Division and Modulo from Recursive Normalization

Thiago Henrique Mata
Independent Researcher
thiago.henrique.mata@gmail.com
https://thiagomata.com/
https://github.com/thiagomata

Abstract

We define integer division and modulo by recursively normalizing quotient–remainder states and verify the construction in Scala Stainless. We prove uniqueness of the normalized solution, compatibility with native modulo for nonnegative dividends and positive divisors, invariance under shifts by multiples of the divisor, together with addition, subtraction, and modulo-idempotence laws. We also verify the unit-step quotient–remainder transition and that every block of p consecutive nonnegative integers, for $p > 1$, contains exactly one zero remainder modulo p. Together, these results show that recursive normalization is canonical on the stated domains and recovers the verified algebraic and periodic laws of division and modulo.

1 Introduction

Integer division and modulo operations are central tools in discrete mathematics, number theory, and algorithms. While their properties are well known, rigorous formalization and verification, particularly via recursive definitions, offer an interesting alternative to the traditional axiomatic model.

This article takes the recursive route. Instead of assuming native division and modulo, it defines a state $DivMod(a, b, q, r)$ where $a = bq + r$, then proves that normalizing the pair (q, r) preserves the represented dividend and reaches the canonical remainder interval. The familiar operations div and mod are then projections from that normalized state.

The mathematical statements below are backed by Scala source verified with Stainless [1]. The article keeps the proof discussion centered on the properties; source links point to the maintained verification code.

This article establishes:

- Foundational identities: trivial case, self-identity, division by one, and agreement with the native modulo operator (Sections 6.1–6.4).
- Linear shift laws under single-step and multiple-step divisor addition (Sections 6.5–6.6).
- Uniqueness and idempotence of the normalized remainder (Sections 6.7–6.8).
- Distributivity of modulo and division over addition and subtraction (Sections 6.9–6.10).
- Divisible-base shift invariance and symmetric remainder pairs (Sections 6.11–6.12).
- The unit-step increment law and zero-density over consecutive integers (Sections 6.13–6.14).

2 Limitations

The implementation presented in this article is limited to the division and modulo operations for integers. Its goal is to make available a set of lemmas and proofs that can be verified and used as a base to prove other properties related to the division and modulo operations. Therefore, the implementation is optimized to correctness and not to performance.

The use of `BigInt` in the implementation focused on unbounded integers, without the need to worry about overflow or underflow issues. But they are still constrained by the memory available in the system. Similarly, some lemmas and proofs use the recursive definition of the division and modulo operations, which could trigger a stack overflow for large numbers. Those issues do not invalidate the mathematical properties proved in this article, which are the main focus of this article.

3 Traditional Definition

Given integers dividend and divisor where divisor $\neq 0$, the division algorithm determines integers quotient and remainder such that:

$$\begin{aligned}
&\forall\, \text{dividend}, \text{divisor} \in \mathbb{Z},\ \text{divisor} \neq 0,\ \exists!\, \text{quotient}, \text{remainder}: \\
&\qquad\qquad \text{dividend} = \text{divisor} \cdot \text{quotient} + \text{remainder} \\
&\qquad\qquad 0 \leq \text{remainder} < |\text{divisor}| \\
&\qquad \text{dividend}\,\text{div}\,\text{divisor} := \text{quotient} \\
&\qquad \text{dividend}\,\text{mod}\,\text{divisor} := \text{remainder}
\end{aligned}$$

The first two lines state the division relation and the canonical remainder range. The final two lines introduce the operation notation: division returns the quotient, and modulo returns the remainder.

4 Recursive Definition

We introduce a recursive definition of division and modulo because the proof can be built from one invariant: shifting one unit of b between quotient and remainder preserves the represented dividend. In Section 5, that invariant connects the recursive normal form back to the traditional division equation from Section 3.

From here on, a is the dividend, b is the divisor, q is the candidate quotient, and r is the candidate remainder. The shorter names keep the recursive equations readable while preserving the same roles as the traditional definition. We reserve mod for the modulo operation itself. Throughout, $\mathbb{N} = \{0, 1, 2, \ldots\}$.

We define $\text{DivMod}(a, b, q, r)$ such that:

$$\forall a, b, q, r \in \mathbb{Z} : b \neq 0,\ a = bq + r$$

The solved DivMod states are those where the remainder r satisfies:

$$\begin{cases} 0 \leq r < b & \text{if } b > 0, \\ 0 \leq r < -b & \text{if } b < 0. \end{cases}$$

$$\mathrm{DivMod.solve}(a,b,q,r) := \begin{cases} \mathrm{DivMod}(a,b,q,r) & \text{if } 0 \le r < |b|, \\ \mathrm{DivMod.solve}(a,b,q+\mathrm{sign}(b),r-|b|) & \text{if } r \ge |b|, \\ \mathrm{DivMod.solve}(a,b,q-\mathrm{sign}(b),r+|b|) & \text{if } r < 0. \end{cases}$$

The recursive definition is implemented in `DivMod.scala`.

5 DivMod Solution Invariance Under Linear Shift

Moving one copy of the divisor between quotient and remainder preserves the represented dividend; normalization therefore reaches the same final state.

$$\begin{aligned} \forall a,b,q,r \in \mathbb{Z},\ b \neq 0,\ a &= bq + r \\ a &= b(q+1) + (r-b) \\ a &= b(q-1) + (r+b) \\ \mathrm{DivMod}(a,b,q+1,r-b).\mathrm{solve} &= \mathrm{DivMod}(a,b,q,r).\mathrm{solve} \\ \mathrm{DivMod}(a,b,q-1,r+b).\mathrm{solve} &= \mathrm{DivMod}(a,b,q,r).\mathrm{solve} \end{aligned}$$

This invariant is verified for the positive shift and negative shift.

5.1 Creating the Division and Modulo Operations

Using the normalized `DivMod` value, `Calc.scala` defines div and mod as the quotient and remainder projections of the solved state. Starting from $\mathrm{DivMod}(a,b,0,a)$, let:

$$\begin{aligned} S &:= \mathrm{DivMod}(a,b,0,a).\mathrm{solve} \\ q &:= S.\mathrm{div} \\ r &:= S.\mathrm{mod} \\ \mathrm{div}(a,b) &:= q \\ \mathrm{mod}(a,b) &:= r \end{aligned}$$

So $\mathrm{div}(a,b)$ names the normalized quotient, while $\mathrm{mod}(a,b)$ names the normalized remainder. In the source code these are the `div` and `mod` fields of the solved `DivMod`; in the article notation, q and r keep the quotient and remainder roles separate from the operation names.

The article uses both functional and infix notation for the same operations: $\mathrm{div}(x,y)$ and $x \operatorname{div} y$ are equivalent, as are $\mathrm{mod}(x,y)$ and $x \bmod y$. Functional notation is useful when the operation is nested inside another expression; infix notation keeps simple algebraic identities closer to the traditional presentation.

6 Some Important Properties of Modulo and Division

This chapter develops the concrete identities that follow from the definitions and the linear-shift invariant of Section 5. It establishes:

- The base cases where normalization is immediate: a small dividend, self-division, division by one, and agreement with the native modulo operator (Subsections 6.1–6.4)

- How a single or repeated shift of the dividend by the divisor moves the quotient without disturbing the remainder (Subsections 6.5–6.6)
- The uniqueness of the normalized remainder and its idempotence under repeated reduction (Subsections 6.7–6.8)
- Distributivity of modulo and division over addition and subtraction (Subsections 6.9–6.10)
- Shift invariance when the dividend is already divisible by the base, and the symmetry of remainder pairs around that base (Subsections 6.11–6.12)
- The unit-step increment law and the density of zero remainders across consecutive integers (Subsections 6.13–6.14)

6.1 Trivial Case

If the dividend is smaller than a positive divisor, the candidate state $\mathrm{DivMod}(a, b, 0, a)$ is already final. No subtraction of b is needed, so the quotient is zero and the remainder is the original dividend.

$$\begin{aligned} &\forall\, a, b \in \mathbb{N} : b \neq 0 \\ &a < b \implies a \bmod b = a \\ &a < b \implies a \operatorname{div} b = 0 \end{aligned}$$

This property is verified in `ModSmallDividend`.

6.2 Identity

The modulo of every number by itself is zero and the division of every number by itself is one.

$$\begin{aligned} &\forall\, n \in \mathbb{N} : n \neq 0 \\ &n \bmod n = 0 \\ &n \operatorname{div} n = 1 \end{aligned}$$

The candidate state normalizes in one shift to the final state with quotient 1 and remainder 0:

$$\begin{aligned} n = n \cdot 0 + n = n \cdot 1 + 0 \\ 0 \leq 0 < |n| \\ \mathrm{DivMod}(n, n, 0, n).\,\mathrm{solve} = \mathrm{DivMod}(n, n, 1, 0). \quad \blacksquare\ \text{[Q.E.D.]} \end{aligned}$$

This property is verified in `ModIdentity::modIdentity`. A longer source proof showing the normalization path is available in `ModIdentity::longProof`.

6.3 Modulo and Division by One

Modulo by one always returns zero and division by one always returns the dividend.

$$\begin{aligned} &\forall\, n \in \mathbb{N} : \\ &n \bmod 1 = 0 \\ &n \operatorname{div} 1 = n \end{aligned}$$

Both identities follow directly from the already canonical decomposition $n = 1 \cdot n + 0$; no induction or later unit-step result is needed.

$$n = 1 \cdot n + 0, \qquad 0 \le 0 < 1$$
$$\mathrm{mod}(n, 1) = 0, \qquad \mathrm{div}(n, 1) = n. \quad \blacksquare \text{ [Q.E.D.]}$$

These properties are verified in `ModOne::modOneIsZero` and `ModOne::divOneIsN`.

6.4 Compatibility with Native Modulo

For non-negative dividends and positive divisors, the recursively normalized modulo agrees with `BigInt`'s native `%` operator.

$$\forall\, a, b \in \mathbb{Z} : a \ge 0,\ b > 0$$
$$a \bmod b = a \,\%\, b$$

This is not a new mathematical fact but a bridge lemma: it confirms that the native `%` operator and the recursively defined mod agree on their shared domain of non-negative dividends and positive divisors, so results derived from one match results derived from the other.

This property is verified in `ModNativeCompatibility::percentEqualsCalcMod`.

6.5 Quotient Invariance Under Linear Shift

Adding or subtracting the divisor from the dividend changes the quotient by one but leaves the remainder unchanged.

$$\forall a, b, q, r \in \mathbb{Z} : b \neq 0,\ a = bq + r$$
$$\mathrm{mod}(a + b, b) = \mathrm{mod}(a, b)$$
$$\mathrm{div}(a + b, b) = \mathrm{div}(a, b) + 1$$
$$\mathrm{mod}(a - b, b) = \mathrm{mod}(a, b)$$
$$\mathrm{div}(a - b, b) = \mathrm{div}(a, b) - 1$$

Let $q = \mathrm{div}(a, b)$ and $r = \mathrm{mod}(a, b)$. The normalized state satisfies $a = bq + r$, with r already in the canonical remainder interval. Shifting the dividend by one divisor gives two equally canonical states:

$$a + b = bq + r + b = b(q + 1) + r$$
$$a - b = bq + r - b = b(q - 1) + r$$
$$\mathrm{DivMod}(a + b, b, q + 1, r).\mathrm{solve} = \mathrm{DivMod}(a + b, b, q + 1, r)$$
$$\mathrm{DivMod}(a - b, b, q - 1, r).\mathrm{solve} = \mathrm{DivMod}(a - b, b, q - 1, r). \quad \blacksquare \text{ [Q.E.D.]}$$

This property is verified for the positive case and negative case.

6.6 Quotient Invariance Under Linear Shift by Multiplier

Adding a multiple of the divisor changes the quotient by that multiplier but leaves the remainder unchanged.

As a direct consequence of the one-step shift laws, we can also prove that:

$$\begin{aligned}
&\forall a, b, q, r, m \in \mathbb{Z} : b \neq 0,\ a = bq + r \\
&\operatorname{mod}(a + m \cdot b, b) = \operatorname{mod}(a, b) \\
&\operatorname{div}(a + m \cdot b, b) = \operatorname{div}(a, b) + m \\
&\operatorname{mod}(a - m \cdot b, b) = \operatorname{mod}(a, b) \\
&\operatorname{div}(a - m \cdot b, b) = \operatorname{div}(a, b) - m
\end{aligned}$$

For $m \geq 0$, induction applies the positive one-step law to $a + mb$: the remainder is unchanged at each step and the quotient gains one. Thus the result holds at $m = 0$ and is preserved from m to $m + 1$. The subtraction identities follow by the same induction using the negative one-step law. If $m < 0$, write $m = -t$ with $t > 0$ and exchange the addition and subtraction identities just obtained.

$$\begin{aligned}
&\operatorname{mod}(a + (m+1)b, b) = \operatorname{mod}((a + mb) + b, b) = \operatorname{mod}(a, b) \\
&\operatorname{div}(a + (m+1)b, b) = \operatorname{div}(a + mb, b) + 1 = \operatorname{div}(a, b) + m + 1. \quad \blacksquare \text{ [Q.E.D.]}
\end{aligned}$$

This property is verified for the positive case and negative case.

6.7 Unique Remainder

There is only one single remainder value for every a, b pair with $b > 0$.

$$\begin{aligned}
&\forall\, a, b \in \mathbb{N} : b > 0 \\
&\exists!\, r \in \mathbb{N} : 0 \leq r < b \ \wedge\ a = \left\lfloor \frac{a}{b} \right\rfloor \cdot b + r
\end{aligned}$$

in other words, two DivMod instances with the same dividend a and divisor b will have the same solution.

$$\begin{aligned}
\forall a, b, q_x, r_x, q_y, r_y \in \mathbb{N},& \\
\text{where } b \neq 0,& \\
a = bq_x + r_x \text{ and}& \\
a = bq_y + r_y \text{ then}& \\
\operatorname{DivMod}(a, b, q_x, r_x).\operatorname{solve} &= \operatorname{DivMod}(a, b, q_y, r_y).\operatorname{solve}
\end{aligned}$$

For every a, b pair, with any candidate quotients and remainders (q_x, r_x) and (q_y, r_y) representing the same dividend, normalization reaches the same solution. This property is verified in `ModIdempotence::modUnique`.

6.8 Modulo Idempotence

Taking the modulo of a number twice gives the same result as taking it once.

$$\begin{aligned}
&\forall\, a, b \in \mathbb{Z} : b \neq 0 \\
&a \bmod b = (a \bmod b) \bmod b
\end{aligned}$$

This property is verified in `ModIdempotence::modIdempotence`.

6.9 Distributivity over Addition

The modulo operation distributes over addition, meaning that the remainder of a sum equals the remainder of the sum of remainders. This allows us to break down complex modulo operations into simpler components.

$$\begin{aligned}
&\forall\, a, b, c \in \mathbb{Z} : b \neq 0 \\
&\quad (a + c) \bmod b = (a \bmod b + c \bmod b) \bmod b \\
&\quad (a + c) \operatorname{div} b = a \operatorname{div} b + c \operatorname{div} b + (a \bmod b + c \bmod b) \operatorname{div} b \\
&\quad (a + c) \bmod b = (a \bmod b) + (c \bmod b) - b \cdot (((a \bmod b) + (c \bmod b)) \operatorname{div} b)
\end{aligned}$$

Let $q_a = \operatorname{div}(a, b)$, $r_a = \operatorname{mod}(a, b)$, $q_c = \operatorname{div}(c, b)$, and $r_c = \operatorname{mod}(c, b)$. Thus $a = bq_a + r_a$ and $c = bq_c + r_c$. Normalizing the sum of the two remainders gives $r_a + r_c = bs + t$, where $s = \operatorname{div}(r_a + r_c, b)$ and $t = \operatorname{mod}(r_a + r_c, b)$. Substitution gives the quotient and remainder of $a + c$:

$$\begin{aligned}
a + c &= b(q_a + q_c) + (r_a + r_c) \\
&= b(q_a + q_c + s) + t \\
t &= r_a + r_c - b \cdot \operatorname{div}(r_a + r_c, b). \quad \blacksquare \text{ [Q.E.D.]}
\end{aligned}$$

This property is verified in `ModOperations::modAdd`. The third identity, isolating the multiple of b subtracted out, is proved directly in `ModIdempotence.scala#modModPlus`.

6.10 Distribution over Subtraction

Similar to addition, the modulo operation distributes over subtraction. The remainder of a difference equals the remainder of the difference of remainders, with appropriate handling of negative values.

$$\begin{aligned}
&\forall\, a, b, c \in \mathbb{Z} : b \neq 0 \\
&\quad (a - c) \bmod b = (a \bmod b - c \bmod b) \bmod b \\
&\quad (a - c) \operatorname{div} b = a \operatorname{div} b - c \operatorname{div} b + (a \bmod b - c \bmod b) \operatorname{div} b \\
&\quad (a - c) \bmod b = (a \bmod b) - (c \bmod b) - b \cdot (((a \bmod b) - (c \bmod b)) \operatorname{div} b)
\end{aligned}$$

Let $q_a = \operatorname{div}(a, b)$, $r_a = \operatorname{mod}(a, b)$, $q_c = \operatorname{div}(c, b)$, and $r_c = \operatorname{mod}(c, b)$. Thus $a = bq_a + r_a$ and $c = bq_c + r_c$. Normalize the difference $r_a - r_c = bs + t$. Then the canonical decomposition of $a - c$ is obtained without any assumption that $r_a - r_c$ is nonnegative:

$$\begin{aligned}
a - c &= b(q_a - q_c) + (r_a - r_c) \\
&= b(q_a - q_c + s) + t \\
t &= r_a - r_c - b \cdot \operatorname{div}(r_a - r_c, b). \quad \blacksquare \text{ [Q.E.D.]}
\end{aligned}$$

This property is verified in `ModOperations::modLess`. The third identity, isolating the multiple of b subtracted out, is proved directly in `ModIdempotence.scala#modModMinus`.

6.11 Modular Shift Invariance under Divisible Base

When a number is a multiple of the divisor (modulo equals zero), adding any value does not change the modulo of that value. This property simplifies calculations when one operand is already divisible by the base. It holds for any integer c, including negative values.

$$\forall\, a, b, c \in \mathbb{Z} : b \neq 0$$
$$a \bmod b = 0 \implies (a + c) \bmod b = c \bmod b$$

The addition law from Subsection 6.9 reduces the left-hand side to the modulo of the two remainders. The hypothesis removes the first one, and modulo idempotence removes the remaining repetition:

$$\begin{aligned} \operatorname{mod}(a + c, b) &= \operatorname{mod}(\operatorname{mod}(a, b) + \operatorname{mod}(c, b), b) \\ &= \operatorname{mod}(\operatorname{mod}(c, b), b) \\ &= \operatorname{mod}(c, b). \quad \blacksquare \text{ [Q.E.D.]} \end{aligned}$$

This property is verified in `ModOperations::modZeroPlusC`.

Substituting $-c$ for c gives the subtraction corollary directly, since c is unrestricted:

$$\forall\, a, b, c \in \mathbb{Z} : b \neq 0$$
$$a \bmod b = 0 \implies (a - c) \bmod b = (-c) \bmod b$$

This corollary is verified by the same lemma, `ModOperations::modZeroPlusC`, called with $-c$ in place of c.

6.12 Symmetrical Modulo Pairs

The modulo of a value and the modulo of its complement relative to the base sum to the base.

$$b > 0, \quad 0 < k < b$$
$$k \bmod b + (b - k) \bmod b = b$$

Since both k and $b - k$ already lie inside the canonical remainder interval, their remainders are themselves:

$$\begin{aligned} 0 < k < b &\implies \operatorname{mod}(k, b) = k \\ 0 < b - k < b &\implies \operatorname{mod}(b - k, b) = b - k \\ \operatorname{mod}(k, b) + \operatorname{mod}(b - k, b) = k + (b - k) = b. &\quad \blacksquare \text{ [Q.E.D.]} \end{aligned}$$

This property is verified in `ModSum::sumSymmetricalMods`. The source excerpt is included in Appendix A.2.

6.13 Unit-Step Modulo-Division Increment Law

When incrementing a number by one, the modulo cycles from 0 to b-1 and resets, while the division increments only when the modulo reaches its maximum value. This captures the "carry" behavior of division when counting.

$$\begin{aligned} &\forall\, a, b \in \mathbb{N} : b \neq 0 \\ &a \bmod b = b - 1 \implies (a + 1) \bmod b = 0 \\ &a \bmod b \neq b - 1 \implies (a + 1) \bmod b = (a \bmod b) + 1 \\ &a \bmod b = b - 1 \implies (a + 1) \operatorname{div} b = (a \operatorname{div} b) + 1 \\ &a \bmod b \neq b - 1 \implies (a + 1) \operatorname{div} b = a \operatorname{div} b \end{aligned}$$

Let $q = \mathrm{div}(a, b)$ and $r = \mathrm{mod}(a, b)$, so $a = bq + r$ with $0 \le r < b$. If $r = b - 1$, adding one produces the canonical state $(q + 1, 0)$. Otherwise $r < b - 1$, so $(q, r + 1)$ is already canonical. This also covers $b = 1$: only the first case can occur.

$$r = b - 1 \implies a + 1 = bq + (b - 1) + 1 = b(q + 1) + 0$$
$$r < b - 1 \implies a + 1 = bq + (r + 1), \quad 0 \le r + 1 < b. \quad \blacksquare \text{ [Q.E.D.]}$$

This property is verified in `ModOperations::addOne`.

6.14 Consecutive Integers: Zero Density

In any block of p consecutive integers, exactly one is divisible by p. This is the basic counting form of modulo periodicity: as we advance through consecutive integers, the remainder modulo p visits zero once per complete period.

At most one zero per block. If $\mathrm{mod}(a, p) = 0$ and $0 < d < p$, then $\mathrm{mod}(a + d, p) \neq 0$. Within any block of size p starting from a multiple, no later offset inside the same block can also be divisible by p.

$$\mathrm{mod}(a,\ p) = 0 \ \wedge\ 0 < d < p \implies \mathrm{mod}(a + d,\ p) \neq 0 \quad \text{[nonzeroAfterZero]}$$

At least one zero per block. For any starting value n and modulus $p > 1$, there exists a $k \in [0, p)$ such that $\mathrm{mod}(n + k, p) = 0$. The witness is $k = 0$ when n is already divisible by p, and $k = p - \mathrm{mod}(n, p)$ otherwise.

$$\forall n \ge 0,\ p > 1,\ \exists\, k \in [0, p) : \mathrm{mod}(n + k,\ p) = 0 \quad \text{[existsZero]}$$

Exactly one zero per block. Existence gives a zero offset, while uniqueness says two zero offsets in the same block must be equal. Together, among p consecutive integers starting from n, exactly one is divisible by p.

$$\forall n \ge 0,\ p > 1,\ \exists!\, k \in [0, p) : \mathrm{mod}(n + k,\ p) = 0 \quad \text{[existsZero + atMostOneZero]}$$

More explicitly, let k be the offset supplied by existence. If i and j are two offsets in $[0, p)$ with zero remainder, the at-most-one result applied to i and j yields $i = j$; hence the existing k is unique.

$$\exists\, k \in [0, p) :\ \mathrm{mod}(n + k, p) = 0$$
$$\mathrm{mod}(n + i, p) = \mathrm{mod}(n + j, p) = 0, \quad i, j \in [0, p) \implies i = j$$
$$\therefore\ \exists!\, k \in [0, p) :\ \mathrm{mod}(n + k, p) = 0. \quad \blacksquare \text{ [Q.E.D.]}$$

These properties are verified in `ConsecutiveIntegers::nonzeroAfterZero`, `ConsecutiveIntegers::existsZero`, `ConsecutiveIntegers::exactlyOneZeroInConsecutive`, and `ConsecutiveIntegers::atMostOneZero`. The compact source shape is included in Appendix A.3. The maintained source is `ConsecutiveIntegers.scala`. The file also contains multi-factor density helpers, such as `twoFactorsDensity` and `densityForFactorList`; their statements are not established in this article, and the multiplicative density extension they aim at is discussed as an open direction in the Future Work section.

7 Conclusion

This article established a formally verified property theory for division and modulo generated by recursive quotient–remainder normalization. Within the stated domains, the following results show that the normalization is canonical and satisfies the expected algebraic and periodic laws:

$$\begin{aligned}
&\forall\, a, b \in \mathbb{N} : b \neq 0 \\
&a < b \implies a \bmod b = a \quad \text{[Trivial Case]} \\
&a < b \implies a \operatorname{div} b = 0 \quad \text{[Trivial Case]}
\end{aligned}$$

$$\begin{aligned}
&\forall\, n \in \mathbb{N} : n \neq 0 \\
&n \bmod n = 0 \quad \text{[Identity]} \\
&n \operatorname{div} n = 1 \quad \text{[Identity]}
\end{aligned}$$

$$\begin{aligned}
&\forall\, n \in \mathbb{N} : \\
&n \bmod 1 = 0 \quad \text{[Division by One]} \\
&n \operatorname{div} 1 = n \quad \text{[Division by One]}
\end{aligned}$$

$$\begin{aligned}
&\forall\, a, b \in \mathbb{Z} : a \geq 0,\ b > 0 \\
&a \bmod b = a \,\%\, b \qquad \text{[Native Modulo Compatibility]}
\end{aligned}$$

$$\begin{aligned}
&\forall a, b, q, r \in \mathbb{Z} : b \neq 0,\ a = bq + r \\
&\operatorname{mod}(a + b, b) = \operatorname{mod}(a, b) \qquad \text{[Linear Shift]} \\
&\operatorname{div}(a + b, b) = \operatorname{div}(a, b) + 1 \qquad \text{[Linear Shift]} \\
&\operatorname{mod}(a - b, b) = \operatorname{mod}(a, b) \qquad \text{[Linear Shift]} \\
&\operatorname{div}(a - b, b) = \operatorname{div}(a, b) - 1 \qquad \text{[Linear Shift]}
\end{aligned}$$

$$\begin{aligned}
&\forall a, b, q, r, m \in \mathbb{Z} : b \neq 0,\ a = bq + r \\
&\operatorname{mod}(a + m \cdot b, b) = \operatorname{mod}(a, b) \qquad \text{[Linear Shift by Multiplier]} \\
&\operatorname{div}(a + m \cdot b, b) = \operatorname{div}(a, b) + m \qquad \text{[Linear Shift by Multiplier]} \\
&\operatorname{mod}(a - m \cdot b, b) = \operatorname{mod}(a, b) \qquad \text{[Linear Shift by Multiplier]} \\
&\operatorname{div}(a - m \cdot b, b) = \operatorname{div}(a, b) - m \qquad \text{[Linear Shift by Multiplier]}
\end{aligned}$$

$$\begin{aligned}
&\forall\, a, b, q_x, r_x, q_y, r_y \in \mathbb{N},\ b \neq 0,\ a = bq_x + r_x = bq_y + r_y \\
&\operatorname{DivMod}(a, b, q_x, r_x).\operatorname{solve} = \operatorname{DivMod}(a, b, q_y, r_y).\operatorname{solve} \qquad \text{[Unique Remainder]}
\end{aligned}$$

$$\begin{aligned}
&\forall\, a, b \in \mathbb{Z} : b \neq 0 \\
&a \bmod b = (a \bmod b) \bmod b \quad \text{[Modulo Idempotence]}
\end{aligned}$$

$$\begin{aligned}
&\forall\, a, b, c \in \mathbb{Z} : b \neq 0 \\
&(a + c) \bmod b = (a \bmod b + c \bmod b) \bmod b \qquad \text{[Distributivity, Addition]} \\
&(a + c) \operatorname{div} b = a \operatorname{div} b + c \operatorname{div} b + (a \bmod b + c \bmod b) \operatorname{div} b \quad \text{[Distributivity, Addition]} \\
&(a + c) \bmod b = (a \bmod b) + (c \bmod b) - b \cdot (((a \bmod b) \\
&\qquad + (c \bmod b)) \operatorname{div} b) \qquad \text{[Distributivity, Addition]}
\end{aligned}$$

$$\forall a, b, c \in \mathbb{Z} : b \neq 0$$
$$(a - c) \bmod b = (a \bmod b - c \bmod b) \bmod b \quad \text{[Distributivity, Subtraction]}$$
$$(a - c) \operatorname{div} b = a \operatorname{div} b - c \operatorname{div} b + (a \bmod b - c \bmod b) \operatorname{div} b \quad \text{[Distributivity, Subtraction]}$$
$$(a - c) \bmod b = (a \bmod b) - (c \bmod b) - b \cdot (((a \bmod b) - (c \bmod b)) \operatorname{div} b) \quad \text{[Distributivity, Subtraction]}$$

$$\forall a, b, c \in \mathbb{Z} : b \neq 0$$
$$a \bmod b = 0 \implies (a + c) \bmod b = c \bmod b \quad \text{[Divisible-Base Shift Invariance]}$$

$$b > 0, \quad 0 < k < b$$
$$k \bmod b + (b - k) \bmod b = b \quad \text{[Symmetrical Modulo Pairs]}$$

$$\forall a, b \in \mathbb{N} : b \neq 0$$
$$a \bmod b = b - 1 \implies (a + 1) \bmod b = 0 \quad \text{[Unit-Step Increment]}$$
$$a \bmod b \neq b - 1 \implies (a + 1) \bmod b = (a \bmod b) + 1 \quad \text{[Unit-Step Increment]}$$
$$a \bmod b = b - 1 \implies (a + 1) \operatorname{div} b = (a \operatorname{div} b) + 1 \quad \text{[Unit-Step Increment]}$$
$$a \bmod b \neq b - 1 \implies (a + 1) \operatorname{div} b = a \operatorname{div} b \quad \text{[Unit-Step Increment]}$$

$$\forall n, p \in \mathbb{N} : p > 1$$
$$\exists! \, k \in [0, p) : \operatorname{mod}(n + k, \ p) = 0 \quad \text{[Exactly One Zero per Block]}$$

Those formally verified properties are collected in `Summary.scala` and supported by the individual proof modules linked above. The recursive formulation makes the proof structure transparent: normalize (q, r) without changing $a = bq + r$, extract quotient and remainder from the final state, then derive the algebraic laws from that normal form.

Taken together, the canonical solution theorem, shift laws, arithmetic identities, unit-step transition, and one-zero-per-period result characterize both the normalized quotient–remainder state and its periodic behavior on consecutive integers.

8 Future Work

The recursive normalization argument used throughout this article generalizes beyond the integers: the same shift-and-check invariant applies to any Euclidean domain equipped with a well-founded remainder measure, suggesting a more general recursive division theorem. The zero-density results of Section 6.14 also invite a natural multiplicative extension: when a block of consecutive integers is filtered by several pairwise coprime moduli, the resulting density is expected to be the product of the individual single-modulus densities, in the spirit of the Chinese Remainder Theorem [2]. Formalizing that multiplicative extension, and connecting the recursive DivMod state to congruence-class arithmetic more broadly, are natural next steps building on the identities established here.

References

[1] Jad Hamza, Nicolas Voirol, and Viktor Kuncak. System FR: Formalized Foundations for the Stainless Verifier. In *Object-Oriented Programming, Systems, Languages & Applications (OOPSLA)*, 2019. https://lara.epfl.ch/~kuncak/papers/HamzaETAL19SystemFR.pdf.

[2] G. H. Hardy and E. M. Wright. *An Introduction to the Theory of Numbers*. Clarendon Press, Oxford, fifth edition, 1979. See Section 5.4 for the Chinese Remainder Theorem.

A Appendix

A.1 Identity Property Excerpt

Source: ModIdentity.scala.

```
def modIdentity(a: BigInt): Boolean = {
  require(a != 0)
  Calc.mod(a, a) == 0 && Calc.div(a, a) == 1
}.holds
```

A.2 Symmetrical Modulo Pairs Excerpt

Source: ModSum.scala.

```
def sumSymmetricalMods(b: BigInt, step: BigInt): Boolean = {
  require(b > 0)
  require(step > 0)
  require(step < b)
  assert(Calc.mod(step, b) == step)
  assert(Calc.mod(b - step, b) == b - step)
  assert(Calc.mod(step, b) + Calc.mod(b - step, b) == step + b - step)
  Calc.mod(step, b) + Calc.mod(b - step, b) == b
}.holds
```

A.3 Consecutive Zero Density Excerpt

Source: ConsecutiveIntegers.scala.

```
def nonzeroAfterZero(a: BigInt, p: BigInt, d: BigInt): Boolean = {
  require(p > 1)
  require(a >= 0)
  require(d > 0)
  require(d < p)
  require(Calc.mod(a, p) == 0)

  ModOperations.modAdd(a, p, d)
  ModIdempotence.modIdempotence(d, p)
  ModSmallDividend.modSmallDividend(d, p)

  Calc.mod(a + d, p) != 0
}.holds

def existsZero(n: BigInt, p: BigInt): Boolean = {
  require(p > 1)
  require(n >= 0)

  val r = Calc.mod(n, p)

  if (r == 0) {
```

```
    Calc.mod(n, p) == 0
  } else {
    val k = p - r
    ModOperations.modAdd(n, p, k)
    ModSmallDividend.modSmallDividend(k, p)
    Calc.mod(n + k, p) == 0
  }
}.holds

def atMostOneZero(n: BigInt, p: BigInt, i: BigInt, j: BigInt): Boolean = {
  require(p > 1)
  require(n >= 0)
  require(i >= 0 && i < p)
  require(j >= 0 && j < p)
  require(Calc.mod(n + i, p) == 0)
  require(Calc.mod(n + j, p) == 0)

  val smaller = if (i <= j) i else j
  val larger  = if (i <= j) j else i
  val d       = larger - smaller
  assert(d >= 0 && d < p)

  if (d > 0) {
    nonzeroAfterZero(n + smaller, p, d)
  }

  i == j
}.holds
```

A.4 Verification Log

All formally verified results in this article were checked with Stainless 0.9.8.8, compiling with its standard Scala 3.3.3 front end, by running `just verify-ch 2` on the exact sources tagged `modulo-article-v1.0.0`. The run reported 1,374 valid, 0 invalid, and 0 unknown verification conditions. The pinned verification log is `logs/verify-ch-2-v1-chapter2-_.log`. A containerized reproduction of the same chapter-scoped verification, using the environment definition committed with the sources and a cold verification cache (`just verify-ch 2 -docker`), reports identical totals; its pinned log is `logs/verify-ch-2-v1-chapter2-_-docker.log`.