Part 3. Arithmetic within the VOID Granularity Framework

Konrad Wojnowski, PhD
Faculty of Polish Studies, Jagiellonian University
ChatGPT
Claude.ai

October 7, 2024

1 Introduction

In this chapter, we develop Arithmetic within the VOID Granularity Framework (VGF), building upon the Number Theory established in Chapter 3. Arithmetic operations form the foundational tools for mathematical computation and reasoning, and adapting them to the VGF ensures consistency across the framework. We aim to define arithmetic operations that respect the minimal meaningful distinction, δ_{VOID}, and integrate seamlessly with the VOID numbers previously defined. By doing so, we bridge Number Theory and Algebra within the VGF, providing a cohesive mathematical structure.

2 Referencing Existing Framework

To ensure continuity, we reference key axioms and definitions from previous chapters, using their designated numbers:

- **Axiom 1.1 (Universal VOID Granularity)**: Establishes δ_{VOID} as the minimal meaningful distinction in any system, affecting the distinguishability of elements.
- **Axiom 1.2 (Onto-Epistemological Granularity)**: Links ontological and epistemological distinctions, asserting that limitations in distinguishing elements are inherent to both their existence and our knowledge.
- **Definition 5 (Distinguishability Function $\mu_S(x, y)$)**: Provides a measure of the probability that two elements x and y are distinguishable based on δ_{VOID}.

We also build upon the definitions of VOID Natural Numbers, VOID Integers, and VOID Real Numbers introduced in Chapter 3, where numbers represent quantifiable changes exceeding δ_{VOID}.

3 Establishing Continuity

Arithmetic operations must align with the properties of VOID numbers and the constraints imposed by δ_{VOID}. Since VOID numbers are grounded in observable changes within dynamic systems, arithmetic operations should respect the finite granularity and probabilistic nature of the VGF. This means that operations like addition and multiplication need to be redefined to incorporate the VOID threshold. By establishing this continuity, we ensure that arithmetic within the VGF is consistent with the foundational principles laid out in earlier chapters.

4 Essential Definitions in VGF Arithmetic

4.1 VOID Rounding Function

Definition 4.1 (VOID Rounding Function, void_round(x)):

For any real number $x \in R_{\text{VOID}}$, the VOID Rounding Function rounds x to the nearest multiple of δ_{VOID}:

$$\text{void\_round}(x) = \delta_{\text{VOID}} \times \left\lfloor \frac{x}{\delta_{\text{VOID}}} + \frac{1}{2} \right\rfloor$$

This function ensures that all numerical values conform to the granularity imposed by δ_{VOID}, as per Axiom 1.1. It adjusts values to the nearest distinguishable quantity within the system, maintaining consistency with the concept of finite precision.

4.2 Distinguishability Criterion

Definition 4.2 (Distinguishability Criterion):

Two numbers x and y are VOID-distinguishable if:

$$|x - y| \geq \delta_{\text{VOID}}$$

Otherwise, they are considered indistinguishable within the VGF. This criterion is based on the Distinguishability Function $\mu_S(x, y)$ from Definition 5, which quantifies the probability of distinguishing between x and y. It ensures that arithmetic operations result in values that are meaningfully different within the framework.

4.3 VOID Addition ($\oplus_{\text{VOID}}$)

Definition 4.3 (VOID Addition):

For $x, y \in R_{\text{VOID}}$, the VOID Addition is defined as:

$$x \oplus_{\text{VOID}} y = \text{void\_round}(x + y)$$

This operation adds x and y using standard addition, then rounds the result using the VOID Rounding Function to ensure it adheres to δ_{VOID}. It aligns with the need for operations to produce distinguishable results within the VGF.

4.4 VOID Subtraction ($\ominus_{\text{VOID}}$)

Definition 4.4 (VOID Subtraction):
For $x, y \in R_{\text{VOID}}$, the VOID Subtraction is defined as:

$$x \ominus_{\text{VOID}} y = \text{void\_round}(x - y)$$

Similar to VOID Addition, this operation subtracts y from x and rounds the result, ensuring that the difference is a distinguishable quantity within the framework.

4.5 VOID Multiplication ($\otimes_{\text{VOID}}$)

Definition 4.5 (VOID Multiplication):
For $x, y \in R_{\text{VOID}}$, the VOID Multiplication is defined as:

$$x \otimes_{\text{VOID}} y = \text{void\_round}(x \times y)$$

This operation multiplies x and y and applies the VOID Rounding Function to maintain adherence to δ_{VOID}. It ensures that the product is a value that can be meaningfully represented within the VGF.

4.6 VOID Division ($\oslash_{\text{VOID}}$)

Definition 4.6 (VOID Division):
For $x, y \in R_{\text{VOID}}$ with $y \neq 0$, the VOID Division is defined as:

$$x \oslash_{\text{VOID}} y = \text{void\_round}\left(\frac{x}{y}\right)$$

Division by zero remains undefined. This operation divides x by y and rounds the result, ensuring that the quotient conforms to the granularity threshold δ_{VOID}.

5 Core Principles to Establish

5.1 Properties of VOID Arithmetic Operations

We examine how basic arithmetic properties are affected by the VOID adjustments:

- **Associativity**: Due to the application of the VOID Rounding Function after each operation, associativity may not hold strictly. For example, $(x \oplus_{\text{VOID}} y) \oplus_{\text{VOID}} z$ may not equal $x \oplus_{\text{VOID}} (y \oplus_{\text{VOID}} z)$.

- **Commutativity**: VOID Addition and VOID Multiplication remain commutative because the order of operands does not affect the standard operation or the rounding process.
- **Distributivity**: The distributive law may not hold precisely due to rounding effects. Specifically, $x \otimes_{\text{VOID}} (y \oplus_{\text{VOID}} z)$ may not equal $(x \otimes_{\text{VOID}} y) \oplus_{\text{VOID}} (x \otimes_{\text{VOID}} z)$.

These deviations highlight the impact of finite granularity on arithmetic properties and emphasize the importance of considering δ_{VOID} in mathematical reasoning within the VGF.

5.2 Relationship to Regular Arithmetic

VOID arithmetic approximates regular arithmetic within the precision limit imposed by δ_{VOID}. As δ_{VOID} approaches zero, VOID arithmetic operations converge to standard arithmetic operations. This demonstrates that the VGF is an extension of classical mathematics, adjusted to account for finite precision and distinguishability constraints.

6 Key Problems to Solve

6.1 Rounding and Precision

Challenge: Ensuring that all numerical values produced by arithmetic operations respect δ_{VOID}.

Solution: Utilize the VOID Rounding Function after each operation to adjust results to the nearest multiple of δ_{VOID}. This approach maintains consistency with Axiom 1.1 and ensures that all values are representable within the framework's finite precision.

6.2 Distinguishability

Challenge: Determining when two numerical results are considered different within the VGF.

Solution: Apply the Distinguishability Criterion from Definition 4.2, which states that two numbers are distinguishable if their difference is at least δ_{VOID}. This criterion ensures that comparisons and equality checks are meaningful within the context of the VGF.

6.3 Edge Cases

Challenge: Handling operations resulting in very small numbers, such as those approaching zero or exceeding representable limits.

Solution:

- **Underflow**: If the result of an operation is less than δ_{VOID}, it is considered indistinguishable from zero and assigned the value zero.
- **Overflow**: If an operation results in a value beyond the representable range, standard overflow handling techniques are applied, consistent with the system's limitations.

These solutions ensure that arithmetic operations remain robust and consistent, even in edge cases.

7 Practical Considerations

7.1 Algorithms for VGF Arithmetic Operations

We provide practical algorithms for implementing VOID arithmetic operations:

- **VOID Rounding Function**:

```
def void_round(x, delta_VOID):
    return round(x / delta_VOID) * delta_VOID
```

- **VOID Addition**:

```
def void_add(x, y, delta_VOID):
    return void_round(x + y, delta_VOID)
```

- **VOID Subtraction**:

```
def void_subtract(x, y, delta_VOID):
    return void_round(x - y, delta_VOID)
```

- **VOID Multiplication**:

```
def void_multiply(x, y, delta_VOID):
    return void_round(x * y, delta_VOID)
```

- **VOID Division**:

```
def void_divide(x, y, delta_VOID):
    if y == 0:
        raise ValueError("Division by zero is undefined.")
    return void_round(x / y, delta_VOID)
```

These algorithms demonstrate how VOID arithmetic can be implemented in computational systems, respecting the granularity threshold δ_{VOID}.

7.2 Examples

7.2.1 Addition Example

Compute $2.3 + 1.7$ with $\delta_{\text{VOID}} = 0.05$:

1. Standard Addition: $2.3 + 1.7 = 4.0$.
2. VOID Rounding: void_round$(4.0) = 4.0$.
3. Result: 4.0.

This example illustrates how VOID Addition adjusts the result to align with δ_{VOID}.

7.2.2 Multiplication Example

Compute 2.5×3.0 with $\delta_{\text{VOID}} = 0.1$:

1. Standard Multiplication: $2.5 \times 3.0 = 7.5$.
2. VOID Rounding: void_round$(7.5) = 7.5$.
3. Result: 7.5.

In this case, the result is already aligned with δ_{VOID}, so no adjustment is needed.

7.2.3 Division Example

Compute $5.0 \div 2.0$ with $\delta_{\text{VOID}} = 0.1$:

1. Standard Division: $5.0 \div 2.0 = 2.5$.
2. VOID Rounding: void_round$(2.5) = 2.5$.
3. Result: 2.5.

This example shows how VOID Division produces a result consistent with the VGF's finite precision.

8 Conclusion

In this chapter, we have developed arithmetic operations within the VOID Granularity Framework, ensuring that they respect the minimal meaningful distinction δ_{VOID} and integrate seamlessly with the VOID numbers defined in Number Theory. By redefining arithmetic operations and examining their properties, we have provided a consistent and coherent mathematical structure within the VGF. This development bridges Number Theory and Algebra, laying the groundwork for further exploration of mathematical concepts within the framework and demonstrating the practicality of applying VGF principles to fundamental mathematical operations.