Normal Forms and Decompositions of Monotone Ternary Functions

Klaus Schneider, IEEE Senior Member
Department of Computer Science
RPTU University Kaiserslautern-Landau
Kaiserslautern, Germany
https://es.cs.rptu.de
ORCID: https://orcid.org/0000-0002-1305-7132

Nadine Kercher
Department of Computer Science
RPTU University Kaiserslautern-Landau
Kaiserslautern, Germany
https://es.cs.rptu.de
ORCID: https://orcid.org/0009-0007-0300-0689

2025 IEEE 55th International Symposium on Multiple-Valued Logic (ISMVL) | 979-8-3315-0744-2/25/$31.00 ©2025 IEEE | DOI: 10.1109/ISMVL64713.2025.00039

***Abstract*—Among the ternary functions, the subset of monotone functions undoubtedly has the most applications. For this reason, monotone ternary functions have been intensively studied in previous research, although sometimes under different names. For example, Mukaidono called them regular ternary functions and proved that these are exactly the ternary functions that can be represented by propositional logic formulas.**

In this paper, we present a decomposition theorem for monotone ternary functions to split a given function into four functions by eliminating one of the argument variables. The decomposition theorem presented in this paper is the basis for recursive algorithms on monotone ternary functions that eliminate one of the used variables in each recursion step. As an example, we present such a recursive algorithm that translates a given monotone function into a propositional logic formula, and we easily prove this way Mukaidono's theorems on disjunctive normal forms of regular ternary, i.e., monotone, and B-ternary functions.

***Index Terms*—ternary logic, three-valued logic, monotone functions, function decomposition**

I. Introduction

Three-valued logic has been introduced by Jan Łukasiewicz and Emil Post [31] more than a century ago to refine logical reasoning by adding a third truth value for a possible but unprovable truth. Kleene used ternary logic to reason about computable functions where the third truth value denotes a value that cannot be computed because it is undefined, unknown or not yet available. Kleene's negation, conjunction, and disjunction were monotone so that fixpoints can be computed to define the semantics of programs.

Meanwhile, ternary logic and ternary functions have proven to be useful in many areas such as logic programming [13]–[16], hazard analysis in circuits [5], [8], [12], [17], [18], [25], [50], and the semantics of programs [2], [3], [23], [28], [33], [34], [36], [37], [45], [47], just to name a few.

Since monotone functions are closed under composition, it is clear that all propositional logic formulas using Kleene's operators define monotone functions. It is more difficult to determine an equivalent propositional logic formula for a given ternary monotone function. Mukaidono proved that propositional logic with the constants 0, 1, and the Kleene operators $\neg$, $\wedge$, $\vee$ defines exactly a subset of the monotone functions called the B-ternary functions (see Theorem 9 and [24], [25]). These are the monotone functions that map boolean argument vectors to boolean values. Later, Mukaidono also proved that adding the constant $\bot$ allows propositional logic to represent even all monotone functions [27], [49].

In this paper, we revisit the monotone ternary functions and present a decomposition theorem to split a given monotone ternary function into four functions by eliminating one of the argument variables. Conversely, we can construct all monotone ternary functions by combining four monotone ternary functions with a new variable according to the decomposition theorem. The decomposition theorem presented in this paper is therefore the basis for recursive algorithms on monotone ternary functions that eliminate one of the variables used in each recursion step. As an example, we present such a recursive algorithm that translates a given monotone function into a propositional logic formula, and we easily prove this way Mukaidono's theorems on disjunctive normal forms of regular ternary, i.e., monotone, and B-ternary functions.

The paper has the following outline: Section II lists the notation used, and Section III discusses a straightforward decomposition of general ternary functions into three functions by fixing a variable to one of the constants $\bot$, 0, or 1. Since this theorem can be used for general ternary functions, it requires non-monotone operators such as the absence test (Definition 1) and is therefore not useful for monotone functions. Section IV presents the main contribution of the paper: Theorems 3-7 show the purpose of the base functions used to construct a monotone ternary function according to the final decomposition theorem. Conversely, Theorem 8 shows that all monotone functions can be recursively constructed by the decomposition theorem of this paper, so that we can use it to define recursive algorithms over the monotone ternary functions. In Section V, we use the results to reprove Mukaidono's theorems about the syntactic representation of monotone ternary and B-ternary functions by propositional logic. We also consider the important set of sequential functions as a further subset of the monotone functions with another decomposition theorem.

2378-2226/25/$31.00 ©2025 IEEE
DOI 10.1109/ISMVL64713.2025.00039

II. Preliminaries

In this paper, we consider ternary functions over the truth values $\mathbb{T} = \{\bot, 0, 1\}$ where $\mathbb{B} = \{0, 1\}$ is the subset of boolean values, and $\bot$ is the third truth value that denotes an unknown, unstable, unavailable, or undefined value depending on the application. We use the partial order $\preceq$ on $\mathbb{T}$ that is defined as $x \preceq y :\Leftrightarrow x = \bot \vee x = y$ so that all pairs of values have an infimum, but a supremum does not always exist.

For the syntactic representation of ternary functions, we use propositional logic with the following Kleene operators $\neg$, $\wedge$, $\vee$ that are monotone ternary extensions of the binary negation, conjunction, and disjunction[1]:

x	$\neg x$
$\bot$	$\bot$
0	1
1	0

$\wedge$	$\bot$	0	1
$\bot$	$\bot$	0	$\bot$
0	0	0	0
1	$\bot$	0	1

$\vee$	$\bot$	0	1
$\bot$	$\bot$	$\bot$	1
0	$\bot$	0	1
1	1	1	1

In addition to the above logic connectives, we also consider the exclusive disjunction written as $\oplus$, the sequential if-then-else (ITE) $(\gamma \Rightarrow \varphi_1 \mid \varphi_0)$, and the parallel if-then-else (ITE) $(\gamma \Rrightarrow \varphi_1 \mid \varphi_0)$ which are defined as follows:

- $\varphi \oplus \psi :\Leftrightarrow \neg\varphi \wedge \psi \vee \varphi \wedge \neg\psi$
- $(\gamma \Rightarrow \varphi_1 \mid \varphi_0) :\Leftrightarrow \neg\gamma \wedge \varphi_0 \vee \gamma \wedge \varphi_1 \vee \neg\gamma \wedge \gamma$
- $(\gamma \Rrightarrow \varphi_1 \mid \varphi_0) :\Leftrightarrow \neg\gamma \wedge \varphi_0 \vee \gamma \wedge \varphi_1 \vee \varphi_0 \wedge \varphi_1$

We clearly have $(0 \Rightarrow \varphi_1 \mid \varphi_0) = (0 \Rrightarrow \varphi_1 \mid \varphi_0) = \varphi_0$ and $(1 \Rightarrow \varphi_1 \mid \varphi_0) = (1 \Rrightarrow \varphi_1 \mid \varphi_0) = \varphi_1$, but for $\gamma = \bot$, sequential ITE and parallel ITE are different: $(\bot \Rightarrow \varphi_1 \mid \varphi_0) = \bot$ and $(\bot \Rrightarrow \varphi_1 \mid \varphi_0) = \bot \wedge (\varphi_0 \vee \varphi_1) \vee \varphi_0 \wedge \varphi_1$. The latter is the infimum of φ_0 and φ_1 which is 0 if both are 0, 1 if both are 1, and $\bot$ otherwise. Parallel ITE can express the Kleene operators, but since sequential functions [2], [28], [33], [45]–[47] are closed under composition, sequential ITE cannot express any parallel operators.

III. General Function Decompositions

In general, a function decomposition is an algorithm that determines for a given function $f : \mathcal{D}^{n+1} \to \mathcal{D}$ and one of its arguments $x \in \mathcal{D}$, some functions $f_1, \ldots, f_m : \mathcal{D}^n \to \mathcal{D}$ such that there is a function $g : \mathcal{D}^{m+1} \to \mathcal{D}$ with $f(x, y) = g(x, f_1(y), \ldots, f_m y)$. It is well known that the following are the only possible decompositions [1] of binary functions $f : \mathbb{B}^{n+1} \to \mathbb{B}$ besides permutations and negations:

- *Shannon:* $f(x, y) = f_0(y) \wedge \neg x \vee f_1(y) \wedge x$
- *Positive Davio:* $f(x, y) = f_0(y) \oplus x \wedge (f_0(y) \oplus f_1(y))$
- *Negative Davio:* $f(x, y) = f_1(y) \oplus \neg x \wedge (f_0(y) \oplus f_1(y))$

For all of the above decompositions, the functions f_0 and f_1 can be determined as co-factors of the given function f, i.e., $f_0(y) := f(0, y)$ and $f_1(y) := f(1, y)$.

[1]These embeddings of the two-valued operations in ternary functions are not the only possible ones even if we insist on montone functions: For the conjunction, there are three further solutions, namely the left-sequential conjunction $x \wedge (y \vee \neg x)$, the right-sequential conjunction $y \wedge (x \vee \neg y)$, and the strict conjunction $x \wedge y \vee \neg x \wedge x \vee \neg y \wedge y$. Analogously, there are the corresponding three other versions of the two-valued disjunction. However, since sequential functions are closed under composition, it is not possible to express the parallel operators shown in the table with the sequential ones which is a strong argument in favor of the parallel operators.

The Shannon decomposition is the basis for binary decision diagrams (BDDs) [6], and the Davio decompositions are the basis for functional decision diagrams (FDDs) [19], and both are used in combination in Kronecker decision diagrams [1], [11]. The Shannon decomposition is based on a case distinction and a combination of the cases by disjunction. For ternary functions, a generalization of this principle is possible, but requires additional operators:

Definition 1 (Presence/Absence Test)**.** *The ternary functions $\eta_\bot$, η_0, and η_1 are defined as follows:*

x	$\eta_\bot(x)$	$\eta_0(x)$	$\eta_1(x)$
$\bot$	1	0	0
0	0	1	0
1	0	0	1

In particular, $\eta_\bot$ is called the absence test, and its negation is called the presence test.

The names 'presence/absence' test are derived from applications in model-based design where $\bot$ denotes the absence of a data value. None of these functions is monotone, so none of them can be expressed by the composition of monotone operators, and in particular, we cannot define such operations with propositional logic. However, we only need one of them:

Lemma 1 (Presence/Absence Test)**.** *Each one of $\eta_\bot$, η_0, and η_1 can express the other two:*

- $\eta_\bot(x) \Leftrightarrow \neg\eta_1(x \vee \neg x)$
- $\eta_\bot(x) \Leftrightarrow \neg\eta_0(x \wedge \neg x)$
- $\eta_0(x) \Leftrightarrow \neg x \wedge \neg\eta_\bot(x)$
- $\eta_1(x) \Leftrightarrow \quad x \wedge \neg\eta_\bot(x)$

For boolean values, we clearly have $\eta_0(x) = \neg x$ and $\eta_1(x) = x$ so that we can write the Shannon decomposition as $f(x, y) = f_0(y) \wedge \eta_0(x) \vee f_1(y) \wedge \eta_1(x)$ which immediately leads to the following ternary function decomposition:

Theorem 1 (Ternary Function Decomposition)**.** *For any function $f : \mathbb{T}^{n+1} \to \mathbb{T}$, there are functions $f_0, f_1, f_\bot : \mathbb{T}^{n+1} \to \mathbb{T}$ such that the following holds:*

$$\begin{aligned} f(x, y) &\Leftrightarrow f_0(y) \wedge \eta_0(x) \vee f_1(y) \wedge \eta_1(x) \vee f_\bot(y) \wedge \eta_\bot(x) \\ &\Leftrightarrow (f_0(y) \wedge \neg x \vee f_1(y) \wedge x) \wedge \neg\eta_\bot(x) \vee f_\bot(y) \wedge \eta_\bot(x) \\ &\Leftrightarrow (\eta_\bot(x) \Rightarrow f(\bot, y) \mid (x \Rightarrow f_1(y) \mid f_0(y))) \end{aligned}$$

In particular, we may use the co-factors $f_\bot(y) := f(\bot, y)$, $f_0(y) := f(0, y)$, and $f_1(y) := f(1, y)$.

The above ternary function decomposition is a straightforward generalization of the two-valued Shannon decomposition, and it can be directly used to define ternary decision diagrams. We can also use it to convert any given ternary function table into a disjunctive normal form:

Theorem 2 (Ternary DNF)**.** *Procedure* TTab2DNF *as implemented by Algorithm 1 converts any given ternary function table into an equivalent formula in disjunctive normal form. Therefore, any ternary function can be represented by a propositional logic formula using the constants $\bot$, 0, 1, the monotone ternary functions $\neg$, $\wedge$, $\vee$, and the non-monotone ternary function $\eta_\bot$.*

Algorithm 1 Converting Ternary Function Tables to DNFs

```
procedure TTab2DNF(n, f)
    if n = 0 then
        if f = ⊥ then return {{⊥}}
        else if f = 0 then return {}
        else return {{}}
        end if
    else
        D_⊥ ← TTab2DNF(n − 1, f(⊥, x_2, ..., x_n))
        D_0 ← TTab2DNF(n − 1, f(0, x_2, ..., x_n))
        D_1 ← TTab2DNF(n − 1, f(1, x_2, ..., x_n))
        f_⊥ ← {c ∪ {η_⊥(x_1)} | c ∈ D_⊥}
        f_0 ← {c ∪ {¬x_1, ¬η_⊥(x_1)} | c ∈ D_0}
        f_1 ← {c ∪ {x_1, ¬η_⊥(x_1)} | c ∈ D_1}
        return Absorp(f_⊥ ∪ f_0 ∪ f_1)
    end if
end procedure
```

The algorithm returns a set of sets of literals, i.e., either possibly negated variables or possibly negated absence tests representing a disjunctive normal form. $\mathsf{Absorp}(\mathcal{D})$ removes all cubes in $\mathcal{D}$ that are strict supersets of other cubes in $\mathcal{D}$.

IV. Decompositions for Monotone Functions

Theorem 2 can be used to convert any ternary function into a propositional logic formula that is even in disjunctive normal form. However, each product term of the DNF contains an absence test $\eta_\bot(x_i)$ for each variable in the clause. For many function classes, especially for the monotone functions, such an absence test is not necessary. Therefore, this section presents new decomposition theorems for monotone ternary functions that do not require the absence test.

We start with a first theorem that shows the limits of the Shannon decomposition in ternary logic, so that we can see what other parts of the decomposition are needed.

Theorem 3 (DC1). *For any monotone function $f : \mathbb{T}^{n+1} \to \mathbb{T}$, the following are equivalent:*

- *there are monotone functions f_0,$f_1 : \mathbb{T}^n \to \mathbb{T}$ such that the following holds: $f(x,y) = f_0(y) \wedge \neg x \vee f_1(y) \wedge x$*
- *for all $y \in \mathbb{T}^n$, we have*

$$f(\bot, y) = \begin{cases} 0 & : \text{if } f(1,y) = f(0,y) = 0 \\ \bot & : \text{otherwise} \end{cases}$$

Proof. We prove the two implications as follows:

$\Rightarrow$: For the functions f_0 and f_1 in the decomposition, we obviously have $f(1,y) = f_1(y)$, $f(0,y) = f_0(y)$ and

$$f(\bot, y) = (f_0(y) \vee f_1(y)) \wedge \bot = \begin{cases} 0 & : \text{if } f_1(y) = f_0(y) = 0 \\ \bot & : \text{otherwise} \end{cases}$$

$\Leftarrow$: For any given function f with the mentioned property, we define $f_0(y) := f(0,y)$ and $f_1(y) := f(1,y)$ so that the decomposition holds. □

Although the set of functions that have a DC1 decomposition is quite large, it is rather limited in terms of interesting functions: Not even the sequential functions belong to this set. In particular, this set of functions is neither closed under conjunction nor under negation, since both operations would produce disjunctive normal forms with product terms containing complementary pairs like $\neg x \wedge x$. This leads to the following extension:

Theorem 4 (DC2). *For any monotone function $f : \mathbb{T}^{n+1} \to \mathbb{T}$, the following are equivalent:*

- *there are monotone functions f_0,f_1,$f_2 : \mathbb{T}^n \to \mathbb{T}$ such that the following holds:*

$$f(x,y) = f_0(y) \wedge \neg x \vee f_1(y) \wedge x \vee f_2(y) \wedge \neg x \wedge x$$

- *for all $y \in \mathbb{T}^n$, we have $f(\bot, y) \neq 1$*

Proof. Note that $f(\bot, y) = 0$ implies $f(1,y) = f(0,y) = 0$ for any montone function f.

$\Rightarrow$: For the functions f_0, f_1, and f_2 in the decomposition, we obviously have $f(1,y) = f_1(y)$, $f(0,y) = f_0(y)$ and

$$f(\bot, y) = (f_0(y) \vee f_1(y) \vee f_2(y)) \wedge \bot = \begin{cases} 0 & : \text{if } f_2(y) = f_1(y) = f_0(y) = 0 \\ \bot & : \text{otherwise} \end{cases}$$

$\Leftarrow$: For any given function f with the mentioned property, we define $f_0(y) := f(0,y)$, $f_1(y) := f(1,y)$, and $f_2(y) = f(\bot, y)$ so that the decomposition holds. □

To understand the need for f_2, consider the following: If $f(1,y) \neq f(0,y)$ holds, then we have $f(\bot, y) = \bot$ for any monotone function f. However, if $f(1,y) = f(0,y) = b \in \mathbb{B}$ holds, then we can have either $f(\bot, y) = \bot$ or $f(\bot, y) = b$. The DC1-functions are forced to choose $f(\bot, y) = 0$ for $f_0(y) = f_1(y) = 0$, while DC2-functions also allow $f(\bot, y) = \bot$ in this case by choosing $f_2(y) \neq 0$. This can be seen from the following case distinctions for DC2-functions (which also hold for DC1-functions for $f_2(y) = 0$):

$$f(x,y) = \begin{cases} f_0(y) & : \text{if } x = 0 \\ f_1(y) & : \text{if } x = 1 \\ 0 & : \text{if } x = \bot \text{ and } f_2(y) = f_1(y) = f_0(y) = 0 \\ \bot & : \text{otherwise} \end{cases}$$

Therefore, f_2 removes the restriction to determine $f(\bot, y) = 0$ if $f(1,y) = f(0,y) = 0$ holds. However, as one can see from the cases above, we can never have $f(\bot, y) = 1$ for DC2-functions. This leads to the DC3-functions:

Theorem 5 (DC3). *For any monotone function $f : \mathbb{T}^{n+1} \to \mathbb{T}$, the following are equivalent:*

- *there are monotone functions f_0,f_1,$f_3 : \mathbb{T}^n \to \mathbb{T}$ such that the following holds:*

$$f(x,y) = f_0(y) \wedge \neg x \vee f_1(y) \wedge x \vee f_3(y)$$

- *for all $y \in \mathbb{T}^n$, $f(1,y) = f(0,y) = 0$ implies $f(\bot, y) = 0$*

Proof. $\Rightarrow$: For the functions f_0, f_1, and f_3 in the decomposition, we obviously have $f(0,y) = f_0(y) \vee f_3(y)$, $f(1,y) = f_1(y) \vee f_3(y)$, and

$$f(\bot, y) = (f_0(y) \vee f_1(y)) \wedge \bot \vee f_3(y) = \begin{cases} 1 & : \text{if } f_3(y) = 1 \\ 0 & : \text{if } f_3(y) = f_1(y) = f_0(y) = 0 \\ \bot & : \text{otherwise} \end{cases}$$

$\Leftarrow$: For any given function f with the mentioned property, we define $f_0(y) := f(0,y)$, $f_1(y) := f(1,y)$, and $f_3(y) := f(\bot,y) \wedge f(0,y) \wedge f(1,y)$ so that the decomposition holds. □

As can be seen from the proof of the above theorem, it is possible for DC3-functions to have $f(\bot,y) = 1$ which is not possible for DC2-functions. One may wonder why the above theorem cannot be generalized similarly to Theorem 1. To discuss this, we can first prove the following theorem:

Theorem 6. *For any monotone function $f : \mathbb{T}^{n+1} \to \mathbb{T}$, there are monotone functions f_0, f_1, $f_\bot$ such that the following holds for all $y \in \mathbb{T}^n$:*

$$f(x,y) = f_0(y) \wedge \neg x \vee f_1(y) \wedge x \vee f_\bot(y) \wedge \eta_\bot(x)$$

Proof. Obviously, we can use $f_0(y) := f(0,y)$, $f_1(y) := f(1,y)$, and $f_\bot(y) := f(\bot,y)$ for the decomposition. □

The above ternary decomposition is possible for all monotone ternary functions. However, for some monotone functions f_0, f_1, $f_\bot$, the function f defined in the theorem is not monotone, e.g., the following one is not monotone

$$f(x,y) = \bot \wedge \neg x \vee \bot \wedge x \vee (\bot \vee y) \wedge \eta_\bot(x) = \bot \vee y \wedge \eta_\bot(x)$$

For this reason, the above decomposition is not useful for monotone functions in the sense that we cannot use it to enumerate the monotone functions by composing already given monotone functions with fewer variables. Finally, we therefore have the final theorem for a complete monotone function decomposition:

Theorem 7 (DC4). *For any monotone function $f : \mathbb{T}^{n+1} \to \mathbb{T}$, there are monotone functions f_0,f_1,f_2,$f_3 : \mathbb{T}^n \to \mathbb{T}$ such that the following holds:*

$$f(x,y) = f_0(y) \wedge \neg x \vee f_1(y) \wedge x \vee f_2(y) \wedge \neg x \wedge x \vee f_3(y)$$

In particular, functions $f_0(y) := f(0,y)$, $f_1(y) := f(1,y)$, $f_2(y) := f(\bot,y)$, and $f_3(y) := f(\bot,y) \wedge f(0,y) \wedge f(1,y)$ satisfy the equation.

Proof. First of all, note that all functions f as defined in the theorem are certainly monotone. Moreover, if we define for a given monotone function f, the functions $f_0(y) := f(0,y)$, $f_1(y) := f(1,y)$, $f_2(y) := f(\bot,y)$, and $f_3(y) := f(\bot,y) \wedge f(0,y) \wedge f(1,y)$, then we get by the absorption theorem

- $f(1,y) = f_1(y) \vee f_3(y)$
- $f(0,y) = f_0(y) \vee f_3(y)$
- $f(\bot,y) = (f_0(y) \vee f_1(y) \vee f_2(y)) \wedge \bot \vee f_3(y)$, i.e.,

$$f(\bot,y) = \begin{cases} 1 & : \text{if } f_3(y) = 1 \\ 0 & : \text{if } f_3(y) = f_2(y) = f_1(y) = f_0(y) = 0 \\ \bot & : \text{otherwise} \end{cases}$$

The above equation holds, since $f(\bot,y) = b \in \mathbb{B}$ implies $f(1,y) = f(0,y) = b$ for any monotone function f. Thus, we have $f(\bot,y) = 1 \Leftrightarrow f_3(y) = 1$, $f(\bot,y) = 0 \Leftrightarrow f_3(y) = f_2(y) = f_1(y) = f_0(y) = 0$. Finally, $f(\bot,y) = \bot$ implies $f_2(y) = \bot$, and $(f_0(y) \vee f_1(y) \vee \bot) \wedge \bot \vee \bot \wedge f(0,y) \wedge f(1,y)$ is always equal to $\bot$. □

For any DC4-function, we therefore have the following equation:

$$f(x,y) = \begin{cases} f_0(y) \vee f_3(y) & : \text{if } x = 0 \\ f_1(y) \vee f_3(y) & : \text{if } x = 1 \\ 1 & : \text{if } x = \bot \text{ and } f_i(y) = 1 \text{ for all } i \\ 0 & : \text{if } x = \bot \text{ and } f_i(y) = 0 \text{ for all } i \\ \bot & : \text{otherwise} \end{cases}$$

As can be seen, f_3 determines function values $f(\bot,y) = 1$ by also enforcing $f(0,y) = f(1,y) = 1$, and that f_2 and f_3 together determine the function value $f(\bot,y) = 0$ by enforcing $f(0,y) = f(1,y) = 0$. Additional function values 0 and 1 are added for $x = 0$ by f_0 and for $x = 1$ by f_1, respectively.

The above decomposition theorem may be unexpected since we need four functions f_0,f_1,f_2,f_3 to construct a function f with one additional argument variable. Typically, for ternary logic, one expects three cases, and thus three sub-functions as in Theorem 1, but this is not useful for constructing the monotone functions as we have outlined step by step with the weaker decompositions.

Algorithm 2 Converting Monotone Function Tables to DNFs

```
procedure MTab2DNF(n, f)
    if n = 0 then
        if f = ⊥ then return {{⊥}}
        else if f = 0 then return {}
        else return {{}}
        end if
    else
        D_⊥ ← MTab2DNF(n − 1, f(⊥, x_2, ..., x_n))
        D_0 ← MTab2DNF(n − 1, f(0, x_2, ..., x_n))
        D_1 ← MTab2DNF(n − 1, f(1, x_2, ..., x_n))
        f_0 ← {c ∪ {¬x_1} | c ∈ D_0}
        f_1 ← {c ∪ {x_1} | c ∈ D_1}
        f_2 ← {(c \ {⊥}) ∪ {¬x_1, x_1} | c ∈ D_⊥}
        f_3 ← {c_⊥ ∪ c_0 ∪ c_1 | c_⊥ ∈ D_⊥, c_0 ∈ D_0, c_1 ∈ D_1}
        return Absorp(f_0 ∪ f_1 ∪ f_2 ∪ {c ∈ f_3 | ⊥ ∉ c})
    end if
end procedure
```

The DC4-decomposition can be used to construct Algorithm 2 which translates a monotone ternary function table into a propositional logic formula in disjunctive normal form. The algorithm is very similar to Algorithm 1. In contrast to Algorithm 1, Algorithm 2 does not require the non-monotone absence tests, and computes the four functions f_0,f_1,f_2,f_3 according to Theorem 7 instead. As in Algorithm 1, Absorp($\mathcal{D}$) removes all cubes in $\mathcal{D}$ that are strict supersets of other cubes in $\mathcal{D}$. In addition, it can remove constants $\bot$, can merge cubes by the following equivalences, and removes cubes with constant $\bot$ from f_3 since these cannot become 1 (recall Theorem 4):

- $\bot \wedge \varphi \wedge \neg\varphi \Leftrightarrow \varphi \wedge \neg\varphi$ and $\bot \vee \varphi \vee \neg\varphi \Leftrightarrow \varphi \vee \neg\varphi$
- $x \wedge \varphi \wedge \neg\varphi \vee \neg x \wedge \varphi \wedge \neg\varphi \Leftrightarrow \varphi \wedge \neg\varphi$
- $\bot \wedge x \wedge \varphi \vee \bot \wedge \neg x \wedge \varphi \Leftrightarrow \bot \wedge \varphi$

V. Related Work

A. Regular Ternary and B-Ternary Functions

Theorem 7 and Algorithm 2 directly imply the following theorem which in turn implies that propositional logic with constants $\bot$, 0, 1 exactly corresponds with the monotone ternary functions (see also [27]):

Theorem 8 (Syntactic Representation of Monotone Ternary Functions)**.** *The set of monotone ternary functions can be recursively constructed as follows:*

- *monotone ternary functions with no variables are the constants* $\bot$, 0, 1
- *every monotone ternary function* f *with* $n+1$ *variables can be written as* $f(x,y) = f_0(y) \wedge \neg x \vee f_1(y) \wedge x \vee f_2(y) \wedge \neg x \wedge x \vee f_3(y)$ *with monotone ternary functions* f_0, f_1, f_2, f_3 *on* n *variables* y

Proof. The proof proceeds by induction on the number of variables: The induction base is clear, and the induction step follows by the induction hypothesis and Theorem 7. □

By Algorithm 2, we further see that every monotone function can be written as a DNF with complementary terms $x \wedge \neg x$ as well as the constant $\bot$ which was proved differently in [27]. However, Algorithm 2 computes formulas different to those in [27].

In previous work, Mukaidono also considered B-ternary functions which is the strict subset of monotone ternary functions that map boolean vectors $x \in \mathbb{B}^n$ to boolean values $f(x) \in \mathbb{B}$. In [24], [25], Mukaidono proved that the B-ternary functions exactly correspond with the propositional logic formulas without the constant $\bot$. Hence, the following theorem can be concluded as well:

Theorem 9 (Syntactic Representation of B-Ternary Functions)**.** *The set of B-ternary functions can be recursively constructed as follows:*

- *B-ternary functions with no variables are constants* 0, 1
- *every B-ternary function function* f *with* $n+1$ *variables can be written as* $f(x,y) = f_0(y) \wedge \neg x \vee f_1(y) \wedge x \vee f_2(y) \wedge \neg x \wedge x \vee f_3(y)$ *with B-ternary functions* f_0, f_1, f_2, f_3 *on* n *variables* y

While $f(0,y)$ and $f(1,y)$ are B-ternary if f is B-ternary, $f(\bot,y)$ may not be B-ternary so that $\mathcal{D}_\bot$ may contain cubes with constant $\bot$. However, Algorithm 2 removes all constants $\bot$ for B-ternary functions: Adding $\neg x \wedge x$ in f_2 eliminates the constants $\bot$ in f_2 anyway, and since cubes with constant $\bot$ cannot become 1, these are also removed from f_3 (recall Theorem 4).

B. Sequential Functions

The set of sequential functions is a very important subset of the monotone functions. Intuitively, sequential functions can avoid failures in the evaluation of expressions like $x > 0 \wedge \frac{y}{x} > x$ by evaluating the left side $x > 0$ first and only if it is true, the right side is evaluated as well.

Definition 2 (Sequential Functions)**.** *The set of sequential ternary functions can be recursively constructed as follows:*

- *sequential ternary functions with no variables are the constants* $\bot$, 0, 1
- *any sequential ternary function with* $n+1$ *variables can be written as* $f(x,y) = (x \Rrightarrow f_1(y) \mid f_0(y))$ *with monotone ternary functions* f_0 *and* f_1 *on* n *variables* y

Functions that are not sequential are called parallel.

It can be easily proved that all sequential functions are monotone and closed under composition. Hence, the sequential functions are a strict subset of the monotone functions:

Theorem 10 (Sequential Functions)**.** *All sequential functions allow a* DC2 *decomposition, but there are* DC2 *functions that are not sequential.*

Proof. The sequential ITE operator can be expressed by $\neg$, $\wedge$, and $\vee$ as follows:

$$(\gamma \Rightarrow \varphi_1 \mid \varphi_0) \Leftrightarrow \varphi_0 \wedge \neg\gamma \vee \varphi_1 \wedge \gamma \vee \neg\gamma \wedge \gamma$$

Hence, all sequential functions are special DC2 functions where $f_2(y) := 1$ holds. A DC2 function which is not sequential is for example $f(x,y) := x \wedge \neg x \wedge y \wedge \neg y$. □

Parallel ITE can also be expressed by $\neg$, $\wedge$, and $\vee$:

$$(\gamma \Rightarrow \varphi_1 \mid \varphi_0) \Leftrightarrow \varphi_0 \wedge \neg\gamma \vee \varphi_1 \wedge \gamma \vee \varphi_0 \wedge \varphi_1$$

and conversely, parallel ITE can express negation, conjunction, and disjunction[2]. However, if we would only consider the functions that were obtained by parallel ITE $f(x,y) := (x \Rrightarrow f_1(y) \mid f_0(y))$ with monotone functions f_0 and f_1, we would only obtain a strict subset of the DC3 functions, so that parallel ITE is not directly useful for the enumeration of all monotone functions.

VI. Conclusions

The monotone ternary functions have undoubtedly the most applications among the subsets of ternary functions considered so far. This paper presents a new decomposition theorem for monotone ternary functions that splits any given monotone ternary function into four functions by eliminating one of the argument variables. Conversely, we can construct all monotone ternary functions by combining four arbitrary monotone ternary functions with a new variable according to the decomposition theorem. This way, we can enumerate all monotone ternary functions, and therefore we can use it for the construction of recursive algorithms on monotone ternary functions that eliminate one of the variables used in each recursion step. As an example, we have described a recursive algorithm that translates a given monotone function into a propositional logic formula, and in this way, we easily prove Mukaidono's theorems on disjunctive normal forms of regular ternary, i.e., monotone, and B-ternary functions.

[2]We have the following equivalences:

- $\neg\varphi \Leftrightarrow (\varphi \Rrightarrow 0 \mid 1)$
- $\varphi \wedge \psi \Leftrightarrow (\varphi \Rrightarrow \psi \mid 0)$
- $\varphi \vee \psi \Leftrightarrow (\varphi \Rrightarrow 1 \mid \psi)$
- $\mathsf{inf}(\varphi,\psi) \Leftrightarrow (\bot \Rrightarrow \varphi \mid \psi)$

References

[1] Becker, B., Drechsler, R., and Theobald, M. On the expressive power of OKFDDs. *Formal Methods in System Design (FMSD) 11*, 1 (July 1997), 5–21.

[2] Berry, G. Stable models of typed lambda-calculi. In *International Colloquium on Automata, Languages and Programming (ICALP)* (Udine, Italy, 1978), G. Ausiello and C. Böhm, Eds., vol. 62 of *LNCS*, Springer, pp. 72–89.

[3] Berry, G. The constructive semantics of pure Esterel, July 1999.

[4] Berry, G., Curien, P., and Levy, J.-J. Full abstraction for sequential languages: The state of the art. Technical Report 197, Institut National de Recherche en Informatique et en Automatique (INRIA), Le Chesnay, Cedex, France, March 1983.

[5] Bredeson, J., and Hulina, P. Elimination of static and dynamic hazards for multiple input changes in combinational switching circuits. *Information and Control 20* (1972), 114–124.

[6] Bryant, R. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers (T-C) 35*, 8 (August 1986), 677–691.

[7] Bryant, R., and Seger, C.-J. Formal verification of digital circuits using symbolic ternary system models. In *Computer Aided Verification (CAV)* (New Brunswick, New Jersey, USA, 1991), E. Clarke and R. Kurshan, Eds., vol. 531 of *LNCS*, Springer, pp. 33–43.

[8] Brzozowski, J. Delay-insensitivity and ternary simulation. *Theoretical Computer Science (TCS) 245*, 1 (August 2000), 3–25.

[9] Brzozowski, J., and Yoeli, M. On a ternary model of gate networks. *IEEE Transactions on Computers 28*, 3 (March 1979), 178–184.

[10] Chenga, D., Liua, Z., and Qi, H. Completeness and normal form of multi-valued logical functions. *Journal of the Franklin Institute 357* (2020), 9871–9884.

[11] Drechsler, R., Sarabi, A., Theobald, M., Becker, B., and Perkowski, M. Efficient representation and manipulation of switching functions based on ordered Kronecker functional decision diagrams. In *Design Automation Conference (DAC)* (San Diego, California, USA, 1994), ACM, pp. 415–419.

[12] Eichelberger, E. Hazard detection in combinational and sequential switching circuits. *IBM Journal of Research and Development 9* (1965), 90–99.

[13] Fitting, M. A Kripke-Kleene semantics for logic programs. *Journal of Logic Programming 2*, 4 (December 1985), 295–312.

[14] Fitting, M. Kleene's logic, generalized. *Journal of Logic and Computation 1*, 6 (1991), 797–810.

[15] Fitting, M. Well-founded semantics, generalized. In *Logic Programming* (San Diego, California, USA, 1991), V. Saraswat and K. Ueda, Eds., MIT Press, pp. 71–84.

[16] Fitting, M. The family of stable models. *The Journal of Logic Programming 17*, 2-4 (November 1993), 197–225.

[17] Ikenmeyer, C., Komarath, B., Lenzen, C., Lysikov, V., Mokhov, A., and Sreenivasaiah, K. On the complexity of hazard-free circuits. *Journal of the ACM (JACM) 66*, 4 (2019), 25:1–25:20.

[18] Jukna, S. Notes on hazard-free circuits. *SIAM Journal on Discrete Mathematics 35*, 2 (2021), 770–787.

[19] Kebschull, U., and Rosenstiel, W. Efficient graph-based computation and manipulation of functional decision diagrams. In *European Design and Test Conference (EDTC)* (Paris, France, 1993), F. Poirot and A.-M. Trullemans, Eds., IEEE Computer Society, pp. 278–283.

[20] Kleene, S. On notation for ordinal numbers. *The Journal of Symbolic Logic 3*, 4 (December 1938), 150–155.

[21] Merrill, R. Ternary logic in digital computers. In *Design Automation Conference (DAC)* (Atlantic City, NJ, USA, 1965), ACM.

[22] Miller, D., and Thornton, M. *Multiple Valued Logic: Concepts and Representations*, vol. 12 of *Synthesis Lectures on Digital Circuits and Systems*. Springer Nature, 2022.

[23] Milner, R. Fully abstract models of typed λ-calculi. *Theoretical Computer Science (TCS) 4*, 1 (February 1977), 1–22.

[24] Mukaidono, M. On the B-ternary logical function - a ternary logic considering ambiguity. *Systems, Computers, Controls 3*, 3 (1972), 27–36.

[25] Mukaidono, M. The B-ternary logic and its applications to the detection of hazards in combinational switching circuits. In *International Symposium on Multiple-Valued Logic (ISMVL)* (Rosemont, IL, USA, 1978), IEEE Computer Society, pp. 269–275.

[26] Mukaidono, M. Advanced results on applications of fuzzy switching functions to hazard detection. In *Advances in Fuzzy Sets, Possibility Theory, and Applications* (1983), P. Wang, Ed., Springer, pp. 335–349.

[27] Mukaidono, M. Regular ternary logic functions - ternary logic functions suitable for treating ambiguity. *IEEE Transactions on Computers C-35*, 2 (1986), 179–183.

[28] Plotkin, G. LCF considered as a programming language. *Theoretical Computer Science (TCS) 5*, 3 (December 1977), 223–255.

[29] Plotkin, G. A structural approach to operational semantics. Tech. Rep. FN-19, DAIMI, Århus, Denmark, 1981.

[30] Pogosyan, G. Efficiently irreducible bases in multiple-valued logic. In *International Symposium on Multiple-Valued Logic (ISMVL)* (Santiago de Compostela, Spain, 1996), IEEE Computer Society, pp. 296–301.

[31] Post, E. Introduction to a general theory of elementary propositions. *American Journal of Mathematics 43*, 3 (1921), 163–185. http://dx.doi.org/10.2307/2370324.

[32] Prior, A. Three-valued logic and future contingents. *The Philosophical Quarterly 3*, 13 (October 1953), 317–326.

[33] Sazonov, V. Sequentially and parallely computable functionals. In *Lambda-Calculus and Computer Science Theory* (Rome, Italy, 1975), C. Böhm, Ed., vol. 37 of *LNCS*, Springer, pp. 312–318.

[34] Sazonov, V. Degrees of parallelism in computations. In *Mathematical Foundations of Computer Science (MFCS)* (Gdansk, Poland, 1976), A. Mazurkiewicz, Ed., vol. 45 of *LNCS*, Springer, pp. 517–523.

[35] Sazonov, V. Expressibility of functions in D. Scott's LCF language. *Algebra and Logic 15* (1976), 192–206.

[36] Scott, D. Outline of a mathematical theory of computation. Technical Monograph PRG-2, Oxford University, Computing Laboratory, Programming Research Group, Oxford, UK, November 1970.

[37] Scott, D. A type-theoretical alternative to ISWIM, CUCH, OWHY. *Theoretical Computer Science (TCS) 121*, 1-2 (December 1993), 411–440.

[38] Stanković, R., Astola, J., and Moraga, C. *Representation of Multiple-Valued Logic Functions*, vol. 37 of *Synthesis Lectures on Digital Circuits and Systems*. Springer Nature, 2022.

[39] Stojmenovic, I. Completeness criteria in many-valued set logic under compositions with boolean functions. In *International Symposium on Multiple-Valued Logic (ISMVL)* (Boston, MA, USA, 1994), IEEE Computer Society, pp. 177–183.

[40] Takagi, N., Nakashima, K., Kikuchi, H., and Mukaidono, M. A characterization of Kleenean functions. In *International Symposium on Multiple-Valued Logic (ISMVL)* (Bloomington, IN, USA, 1995), IEEE Computer Society, pp. 236–241.

[41] Takagi, N., Nakashima, K., and Mukaidono, M. A canonical disjunctive form of extended Kleene-Stone logic functions. In *International Symposium on Multiple-Valued Logic (ISMVL)* (Sacramento, CA, USA, 1993), IEEE Computer Society, pp. 36–41.

[42] Takagi, N., Nakashima, K., and Mukaidono, M. Minimization for Kleene-Stone logic functions. In *International Symposium on Multiple-Valued Logic (ISMVL)* (Boston, MA, USA, 1994), IEEE Computer Society, pp. 124–131.

[43] Takagi, N., Nakashima, K., and Mukaidono, M. A necessary and sufficient condition for lukasiewicz logic functions. In *International Symposium on Multiple-Valued Logic (ISMVL)* (Santiago de Compostela, Spain, 1996), IEEE Computer Society, pp. 37–42.

[44] Tatsumi, H., Miyakawa, M., and Mukaidono, M. Upper and lower bounds on the number of disjunctive forms. In *International Symposium on Multiple-Valued Logic (ISMVL)* (Singapore, 2006), IEEE Computer Society.

[45] Trakhtenbrot, M. On representation of sequential and parallel functions. In *Mathematical Foundations of Computer Science (MFCS)* (Mariánské Lázně, Poland, 1975), J. Bečvář, Ed., vol. 32 of *LNCS*, Springer, pp. 411–417.

[46] Trakhtenbrot, M. Recursive program schemes and computable functionals. In *Mathematical Foundations of Computer Science (MFCS)* (Gdansk, Poland, 1976), A. Mazurkiewicz, Ed., vol. 45 of *LNCS*, Springer, pp. 137–152.

[47] Trakhtenbrot, M. Relationships between classes of monotonic functions. *Theoretical Computer Science (TCS) 2*, 2 (1976), 225–247.

[48] Unger, S. Hazards and delays in asynchronous sequential switching circuits. *IRE Transactions on Circuit Theory 6*, 1 (March 1959), 12–25.

[49] Yamamoto, Y., and Mukaidono, M. Meaningful special classes of ternary logic functions - regular ternary logic functions and ternary majority functions. *IEEE Transactions on Computers 37*, 7 (1988), 799–806.

[50] Yoeli, M., and Rinon, S. Application of ternary algebra to the study of static hazards. *Journal of the ACM (JACM) 11* (1964), 84–97.