2026 SWIM PRESENTATION

When the Machine and Paper Disagree: Who is Right?

Teaching a computer to count what is missing

Jonathan Liu · **Jason Yang** · **Alan Yao**

July 2026

Roadmap

1. **The mathematics.** What is a numerical semigroup, and how can it model real-life situations?
2. **The tool.** What is a proof assistant, and what does it mean for a computer to *check* a proof?
3. **What we built.** An algorithm, a proof that it works, and the results of our audit using the algorithm.
4. **Bonus.** The proof of a related theorem in semidomains that we plan to establish in Lean in the future.

PART 1

The Mathematics

Ronald McDonald's Nugget-Flavored Dilemma

In 1983, Chicken McNuggets came only in boxes of 6, 9, and 20. What could you actually order?

20	**yes** – one box
26	**yes** – $6 + 20$
29	**yes** – $9 + 20$
71	**yes** – $6 + 5 \cdot 9 + 20$
7	**no**
8	**no**
13	**no**

A ferociously hungry customer, ordering any combination of 6-, 9-, and 20-piece boxes, can hit almost any total — but not every one.

The famous question. What is the *largest* number of nuggets you can never exactly order from boxes of 6, 9, and 20?

The answer is **43**. We'll build the machinery to prove that — and get a computer to check it — throughout this presentation.

Formal Definitions

A **numerical semigroup** is a set S of non-negative integers such that

- $0 \in S$;
- if $a \in S$ and $b \in S$ then $a + b \in S$ (closed under addition);
- only *finitely many* non-negative integers are missing from S.

$\iff$ S can be **generated by a finite set of positive integers whose** gcd **is** 1**.**

The set of orderable totals from addends of $6, 9, 20$ is written

$$\langle 6, 9, 20 \rangle$$

and read *the semigroup generated by* 6, 9 *and* 20: every sum $6a + 9b + 20c$ with $a, b, c \geq 0$.

The next natural step is to define and study certain characteristics of these numerical semigroups.

A fun wrinkle: factorization isn't unique

The same total can come from different combinations.

In the semigroup $\langle 2, 3 \rangle$, the number 6 can be built two different ways:

$$\underbrace{2+2+2}_{\text{length } 3} \quad \text{or} \quad \underbrace{3+3}_{\text{length } 2}.$$

In this semigroup, then, the element 6 has factorizations of different lengths. Determining different types of non-unique factorization gets is its own field, called **factorization theory**.

We won't need factorization theory to find the Frobenius number, but it's a good reminder that a numerical semigroup is more than what is missing from the set. There is a wide range of deep theory about numerical semigroups.

Four natural definitions

The numerical semigroup generated by $3, 5,$ and 7, or $\langle 3, 5, 7 \rangle$.

gaps	the numbers not in S	$\{1, 2, 4\}$
Frobenius number	the *largest* gap	4
genus	*how many* gaps there are	3
conductor	where the gaps stop, equals Frobenius plus 1	5

Many papers studying numerical semigroups make claims about the Frobenius number: *the largest element not made by the generators.*

Two generators: Sylvester's formula (Chicken McNugget)

Sylvester (1884). If a and b are coprime, then the Frobenius number of the numerical semigroup generated by a and b is

$$ab - a - b.$$

Check it on $\langle 3, 5\rangle$:

$$3 \cdot 5 - 3 - 5 = 7.$$

And indeed 7 is unpayable, while $8 = 3 + 5$, $9 = 3+3+3$, $10 = 5+5$, and everything after that works.

We are not going to prove this now.

Later in the talk we will build a general tool, and Sylvester's formula will be a direct result.

The difficulties of ≥ 3 generators

1. No closed form. For three generators it is a *theorem* (Curtis, 1990) that no finite collection of polynomial or rational formulas can give the Frobenius number based on the three generators.

2. Fast algorithms exist. For instance, for semigroups with three generators, the Frobenius number has been efficiently solvable since Greenberg (1988). Other efficient algorithms exist that will find the Frobenius number for any other number of generators.

No general formula, but computation is entirely possible.

What researchers do instead

Since there is no single formula for *all* inputs, one can look for a formula that works when the generators follow a pattern.

Examples:

- consecutive Fibonacci numbers
- numbers of the form $(2^k + 1)2^n - (2^k - 1)$
- consecutive triangular numbers $\frac{n(n+1)}{2}$
- geometric sequences $m^k,\ m^{k-1}n,\ \ldots,\ n^k$

For each pattern, a paper proves a closed form for the Frobenius number, often split into several cases depending on the parameters.

Dozens of such results exist.

Every one of them is a concrete, finite, *checkable* claim, which is exactly what a computer is good at.

PART 2

The Tool

What can go wrong in a proof

A proof is a chain of reasoning. Every link must hold; one broken link and the conclusion is worthless, even if everything else is perfect.

Written mathematics is full of abbreviations:

- "it is easy to see that..."
- "similarly for the other case"
- "simplifying, we obtain..."

These are almost always fine. Occasionally, one hides a dropped term or a missing case.

In formal papers, this human error can be made on both the part of the researcher and the referee.

A machine ensures that even the tiny mistakes get caught.

Enter proof assistants

A **proof assistant** is software in which every step of a proof is checked mechanically against the axioms of mathematics.

We use Lean 4, together with *Mathlib*, a community-built library of already-formalized mathematics containing hundreds of thousands of theorems.

You build on what is there rather than starting from the axioms every time.

The tradeoff. A formal proof using a proof assistant is much more complex and time-consuming. The shortest known proof of Fermat's Last Theorem would translate into millions of lines of Lean code!

Examples of Lean proofs

```
theorem add_comm_two (a b : Nat) :
    a + b = b + a := by
  induction b with
  | zero ⇒ simp
  | succ n ih ⇒
      rw [Nat.add_succ, Nat.succ_add, ih]
```

```
example : frobeniusNumber [3, 5, 7] = 4 := by
  decide
```

- `theorem` names the claim; `(a b : Nat)` says *for all* natural numbers a, b.
- `by` begins the proof.
- `induction b` splits it into two cases: $b = 0$, and $b = n + 1$ assuming the result for n.
- `ih` is the inductive hypothesis.
- `rw` rewrites the goal using equations already proved.
- `decide` skips all of this: for a finite claim, the kernel just computes.

The kernel

What it does

Reads a completed proof and checks every single step against a short, fixed list of logical rules. Each step must be an assumption, an axiom, or a legal consequence of earlier steps.

What it doesn't do

It cannot search for proofs, simplify, guess, or consult a library. The kernel is an incredibly strict check to a proof.

It stays small. Mathlib has grown to millions of lines; the kernel is a few thousand, and has barely changed. It is short enough that people can read the whole thing and satisfy themselves it is correct.

Computing as a form of proving

Some proofs are general. Others are specific and can be computed:

"the Frobenius number of $\langle 3, 5, 7 \rangle$ is 4"

is a finite statement about specific numbers.

```
example : frobeniusNumber [3,5,7] = 4 := by
  decide
```

`decide` makes Lean's own *kernel* run the computation and confirm the answer.

Every user of Lean trusts this small, carefully-checked core program, the kernel. Nothing new is added.

Decide vs Native decide

Some claims are far too large for `decide` to grind through.

```
example : frobeniusNumber [725,2177,6533,19601,58805,176417,529253] = 1057773 := by native_decide
```

	`decide`	`native_decide`
Who computes	Lean's own kernel	a compiled program, outside Lean's kernel, like any other computer program
How	substitutes definitions and simplifies, step by step — slow	your Lean definitions become machine code that your CPU runs — often thousands of times faster

An empty shelf in the Mathlib library

Mathlib has:

- monoids, submonoids, $\mathbb{N}$ and its order
- vast machinery for algebra and number theory

Mathlib does not have:

- numerical semigroups
- gaps, Frobenius number, genus, conductor
- any of their factorization invariants

PART 3

What We Built

Our goal

The goal. A Lean core that takes the generators of a numerical semigroup and computes its Frobenius number.

Published Papers

Published papers give closed-form answers for whole families of semigroups. We will use these formulas to test if our Lean core is running correctly.

GAP

Every number is computed a second time in GAP, a mature computer-algebra system.

A simple algorithm

0 is payable, and n is payable exactly when $n-g$ is payable for some generator g. So sweep upward, filling in a table, and read off the largest **F**.

The catch

To find thelargest gap you must know when to stop. Stop too early and the calculated Frobenius number will be too small, even though the table is correct.

The bound is provable. . .

Adding generators only shrinks the gap set, so any coprime pair gives a rigorous ceiling: $F(S) \leq ab - a - b$. No guessing required.

. . . but unrealistic. On one real family that ceiling is about 10^{11} which would take too long.

The Apéry set algorithm

Let m be the **multiplicity** — the smallest non-zero element of S, which is just the smallest generator.

The **Apéry set** of S with respect to m is

$$\mathrm{Ap}(S, m) \;=\; \{\, s \in S \;:\; s - m \notin S \,\}.$$

Equivalently: *for each remainder r modulo m, the smallest element of S leaving remainder r.*

The Apéry set of $\langle 3, 5, 7 \rangle$

Multiplicity $m = 3$, so three remainder classes.

remainder	elements of S	smallest
0	0, 3, 6, 9, ...	**0**
1	7, 10, 13, ...	**7**
2	5, 8, 11, ...	**5**

$$\mathrm{Ap}(S, 3) = \{0,\ 5,\ 7\}$$

The Apéry set will always contain m elements, one for each remainder class.

The next two slides show that these m numbers determine both $F(S)$ and $g(S)$.

Theorem 1: the Frobenius number from the Apéry set

Theorem. $F(S) = \max \mathrm{Ap}(S, m) - m.$

Sketch of proof. Write $w = \max \mathrm{Ap}(S, m)$, and let w_r be the Apéry element in the class of r.

(i) *$w - m$ is a gap.* Being in the Apéry set means exactly that $w - m \notin S$. So $F(S) \geq w - m$.

(ii) *Everything above it is payable.* Take $n > w - m$ with remainder r. Since $w_r \leq w$,

$$n > w - m \geq w_r - m,$$

and $n \equiv w_r$, so $n \geq w_r$. Then $n = w_r + qm$ with $q \geq 0$, and both w_r and m lie in S.

Thus $w - m$ is a gap and nothing above it is, so it is the Frobenius number. □

Sylvester as a direct result

Take $S = \langle a, b \rangle$ with a, b coprime, and use $m = a$. Every element of S is $ia + jb$. To be the *smallest* in its remainder class we need $i = 0$, and since $\gcd(a, b) = 1$ the multiples

$$0,\ b,\ 2b,\ \ldots,\ (a-1)b$$

hit all a remainder classes exactly once. So this *is* the Apéry set.

Theorem 1 then gives

$$F(\langle a, b \rangle) = (a-1)b - a = ab - a - b,$$

which is Sylvester's formula from earlier.

Theorem 2: the genus from the Apéry set

Theorem (Brauer–Shockley). $$g(S) \;=\; \frac{1}{m}\left(\left(\sum_{x\in \mathrm{Ap}(S,m)} x\right) \;-\; \frac{m(m-1)}{2}\right).$$

Sketch of proof. Count the gaps one remainder class at a time.

Fix a class $r \neq 0$, with Apéry element w_r. Since w_r is the *smallest* element of S in that class, the gaps there are precisely

$$r,\; r+m,\; r+2m,\; \ldots,\; w_r - m,$$

which is $(w_r - r)/m$ numbers. Summing over all classes, and using $0+1+\cdots+(m-1) = \frac{m(m-1)}{2}$,

$$g(S) \;=\; \sum_{r=0}^{m-1} \frac{w_r - r}{m} \;=\; \frac{1}{m}\left(\left(\sum_{x\in \mathrm{Ap}(S,m)} x\right) \;-\; \frac{m(m-1)}{2}\right). \qquad \square$$

Finding the Apéry set: the procedure

Setup. Imagine a "slot" for each remainder class. Start with 0 in slot 0.

One step

Take a slot r that holds some value d. For each generator g:

- the total $d + g$ is also in S, and we find its remainder $r' = (r + g) \bmod m$;
- if slot r' is unknown, or holds something *bigger* than $d + g$, overwrite it with $d + g$.

One sweep. Do that for $r = 0, 1, \ldots, m - 1$ in order. Anything written earlier in the sweep is available later in the same sweep.

Repeat the sweep $m - 1$ times. Every slot then holds the true minimum.

Why $m-1$ sweeps are enough

Sketch of proof. *An Apéry element is a sum of at most $m-1$ generators.*

Suppose $s \in S$ is written as a sum of k generators, and look at the running totals

$$0, \quad g_1, \quad g_1+g_2, \quad \ldots, \quad s.$$

That is $k+1$ numbers. If $k \geq m$ there are more than m of them, so two must leave the same remainder mod m. Delete the generators sitting between those two: what remains is still a sum of generators, still leaves the same remainder as s, and is *strictly smaller*. So s was not the smallest in its class so it is not an Apéry element.

The rest follows. Each sweep extends the sums found so far by one more generator, so after $m-1$ sweeps every sum of at most $m-1$ generators has been taken into account. Keeping only the smallest value at each slot along the way leaves exactly the Apèry elements. □

The procedure in action

$\langle 4, 7, 18 \rangle$, so $m = 4$. Note $+4$ always lands back on the same slot, so only $+7$ and $+18$ move us.

what happens	array	
start	$[0, -, -, -]$	
sweep 1, $r = 0$: $+7$	$[0, -, -, \mathbf{7}]$	fills slot 3
$r = 0$: $+18$	$[0, -, \mathbf{18}, 7]$	fills slot 2
$r = 2$ holds 18: $+7$	$[0, \mathbf{25}, 18, 7]$	fills slot 1
$r = 3$ holds 7: $+7$	$[0, 25, \mathbf{14}, 7]$	$14 < 18$, overwrite
sweep 2, $r = 2$ holds 14: $+7$	$[0, \mathbf{21}, 14, 7]$	$21 < 25$, overwrite
sweep 3	$[0, 21, 14, 7]$	nothing changes

In Lean, part 1: one step, and one sweep

An array of m slots; slot r holds the cheapest total yet found in class r. `none` = not reached.

```
def relaxResidue (gens : List Nat) (m : Nat)
    (dist : Array (Option Nat)) (r : Nat) :=
  match dist[r]! with
  | none   ⇒ dist
  | some d ⇒
    gens.foldl (fun acc g ⇒
      let r' := (r + g) % m
      match acc[r']! with
      | none     ⇒ acc.set! r' (some (d + g))
      | some cur ⇒ if d + g < cur then acc.set! r' (some (d+g)) else acc) dist
```

From one residue we have already reached, try every generator and keep whichever total is smaller.

```
def relaxRound (gens : List Nat) (m : Nat) (dist : Array (Option Nat)) :=
  (List.range m).foldl (relaxResidue gens m) dist
```

In Lean, part 2: sweep, then read the answers off

Start from "0 costs nothing, everything else unknown", sweep $m-1$ times, then apply the two theorems.

```
def aperySet (gens : List Nat) (m : Nat) : Array (Option Nat) :=
  let init := (Array.replicate m none).set! 0 (some 0)
  (List.range (m - 1)).foldl (fun acc _ ⇒ relaxRound gens m acc) init

def frobeniusNumber (gens : List Nat) : Nat :=
  let m := gens.foldl Nat.min gens.head!
  let dist := aperySet gens m
  let reps := (List.range m).drop 1 |>.map (fun r ⇒ (dist[r]!).getD 0)
  reps.foldr Nat.max 0 - m

def genusApery (gens : List Nat) : Nat :=
  let m := gens.foldl Nat.min gens.head!
  let dist := aperySet gens m
  let total := (List.range m).foldl (fun acc r ⇒ acc + (dist[r]!).getD 0) 0
  (2 * total - m * (m - 1)) / (2 * m)
```

Theorem 1 and Theorem 2, transcribed. Note $m - 1$ sweeps, not "until nothing changes": Lean requires every function to visibly finish, so the bound we proved is what makes the code legal.

A published formula for the genus

Liu, Xin, Ye and Yin, *A Note on Generalized Repunit Numerical Semigroups* (arXiv:2306.10738)

The family. A *repunit* in base b is a number written with all 1s. In base 2:

$$7 = 111_2, \qquad 15 = 1111_2, \qquad 31 = 11111_2,$$

and the semigroup they generate is $\langle 7, 15, 31 \rangle$. Every base b and every n gives one of these.

The claim. The paper gives a closed form for the genus of any such semigroup:

$$g(S) \;=\; \frac{b^n}{2}\left(\frac{b^n - 1}{b - 1} + n - 1\right).$$

A formula like this is exactly what our core can check: pick small b and n, list the generators, and count the gaps.

Checking the genus formula

$b = 2$, $n = 3$ — $\langle 7, 15, 31 \rangle$

The formula gives

$$\tfrac{8}{2}(7 + 3 - 1) = \mathbf{36}.$$

Counting the gaps gives 32.

$b = 3$, $n = 2$ — $\langle 4, 13 \rangle$

The formula gives

$$\tfrac{9}{2}(4 + 2 - 1) = \mathbf{22.5}.$$

A genus counts gaps. It cannot be a fraction.

What went wrong. This corollary is the special case $d = 1$ of the paper's own earlier theorem — and that theorem is correct. Working the substitution through carefully gives $n - 2$ where the corollary prints $n - 1$.

Corrected: $g(S) = \dfrac{b^n}{2}\left(\dfrac{b^n - 1}{b - 1} + n - 2\right)$ which gives 32 and 18.

A published formula for the Frobenius number

Arias, Borja and Rhenals, on semigroups built from sequences $c\,a^t - d$ (arXiv:2111.04899)

The family. Start with the sequence $x_t = 5 \cdot 3^{\,t} - 1$:

$$4, \quad 14, \quad 44, \quad 134, \quad 404, \quad 1214, \ \ldots$$

Take the numbers from position 1 onwards as generators: $\langle 14, 44, 134, 404 \rangle$. They are all even, so divide through by 2 — otherwise no total is odd and there is no Frobenius number at all:

$$\langle 7,\ 22,\ 67,\ 202 \rangle.$$

The claim. For this family the paper prints the closed form $F = 5 \cdot 3^n R_n + 1$, where R_n is a repunit with n digits in base 3. At $n = 1$ that is

$$5 \cdot 3 \cdot 1 + 1 = \mathbf{16}.$$

Checking the Frobenius formula

The claim: the largest total you *cannot* pay with $7, 22, 67, 202$ is **16**.

We can see that some more impossible sums include $17, 18, 19$, and 20. The true answer is actually 104.

What went wrong. The line *above* the closed form in the paper is correct, and evaluates to 104. The error is in the last step: a quantity was replaced by a similar-looking one that happens to agree in the paper's previous example but not in this one.

Corrected: $F = \dfrac{25 \cdot 3^{2n} - 5 \cdot 3^{n} - 2}{2}$ giving 104 at $n = 1$ and 989 at $n = 2$.

How we handle discrepancies

1 · Suspect ourselves

A mistyped generator, a truncated table, a bound set too low. *This is the most likely explanation, and it has happened to us* — our first attempt at the hardest case used a wrong input list and produced a confidently wrong number.

2 · Look for self-contradiction

The strongest evidence needs no computer at all: a value that cannot be an integer, or a formula contradicting the line above it.

3 · Two systems

Only when Lean and GAP independently agree do we call anything confirmed. Until then it stays a *candidate*.

Note. These are ordinary arithmetic slips in long derivations that were minor enough to be missed by the authors and reviewer, and yet trivial for a machine to notice. This gap is the reason for formalizing mathematics.

Takeaways

1 · The problem. For two coprime coins the answer is $ab - a - b$. For three or more no formula exists, so we compute.

2 · The tool. Lean checks every step against a small fixed kernel. Larger claims can be checked via `native decide` and cross checked through GAP.

3 · The method. Keep one number per remainder — the Apéry set — instead of searching every integer. We proved $F(S) = \max \mathrm{Ap} - m$, the genus formula, and that $m - 1$ sweeps suffice.

4 · The result. 45 machine-checked theorems across 10 papers. Six published values did not match; two were shown here, each traced to one identifiable line.

The Bi-UFS Positive Conjecture for Algebraic Monogenic Semidomains

Joint work by

Aaditya Bilakanti, Amrit Kandasamy, Hengrui Liang, Jason Yang, and Alan Yao

Jonathan Liu

CMI 2026

General notation

Used throughout.

- $\mathbb{N} := \{1, 2, 3, \dots\}$
- $\mathbb{N}_0 := \mathbb{N} \cup \{0\} = \{0, 1, 2, 3, \dots\}$
- $\mathbb{R}$ and $\mathbb{Q}$ denote the real and rational numbers.
- For $S \subseteq \mathbb{R}$ and $r \in \mathbb{R}$,

$$S_{\geq r} := \{s \in S : s \geq r\} \qquad \text{and} \qquad S_{>r} := \{s \in S : s > r\}.$$

Preliminaries: monoids

Definition

A **monoid** $M = (M, *)$ is a set with a binary operation $*$ such that

- $*$ is associative: $b * (c * d) = (b * c) * d$ for all $b, c, d \in M$;
- there is an identity e with $b * e = e * b = b$ for all $b \in M$.

We use multiplicative notation for general properties of monoids.

Examples. $(\mathbb{N}_0, +)$ $(\mathbb{Q}_{\geq 0}, +)$

Preliminaries: units, atoms, primes

A monoid M is **commutative** if $a \cdot b = b \cdot a$ for all $a, b \in M$, and **cancellative** if $a \cdot b = a \cdot c$ implies $b = c$.

In this talk, every monoid is assumed commutative and cancellative.

Definition

Let M be a monoid.

- $u \in M$ is a **unit** if $u \cdot v = e$ for some $v \in M$.
- A nonunit a is an **atom** if $a = xy$ forces x or y to be a unit.

We use $\mathcal{A}^+(S)$ and $\mathcal{A}^*(S)$ to denote the set of additive and multiplicative atoms of S.

Preliminaries: unique factorization and semidomains

Definition

A unique factorization monoid **(UFM)** is a monoid in which every nonunit factors uniquely into atoms, up to order and associates.

$(\mathbb{N}, \cdot)$ is a UFM, by the Fundamental Theorem of Arithmetic.

Preliminaries: unique factorization and semidomains

Definition

A unique factorization monoid **(UFM)** is a monoid in which every nonunit factors uniquely into atoms, up to order and associates.

$(\mathbb{N}, \cdot)$ is a UFM, by the Fundamental Theorem of Arithmetic.

Definition

Let S carry an addition and a multiplication.

- S is a **semiring** if $(S, +)$ and $(S^*, \cdot)$ are commutative monoids and multiplication distributes over addition, where $S^* = S \setminus \{0\}$
- A semiring S is a **semidomain** if it has no zero divisors.

Examples. $\mathbb{N}_0$ is a semiring and a semidomain.

Motivation from $\mathbb{Z}$

A familiar starting point

Consider a unique factorization domain such as $\mathbb{Z}$. There are two natural monoids:

$$(\mathbb{Z}, +) \qquad \text{and} \qquad (\mathbb{Z} \setminus \{0\}, \cdot).$$

- Multiplicatively, we know $\mathbb{Z} \setminus \{0\}$ has unique factorization.
- Additively, $\mathbb{Z}$ is a group, so every element is a unit. Thus there is no interesting additive factorization theory.

What changes for semidomains?

Unlike $\mathbb{Z}$, a semidomain has no "subtraction" in general. So now addition can have a genuine factorization theory at the same time as multiplication.

Can both factorizations be unique?

In a semidomain, when does addition and multiplication both have genuine unique factorization at the same time?

Definition

A semidomain S is a **bi-UFS** if both

$$(S, +) \qquad \text{and} \qquad (S^*, \cdot), \qquad S^* := S \setminus \{0\},$$

are unique factorization monoids.

Can you think of a semidomain where both factorizations should be unique?

What happens for semidomains?

The first example to try: $\mathbb{N}_0$

Additively,

$$n = \underbrace{1 + \cdots + 1}_{n \text{ times}},$$

so $(\mathbb{N}_0, +)$ has unique factorization. Multiplicatively, the Fundamental Theorem of Arithmetic gives unique factorization in $(\mathbb{N}, \cdot)$. Therefore $\mathbb{N}_0$ is a bi-UFS.

The next natural example fails:

The polynomial semidomain $\mathbb{N}_0[x]$ fails multiplicative unique factorization:

$$(x^3 + 1)(x^2 + x + 1) = (x + 1)(x^4 + x^2 + 1).$$

The four displayed factors are multiplicative atoms.

Monogenic semidomains

Definition

For $\alpha \in \mathbb{R}_{>0}$, the **monogenic semidomain** generated by α is

$$\mathbb{N}_0[\alpha] = \{f(\alpha) : f(x) \in \mathbb{N}_0[x]\}.$$

For what α is $\mathbb{N}_0[\alpha]$ a bi-UFS?

Monogenic semidomains

Definition

For $\alpha \in \mathbb{R}_{>0}$, the **monogenic semidomain** generated by α is

$$\mathbb{N}_0[\alpha] = \{f(\alpha) : f(x) \in \mathbb{N}_0[x]\}.$$

For what α is $\mathbb{N}_0[\alpha]$ a bi-UFS?

Theorem (Bilakanti–Kandasamy–Liang–Liu–Yang–Yao). For every positive algebraic number α,

$$\mathbb{N}_0[\alpha] \text{ is a bi-UFS} \iff \mathbb{N}_0[\alpha] = \mathbb{N}_0.$$

Interval exclusions

Lemma 3.9. If S is a bi-UFS positive semidomain, then $S \cap (0,1) = \varnothing$.

Why this matters

If $a \mid_{S^*} b$, then $b = ac$ for some $c \in S^*$, and hence

$$c \geq 1 \quad \Longrightarrow \quad b \geq a.$$

Lemma 3.10. Let $\ell = \min\left(\mathcal{A}^+(S) \setminus \{1\}\right)$. Then $1 < \ell \leq \varphi$.

We will use this lemma to tackle the quadratic case of the bi-UFS positive conjecture.

Additive domains

Theorem (Correa-Morris–Gotti). Let $\alpha \in \mathbb{R}_{>0}$ be algebraic of degree d. If $(\mathbb{N}_0[\alpha], +)$ is a UFM, then

$$\mathcal{A}^+(\mathbb{N}_0[\alpha]) = \{1, \alpha, \dots, \alpha^{d-1}\},$$

and every element has a unique additive normal form $c_0 + c_1\alpha + \cdots + c_{d-1}\alpha^{d-1}$ with $c_i \in \mathbb{N}_0$.

We first analyze the case when $\deg(\alpha) = 2$.

Example. If $S = \mathbb{N}_0[\alpha]$ is bi-UFS and $\deg(\alpha) = 2$, then $\mathcal{A}^+(S) = \{1, \alpha\}$. Since $\ell = \min(\mathcal{A}^+(S) \setminus \{1\})$, we get $\ell = \alpha$, and therefore

$$1 < \alpha \leq \varphi.$$

The quadratic case

Theorem (Bilakanti–Kandasamy–Liang–Liu–Yang–Yao). If $S = \mathbb{N}_0[\alpha]$ is bi-UFS and $\deg(\alpha) = 2$, the least-atom bound gives $1 < \alpha \leq \varphi$. Together with $\alpha^2 = a\alpha + b$, this leaves only

$$\alpha = \sqrt{2} \qquad \text{or} \qquad \alpha = \varphi.$$

What this tells us

There are infinitely many positive quadratic algebraic numbers α that could be bi-UFS. We have now restricted it to just two cases:

$$\alpha = \sqrt{2} \qquad \text{or} \qquad \alpha = \varphi.$$

Thus, for bi-UFS quadratic monogenic semidomains, it now suffices to consider the two cases $\mathbb{N}_0[\sqrt{2}]$ and $\mathbb{N}_0[\varphi]$.

The two cases are not bi-UFS

Example 3.20 — $\mathbb{N}_0[\varphi]$ is not a UFM

Using $\varphi^2 = \varphi + 1$,

$$5(1+\varphi) = (2+\varphi)^2.$$

Both sides are products of multiplicative atoms, giving two distinct factorizations.

Example 3.21 — $\mathbb{N}_0[\sqrt{2}]$ is not a UFM

Using

$$7(1+\sqrt{2}) = (1+2\sqrt{2})(3+\sqrt{2}),$$

again with multiplicative atoms on both sides.

So if $\deg(\alpha) = 2$, then $\mathbb{N}_0[\alpha]$ is not bi-UFS.

Degree three and higher

Suppose that $\mathbb{N}_0[\alpha]$ is bi-UFS, where $\alpha > 0$ is algebraic with $\deg(\alpha) = d \geq 3$.

Minimal polynomial

Let $m_\alpha(x) \in \mathbb{Q}[x]$ be the minimal polynomial of α, so

$$\deg m_\alpha = d, \qquad m_\alpha(x) = x^d - q_\alpha(x).$$

Since $(\mathbb{N}_0[\alpha], +)$ is a UFM, the Correa–Morris–Gotti theorem gives

$$\mathcal{A}^+(\mathbb{N}_0[\alpha]) = \{1, \alpha, \ldots, \alpha^{d-1}\}.$$

The goal is now to show that multiplicative unique factorization places restrictions on $q_\alpha(x)$ such that no possible minimal polynomial remains.

Higher-degree contradictions

Lemmas 4.2–4.6. Assume $d = \deg(\alpha) \geq 3$ and that $\mathbb{N}_0[\alpha]$ is bi-UFS. Writing $m_\alpha(x) = x^d - q_\alpha(x)$, the argument forces

$$\boxed{q_\alpha(1) < 4} \qquad \text{and} \qquad \boxed{q_\alpha(x) \in \{0,1\}[x]}.$$

Theorem 4.7 (Bilakanti–Kandasamy–Liang–Liu–Yang–Yao). Every remaining polynomial contradicts multiplicative unique factorization. So if $\alpha > 0$ is algebraic of degree $d \geq 3$, then

$$\boxed{\mathbb{N}_0[\alpha] \text{ is not a bi-UFS}}.$$

Conclusion

Main Takeaways

- $\mathbb{N}_0$ is the basic positive semidomain with unique factorization under both addition and multiplication.
- If $\deg(\alpha) = 2$, then the bi-UFS assumption forces

$$\alpha = \sqrt{2} \qquad \text{or} \qquad \alpha = \varphi,$$

  and both cases fail.
- If $\deg(\alpha) \geq 3$, then the minimal polynomial of α is forced into forms incompatible with multiplicative unique factorization.
- This leaves $\deg(\alpha) = 1$. When this is the case we have $\mathbb{N}_0[\alpha] = \mathbb{N}_0$, proving the bi-UFS positive conjecture for monogenic positive semidomains.

References

A. Bilakanti, M. Gotti, A. Kandasamy, H. Liang, J. Liu, H. Polo, J. Yang, and A. Yao, *The Bi-UFS Positive Conjecture for Algebraic Semidomains*, preprint, 2026.

N. R. Baeth, S. T. Chapman, and F. Gotti, *Bi-atomic classes of positive semirings*, Semigroup Forum **103** (2021), 1–23.

J. Correa-Morris and F. Gotti, *On the additive structure of algebraic valuations of polynomial semirings*, J. Pure Appl. Algebra **226** (2022), Article 107104.

A. Geroldinger and F. Halter-Koch, *Non-Unique Factorizations: Algebraic, Combinatorial and Analytic Theory*, Chapman & Hall/CRC, 2006.

Acknowledgements

We thank our **mentors** Felix Gotti, Marly Gotti, Harold Polo, and Victor Gonzalez for their guidance, patience, and countless hours spent supporting both our research and preparation of this presentation.

We thank the **SWIM organizers** for the opportunity to share this work, and the **PRIMES** and **CMI** programs for giving us this research opportunity.

And thank *you*, our **audience**, for your time and attention today.

Thank You

Questions?