A Proof-of-Concept for the Reserve Arithmetic System (RAS): A Symbolic Model for Division by Zero

Edrianne Paul B. Casinillo
edzcasinillo@gmail.com
Independent Researcher

Abstract

The Reserve Arithmetic System (RAS) introduces a novel approach to handling division by zero by storing the numerator as a reserve in a symbolic zero. This document provides a proof-of-concept demonstration of the core mechanics of RAS, including its arithmetic properties and operational rules. The notation $0_{\langle x\rangle}$ denotes "zero with reserve x," indicating that the numerical output is zero while the numerator is conceptually preserved.

1. Core Definition

In standard arithmetic, division by zero is undefined. RAS redefines the division operation as follows:

$$\frac{x}{0} = 0_{\langle x\rangle}$$

Here, the result is a special value: a zero that symbolically contains the "reserve" x.

2. Arithmetic Operations on Reserves

Addition

$$0_{\langle a\rangle} + 0_{\langle b\rangle} = 0_{\langle a+b\rangle}$$

Multiplication

Let n be a scalar and $0_{\langle x\rangle}$ a reserve:

$$n \cdot 0_{\langle x\rangle} = 0_{\langle n\cdot x\rangle}$$

Distributive law applies:

$$0_{\langle a\rangle}(b + 0_{\langle c\rangle}) = ab + 0_{\langle ac\rangle}$$

Nested Reserves and Flattening

Reserves can be nested, such as $0_{\langle 0_{\langle x\rangle}\rangle}$, which symbolically preserves the structure. Flattening simplifies:

$$0_{\langle 0_{\langle x\rangle}\rangle} \Rightarrow 0_{\langle x\rangle}$$

3. Evaluation Examples and Results

The following table presents sample computations using RAS and their corresponding outcomes:

Expression	**Result**	**Explanation**
$\frac{5}{0}$	$0_{\langle 5\rangle}$	Division by zero stores 5 in reserve
$0_{\langle 3\rangle} + 0_{\langle 7\rangle}$	$0_{\langle 10\rangle}$	Sum of reserves: $3 + 7$
$0_{\langle 2\rangle} \cdot (4 + 0_{\langle 3\rangle})$	$0_{\langle 14\rangle}$	$2 \cdot 4 + 2 \cdot 3 = 8 + 6 = 14$
$\frac{0_{\langle 5\rangle}}{0}$	$0_{\langle 0_{\langle 5\rangle}\rangle}$	Reserve nested via further division by zero
Flatten $0_{\langle 0_{\langle 5\rangle}\rangle}$	$0_{\langle 5\rangle}$	Simplified reserve form

4. Formal Axiomatic System and Algebraic Properties

4.1 Domain Definition

Let $\mathbb{R}_{\text{RAS}}$ denote the set of real numbers extended with reserved zeros:

$$\mathbb{R}_{\text{RAS}} = \mathbb{R} \cup \left\{0_{\langle r\rangle} \mid r \in \mathbb{R}\right\}$$

where $0_{\langle r\rangle}$ is a symbolic zero containing the reserve r.

4.2 Primitive Operations

Let $a, b \in \mathbb{R}_{\text{RAS}}$:

- **Addition:** $0_{\langle a\rangle} + 0_{\langle b\rangle} = 0_{\langle a+b\rangle}$
- **Scalar Multiplication:** $k \cdot 0_{\langle a\rangle} = 0_{\langle ka\rangle}$ for $k \in \mathbb{R}$
- **Distributive Multiplication:** $0_{\langle a\rangle}(b + 0_{\langle c\rangle}) = ab + 0_{\langle ac\rangle}$
- **Division by Zero Rule:** $\dfrac{x}{0} = 0_{\langle x\rangle}$ for $x \in \mathbb{R}$
- **Reserve Extraction:** $\text{extract}(0_{\langle r\rangle}) = r$

4.3 Axioms of RAS

- **Closure:** $\forall a, b \in \mathbb{R}_{\text{RAS}}$, the operations $a + b$ and ab yield elements in $\mathbb{R}_{\text{RAS}}$.
- **Commutativity:** $a + b = b + a$, $ab = ba$
- **Associativity:** $(a + b) + c = a + (b + c)$, $(ab)c = a(bc)$
- **Distributivity:** $a(b + c) = ab + ac$
- **Additive Identity:** $0_{\langle 0\rangle}$ is the additive identity: $a + 0_{\langle 0\rangle} = a$
- **Multiplicative Identity:** 1 (real number) acts as multiplicative identity: $a \cdot 1 = a$
- **Reserve Nesting:** $\dfrac{0_{\langle x\rangle}}{0} = 0_{\langle 0_{\langle x\rangle}\rangle}$
- **Reserve Flattening:** $0_{\langle 0_{\langle x\rangle}\rangle} \Rightarrow 0_{\langle x\rangle}$

4.5 Formal Proofs of Derived Properties

We now derive and verify key algebraic properties of $\mathbb{R}_{\text{RAS}}$ using the primitive operations defined in Section 4.2 and the axioms in Section 4.3.

Theorem 4.5.1 (Reserve Addition is Commutative). For all $a, b \in \mathbb{R}_{\geq 0}$,

$$0_{\langle a \rangle} + 0_{\langle b \rangle} = 0_{\langle b \rangle} + 0_{\langle a \rangle}$$

Proof.

$$\begin{aligned} 0_{\langle a \rangle} + 0_{\langle b \rangle} &= 0_{\langle a+b \rangle} \quad \text{(by reserve addition definition)} \\ &= 0_{\langle b+a \rangle} \quad \text{(real number addition is commutative)} \\ &= 0_{\langle b \rangle} + 0_{\langle a \rangle} \quad \text{(by reserve addition definition)} \end{aligned}$$

■

Theorem 4.5.2 (Reserve Addition is Associative). For all $a, b, c \in \mathbb{R}_{\geq 0}$,

$$(0_{\langle a \rangle} + 0_{\langle b \rangle}) + 0_{\langle c \rangle} = 0_{\langle a \rangle} + (0_{\langle b \rangle} + 0_{\langle c \rangle})$$

Proof.

$$\begin{aligned} (0_{\langle a \rangle} + 0_{\langle b \rangle}) + 0_{\langle c \rangle} &= 0_{\langle a+b \rangle} + 0_{\langle c \rangle} \\ &= 0_{\langle (a+b)+c \rangle} \\ &= 0_{\langle a+(b+c) \rangle} \quad \text{(real number associativity)} \\ &= 0_{\langle a \rangle} + 0_{\langle b+c \rangle} \\ &= 0_{\langle a \rangle} + (0_{\langle b \rangle} + 0_{\langle c \rangle}) \end{aligned}$$

■

Theorem 4.5.3 (Distributivity of Scalar Multiplication over Reserve Addition). For all $k \in \mathbb{R}$ and $a, b \in \mathbb{R}_{\geq 0}$,

$$k \cdot (0_{\langle a \rangle} + 0_{\langle b \rangle}) = k \cdot 0_{\langle a \rangle} + k \cdot 0_{\langle b \rangle}$$

Proof.

$$\begin{aligned} 0_{\langle a \rangle} + 0_{\langle b \rangle} &= 0_{\langle a+b \rangle} \\ k \cdot (0_{\langle a \rangle} + 0_{\langle b \rangle}) &= k \cdot 0_{\langle a+b \rangle} \\ &= 0_{\langle k(a+b) \rangle} \\ &= 0_{\langle ka+kb \rangle} \\ &= 0_{\langle ka \rangle} + 0_{\langle kb \rangle} \\ &= k \cdot 0_{\langle a \rangle} + k \cdot 0_{\langle b \rangle} \end{aligned}$$

■

Theorem 4.5.4 (Division by Zero Nested Reserves).

$$\frac{0_{\langle x \rangle}}{0} = 0_{\langle 0_{\langle x \rangle} \rangle}$$

Proof. By definition, division by zero yields a reserve containing the numerator, so applying it to $0_{\langle x \rangle}$ yields a reserve containing the reserve $0_{\langle x \rangle}$.

$$\frac{0_{\langle x \rangle}}{0} = 0_{\langle 0_{\langle x \rangle} \rangle}$$

Theorem 4.5.5 (Reserve Flattening).

$$0_{\langle 0_{\langle x\rangle}\rangle} \Rightarrow 0_{\langle x\rangle}$$

Proof. Nested reserves collapse to a single reserve for simplification.

$$0_{\langle 0_{\langle x\rangle}\rangle} \rightarrow 0_{\langle x\rangle}$$

5. Conclusion

The **Reserve Arithmetic System (RAS)** introduces a novel extension to classical arithmetic by redefining division by zero through the concept of *reserved zeros*—symbolic constructs that store the numerator while the result evaluates to zero. This framework preserves many foundational algebraic properties, including commutativity, associativity, and distributivity, while systematically avoiding undefined expressions.

The resulting structure, termed a **reserve semiring**, integrates the arithmetic behavior of reserves with standard real numbers. This opens pathways for consistent and reversible arithmetic operations involving division by zero.

Future work may investigate:

- Extensions of RAS to broader algebraic systems (e.g., rings, fields, or modules),
- Formal treatment of *reserve morphisms* within category theory,
- Embedding RAS into *extended numerical fields*,
- Development of *reserve-aware symbolic computation engines*.

These directions offer promising ground for theoretical enrichment and practical applications in computational mathematics, automated reasoning, and beyond.

6. Implementation: Python Code

To demonstrate the practical implementation of the Reserve Arithmetic System (RAS), the following Python code provides a proof-of-concept for the key operations: division by zero producing a reserved zero, arithmetic operations on reserves, reserve extraction, flattening of nested reserves, and reciprocal handling.

```python
from dataclasses import dataclass
from typing import Union

# Define a type that can be either a real number or a ReservedZero
RAS = Union[float, "ReservedZero"]

@dataclass
class ReservedZero:
    reserve: RAS

    def __repr__(self):
        return f"0<{self.reserve}>"

    def extract(self):
        if isinstance(self.reserve, ReservedZero):
            return self.reserve.extract()
        return self.reserve

    def flatten(self):
        if isinstance(self.reserve, ReservedZero):
            return ReservedZero(self.reserve.flatten().reserve)
        return self

def eR(value: RAS) -> float:
    if isinstance(value, ReservedZero):
        return value.extract()
    return 0.0

def div(x: RAS, y: RAS) -> RAS:
    if isinstance(y, ReservedZero):
        if isinstance(x, ReservedZero):
            return ReservedZero(div(x.reserve, y.reserve))
        elif isinstance(y.reserve, (int, float)) and y.reserve != 0:
            return ReservedZero(x / y.reserve)
        else:
            return ReservedZero(ReservedZero(x))
    elif y == 0:
        return ReservedZero(x)
    return x / y

def add(x: RAS, y: RAS) -> RAS:
    if isinstance(x, ReservedZero) and isinstance(y, ReservedZero):
        return ReservedZero(add(x.reserve, y.reserve))
    elif isinstance(x, ReservedZero):
        return ReservedZero(x.reserve)
    elif isinstance(y, ReservedZero):
        return ReservedZero(y.reserve)
    return x + y

def sub(x: RAS, y: RAS) -> RAS:
    if isinstance(x, ReservedZero) and isinstance(y, ReservedZero):
        return ReservedZero(sub(x.reserve, y.reserve))
    elif isinstance(x, ReservedZero):
        return ReservedZero(x.reserve)
    elif isinstance(y, ReservedZero):
        return ReservedZero(-y.reserve)
    return x - y

def mul(x: RAS, y: RAS) -> RAS:
    if isinstance(x, ReservedZero) and isinstance(y, ReservedZero):
        return ReservedZero(mul(x.reserve, y.reserve))
    elif isinstance(x, ReservedZero):
        return ReservedZero(mul(x.reserve, y))
    elif isinstance(y, ReservedZero):
        return ReservedZero(mul(x, y.reserve))
    return x * y

def reciprocal(x: RAS) -> RAS:
    if isinstance(x, ReservedZero):
        if x.extract() == 0:
            raise ZeroDivisionError("Cannot invert 0<0>")
        return ReservedZero(1 / x.extract())
    else:
        if x == 0:
            raise ZeroDivisionError("Cannot invert 0")
        return 1 / x

def f_R(value: RAS) -> RAS:
    if isinstance(value, ReservedZero):
        return value.flatten()
    return value

# Example usage
if __name__ == "__main__":
    a = div(5, 0)
    b = ReservedZero(3)
    c = ReservedZero(7)
    d = add(b, c)
    e = div(a, 0)
    f = f_R(e)

    print("a =", a)
    print("d =", d)
    print("e =", e)
    print("f =", f)
    print("eR(d) =", eR(d))
    print("reciprocal(b) =", reciprocal(b))

    x = mul(2, b)
    y = sub(d, b)
    print("x =", x)
    print("y =", y)
```

This code serves as a foundational demonstration of RAS operations, suitable for further exploration, refinement, and computational experimentation.

Disclosure

The author acknowledge the use of OpenAI's ChatGPT language model to assist in the preparation and writing of this paper. While ChatGPT provided support in drafting and refining text, all final content and interpretations are the sole responsibility of the author.

Companion Code and Repository

An interactive implementation of the Reserve Arithmetic System (RAS) is available at:
`https://replit.com/@yourusername/ReserveArithmeticSystem`