithm III-A to compute this number would have been impossible due to the immense memory required to store all elements.

**D. Calculation of $\phi_9((12)(34)(56)(789))$**

We adopt the notation from the previous subsection with minor modifications. Let $\tau$ represent a permutation where all cycle lengths are either 1 or 2, and the total length equals $n$. Now, denote $\epsilon$ as the permutation obtained from the composition of $\tau$ and the single cycle permutation of length 2, ensuring that these permutations are disjoint.

**Lemma 11:** For $k = 2$ and $n \geq k$, there exists a decomposition of the poset $B^{n+2}$ into four posets: $R_{00}, R_{01}, R_{10}, R_{11}$, which forms a poset isomorphic to $B^2$.

The permutation $\epsilon$ acting on $B^{n+2}$ affects these four parts in the following manner:
1) \( R_{00}(\epsilon) \) is isomorphic to \( B^n(\tau) \).
2) \( R_{01}(\epsilon) \) and \( R_{10}(\epsilon) \) form \( 2^n \) cycles of length 2: in each cycle, we have the \( i \)-th element of \( R_{01} \) and the \( i \)-th element of \( R_{10}(\tau) \).
3) \( R_{11}(\epsilon) \) is isomorphic to \( B^n(\tau) \).

This method is also described in [14, Algorithm 3] (see also [5, Section IV], [16, Section 6.5]) as a response to the need to calculate \( \phi_5(\pi) \) under \( \pi \) with all cycle lengths of 2. Upon specifying \( \epsilon \), Lemma 11 can be proved by listing \( B^{n+2}(\epsilon) \) and decomposing it into four equal parts. Additionally, a computer-assisted proof for the necessary poset \( B^0(((12)(34)(56)(78)) \) is available at [15] in the file \( P0.java \).

**Example 12:** Decomposition of \( B^4(((12)(34)) \) according to Lemma 11:

1) \( R_{00} \) as \(((0)(1 2)(3)) \), which is isomorphic to \( B^2((12)) \);
2) \( R_{01}, R_{10} \) as four 2–element cycles: \((4 8)(5 10)(6 9)(7 11) \);
3) \( R_{11} \) as \(((12)(13 14)(15)) \), which is isomorphic to \( B^3((12)) \).

**Lemma 13:** Each function \( x \in \Phi_{9}(((12)(34)(56)(78)) \) can be combined from these four functions:

1) \( a \in \Phi_{7}((12)(34)(56)) \);
2) \( b \in D_{7}, b \geq a \);
3) \( c = \tau(b), c \geq a \);
4) \( d \in \Phi_{7}((12)(34)(56)), d \geq b \cup c \).

**Proof:** Considering the decomposition of the poset \( B^{n+2} \) given by Lemma 11, we can infer that \( R_{00} \) and \( R_{11} \) are affected by permutation \( \tau = (12)(34)(56) \), so they are isomorphic to the poset \( B^2((12)(34)(56)) \). The set of functions \( B^{2(12)(34)(56)} \) is equal to \( \Phi_{7}((12)(34)(56)) \), thus we have \( a, b \in \Phi_{7}((12)(34)(56)) \). Moving on to the subposets \( R_{01} \) and \( R_{10} \), we find that while the functions \( b \) and \( c \) belong to \( D_{7} \), they pair in a bounded manner, specifically \( c = \tau(b) \). Moreover, as these four elements are organized on the poset \( B^2 \), we must adhere to the constraints: \( a \leq b \leq d \) and \( a \leq c \leq d \).

**Algorithm III-D Calculation of \( \Phi_{9}((12)(34)(56)(78)) \)**

**Input:** \( D_{7} \) and \( \Phi_{7}((12)(34)(56)) \)

**Output:** \( s = \Phi_{9}((12)(34)(56)(78)) \)

1: Initialize \( s = 0 \)
2: for all \( b \in D_{7} \) do
3: Initialize \( c = \tau(b) \)
4: Calculate \( \text{down} = \{a \in \Phi_{7}((12)(34)(56)) : a \leq (b \cap c)\} \)
5: Calculate \( \text{up} = \{d \in \Phi_{7}((12)(34)(56)) : d \geq (b \cup c)\} \)
6: \( s = s + \text{down} \cdot \text{up} \)
7: end for

\( \Phi_{7}((12)(34)(56)) \) contains 12015832 elements, so we can store it in main memory. The most challenging part is storing \( D_{7} \) (which would require about 35 TB, see Table IV).

We overcome this by storing \( R_{7} \), which contains 490013148 elements that fit in main memory. Then, for each \( x \in R_{7} \), we execute an “unpacking” operation, exploiting

| \( n \) | \( d_n \) | Size of function | Size of \( D_n \) |
|-----|-----|-----------------|----------------|
| 0   | 2   | \( 1b \)        | \( 2 \) \text{B} |
| 1   | 3   | \( 2b \)        | \( 3 \) \text{B} |
| 2   | 6   | \( 4b \)        | \( 6 \) \text{B} |
| 3   | 20  | \( 8b \)        | \( 20 \) \text{B} |
| 4   | 168 | \( 16b \)       | \( 336 \) \text{B} |
| 5   | 7581| \( 32b \)       | \( 29.61 \) \text{KB} |
| 6   | 7828354 | \( 64b \) | \( 59.725 \) \text{MB} |
| 7   | 2414682040998 | \( 128b \) | \( 35.138 \) \text{TB} |

\( 7! = 5040 \) possible permutations of input variables. Finally, we obtained \( \phi_9((12)(34)(56)(78)) = 17143334331688770356814 \).

**E. Calculation of \( \Phi_{9}((123)(456)(789)) \)**

Once again, let us revisit the notation introduced in the previous subsection with a slight modification. This time, let \( \tau \) represent a permutation where all cycle lengths are either 1 or 3, and the total length equals \( n \). Now, denote \( \epsilon \) as the permutation obtained from the composition of \( \tau \) and the single cycle permutation of length 3, ensuring that these permutations are disjoint. With these conventions in place, we are able to state and prove a lemma that resembles the previous one.

**Lemma 14:** For \( k = 3 \) and \( n \geq k \), there exists a decomposition of the poset \( B^{n+3} \) into eight posets: \( R_{000}, R_{001}, R_{010}, \ldots, R_{111}, \) which forms a poset isomorphic to \( B^3 \).

The permutation \( \epsilon \) acting on \( B^{n+3} \) acts on those eight parts in the following way:

1) \( R_{000}(\epsilon) \) is isomorphic to \( B^n(\tau) \).
2) \( R_{001}(\epsilon) \) and \( R_{100}(\epsilon) \) form \( 2^n \) cycles of length 3: in each cycle, we have the \( i \)-th element of \( R_{001} \), the \( i \)-th element of \( R_{100}(\tau^2) \).
3) \( R_{011}(\epsilon) \) and \( R_{110}(\epsilon) \) form \( 2^n \) cycles of length 3: in each cycle, we have the \( i \)-th element of \( R_{011} \), the \( i \)-th element of \( R_{110}(\tau^2) \).
4) \( R_{111}(\epsilon) \) is isomorphic to \( B^n(\tau) \).

If \( \epsilon \) is specified, Lemma 14 can be proved by listing \( B^{n+3}(\epsilon) \) and decomposing it into eight equal parts. Additionally, a computer-assisted proof for the necessary poset \( B^6((123)(456)(789)) \) is available at [15] in the file \( P1.java \).

**Example 15:** Decomposition of \( B^6((123)(456)) \) according to Lemma 14:

1) \( R_{000} \) as \(((0)(1 2 4)(3 6 5)(7)) \), which is isomorphic to \( B^3((123)) \);
2) \( R_{001}, R_{010}, R_{100} \) as eight 3–element cycles: \((8 16 32)(9 18 36)(10 20 33)(11 22 37)(12 17 34)(13 19 38)(14 21 35)(15 23 39) \);
3) \( R_{011}, R_{101}, R_{110} \) as eight 3–element cycles: \((24 48 40)(25 50 44)(26 52 41)(27 54 45)(28 49 42)(29 51 46)(30 53 43)(31 55 47) \);
4) \( R_{111} \) as \(((56)(57 58 60)(59 62 61)(63)) \), which is isomorphic to \( B^3((123)) \).
Lemma 16: Each function $x \in \Phi_9((123)(456)(789))$ can be combined from these eight functions:

1. $a \in \Phi_6((123)(456))$;
2. $b \in D_6, b \geq a$;
3. $c = \tau(b), c \geq a$;
4. $d = \tau(c), d \geq a$;
5. $e \in D_6, e \geq b \cup c$;
6. $f = \tau(e), f \geq b \cup d$;
7. $g = \tau(f), g \geq c \cup d$;
8. $h \in \Phi_9((123)(456)), h \geq e \cup f \cup g$.

Proof: Considering the decomposition of the poset $B^{n+3}$ given by Lemma 14, we can infer that $R_{000}$ and $R_{111}$ are affected by permutation $\tau = (123)(346)$, so they are isomorphic to the poset $B^6((123)(456))$. The set of functions $B^6((123)(456))$ is equal to $\Phi_6((123)(456))$.

Moving on to the subposets $R_{001}, R_{010}$ and $R_{110}$, we find that while the functions $b, c$ and $d$ belong to $D_6$, they are structured as bounded triplets, specifically $c = \tau(b)$ and $d = \tau(c)$. A similar structure is evident with the subposets $R_{011}, R_{101}$ and $R_{110}$.

Moreover, as these eight elements are organized on the poset $B^3$, we must adhere to the constraints: $b \geq a, c \geq a, \ldots, h \geq e \cup f \cup g$. □

Algorithm III-E Calculation of $\phi_9((123)(456)(789))$

Input: $D_6$ and $\Phi_6((123)(456))$
Output: $s = \phi_9((123)(456)(789))$

1: Initialize $s = 0$
2: for all $b \in D_6$ do
3: Initialize $c = \tau(b)$
4: Initialize $d = \tau(c)$
5: Calculate $\downarrow = |\{a \in \Phi_7((123)(456)) : a \leq (b \cap c \cap d)\}|$
6: for all $e \in D_6, e \geq b \cup c$ do
7: Initialize $f = \tau(e)$
8: Initialize $g = \tau(f)$
9: Calculate $\downarrow = |\{h \in \Phi_7((123)(456)) : h \geq (e \cup f \cup g)\}|$
10: $s = s + \downarrow \cdot \uparrow$
11: end for
12: end for

The essence of the algorithm allowing to calculate $\phi_9((123)(456)(789))$ is fully captured in Lemma 16, which is similar in spirit to Lemma 13. $D_6$ has 7828354 elements and $\Phi_6((123)(456))$ has 562 elements, so they fit in the main memory. We completed a calculation in about six hours with a result

$$\phi_9((123)(456)(789)) = 221557843276152.$$

F. Computational Results

The algorithms were implemented in Java. Additionally, for some cases, we have also written the implementation in Rust (for example, for Algorithm III-D) due to its native support of 128-bit integers. We utilized a machine with 32 Xeon threads, and each case was calculated separately.

After executing all the implemented algorithms, we estimate the total computation time to be about 10 days. The simplest cases are computed almost instantly, while the most challenging ones (for example, $\phi_9((12)(34))$) take up to 2 days. The complexity is primarily due to the size and structure of the orbit poset associated with each permutation.

Note that $\phi_9(e)$, which represents the value for the identity permutation, is equal to $d_9$. This specific case was not computed by us and is therefore not included in the table.

IV. Calculation of $r_9$

In April 2023, two research teams independently reported the following value of $d_9$ [9], [10]:

$$d_9 = 28638657766829841112846915167598498812366.$$

We can now finally make direct use of Equation 1, obtaining the following value:

$$r_9 = 789204635842035540527740846300252680.$$

We have a high degree of confidence in the accuracy of our results, supported by the fact that the entire sum:

$$\sum_{i=1}^{k} \mu_i \phi_9(\pi_i)$$
we obtained is divisible by $9! = 362880$. The source code of the project is available in the GitHub repository [15].

V. CONCLUSION

The algorithms for generating and counting the fixed points in the set of monotone Boolean functions under given permutation of input variables have been described. Furthermore, the methods described were used as a basis for tabulating the numbers of fixed points under permutations of all cycle types in $D_9$. By applying Burnside’s lemma and the cardinalities obtained, we computed the number of inequivalent monotone Boolean functions of 9 variables, which equals $r_9 = 789204635842035040527740846300252680$.

REFERENCES

[1] M. Aigner, *Combinatorial Theory*. New York, NY, USA: Springer-Verlag, 1979.
[2] V. Bakoev, “Fast bitwise implementation of the algebraic normal form transform,” *Serdica J. Comput.*, vol. 11, no. 1, pp. 45–57, Nov. 2017.
[3] C. R. Baugh, “Generation of representative functions of the NPN equivalence classes of unate Boolean functions,” *IEEE Trans. Comput.*, vol. C-21, no. 12, pp. 1373–1379, Dec. 1972.
[4] F. A. Campo, “Relations between powers of Dedekind numbers and exponential sums related to them,” *J. Integer Sequences*, vol. 21, pp. 1–25, Jan. 2018. [Online]. Available: https://cs.uwaterloo.ca/journals/JIS/VOL21/Campo/campo3.html
[5] M. Carić and M. Živkovic, “The number of nonequivalent monotone Boolean functions of 8 variables,” *IEEE Trans. Inf. Theory*, vol. 69, no. 6, pp. 4027–4034, 2023.
[6] C. Carlet, D. Joyner, P. Stănică, and D. Tang, “Cryptographic properties of monotone Boolean functions,” *J. Math. Cryptol.*, vol. 10, no. 1, pp. 1–14, Mar. 2016.
[7] R. Fidytek, A. W. Mostowski, R. Somla, and A. Szepietowski, “Algorithms counting monotone Boolean functions,” *Inf. Process. Lett.*, vol. 79, no. 5, pp. 203–209, Sep. 2001.
[8] J. Freixas, “On the enumeration of some inequivalent monotone Boolean functions,” *Optimization*, vol. 73, no. 4, pp. 1253–1266, Apr. 2024.
[9] L. Van Hirtum et al., “A computation of $D(9)$ using FPGA supercomputing,” 2023, arXiv:2304.03039.
[10] C. Jäkel, “A computation of the ninth Dedekind number,” *J. Comput. Algebra*, vols. 6–7, Sep. 2023, Art. no. 100006.
[11] C. C. Liu and S. B. Hu, “A mechanical algorithm of equivalent classification for free distributive lattices,” *Chin. J. Comput.*, vol. 3, no. 2, pp. 128–135, 1985.
[12] C. C. Liu and S. B. Hu, “A note on the problem of computing the number of equivalence classes of free distributive lattices,” *J. Wuhan Univ. Nat. Sci. Ed.*, vol. 1, no. 1, pp. 13–17, 1986.
[13] B. Kovalerchuk, E. Triantaphyllou, A. S. Deshpande, and E. Vityaev, “Interactive learning of monotone Boolean functions,” *Inf. Sci.*, vol. 94, nos. 1–4, pp. 87–118, Oct. 1996.
[14] B. Pawelski, “On the number of inequivalent monotone Boolean functions of 8 variables,” *J. Integer Sequences*, vol. 25, no. 7, pp. 1–11, 2022. [Online]. Available: https://cs.uwaterloo.ca/journals/JIS/VOL25/Pawelski/pawelski7.html
[15] B. Pawelski. (2023). *Algorithms Counting Fixed Points in Sets of Monotone Boolean Functions*. GitHub Repository. [Online]. Available: https://github.com/bpawelski/r9
[16] A. Szepietowski, “Fixes of permutations acting on monotone Boolean functions,” *J. Integer Sequences*, vol. 25, no. 9, pp. 1–15, 2022. [Online]. Available: https://cs.uwaterloo.ca/journals/JIS/VOL25/Szepietowski/szep7.html
[17] I. Shmulevich, O. Yli-Harja, J. Astola, and A. Korshunov, “On the robustness of the class of stack filters,” *IEEE Trans. Signal Process.*, vol. 50, no. 7, pp. 1640–1649, Jul. 2002.
[18] OEIS Foundation Inc. (2023). *Dedekind Numbers*, Entry A000372 in *The on-Line Encyclopedia of Integer Sequences*. [Online]. Available: https://oeis.org/A000372
[19] T. Stephen and T. Yusun, “Counting inequivalent monotone Boolean functions,” *Discrete Appl. Math.*, vol. 167, pp. 15–24, Apr. 2014.
[20] R. Tennakoon, D. Suter, E. Zhang, T.-J. Chin, and A. Bab-Hadiashar, “Consensus maximisation using influences of monotone Boolean functions,” in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 2865–2874.
[21] D. Wiedemann, “A computation of the eighth Dedekind number,” *Order*, vol. 8, no. 1, pp. 5–6, 1991.

Bartłomiej Pawelski is currently with the University of Gdansk, Poland. His current research interests include Boolean functions, especially the integer sequences related to them.