Title Page:

Structural-Numerical-Symmetry-Algorithm-Mikhail-Yushchenko-2025

Algorithm of Structural Numerical Symmetry (SNS)

Author: Yushchenko Mikhail Yuryevich

Date: May 19, 2025

Abstract:

This project is dedicated to the research and development of an original Structural Numerical Symmetry (SNS) algorithm, which represents a universal approach to manipulating numbers and identifying patterns in their structure.

The proposed algorithm enables effective division of a number into blocks, performing calculations, and combining the results, ensuring a high degree of accuracy and reliability.

The main principles and stages of the algorithm include:

* Splitting a natural number into blocks while maintaining closeness in digit length.
* Subsequent local multiplication of each block by a single natural number.
* Combining the obtained results into a single construct.
* Analyzing the resulting value and comparing it with the traditional multiplication method.

Practical application of this algorithm is possible in various fields, including number theory, computer science, biology, economics, chemistry, music, and literature (over 10 disciplines in total). Conducted experimental studies have confirmed the stability and robustness of the algorithm, opening new prospects for further research and practical implementation.

Table of Contents:

1. History of the Idea's Emergence

How the hypothesis about SNS (Structural Numerical Symmetry) was born, which later became a phenomenon due to its proof: on may 4, 2025, I, Yushchenko Mikhail Yuryevich, faced a task: calculate by hand (999 ^ 9999) * 3. This number turned out to be too large for direct multiplication. I then tried to split it into blocks, multiply each block by a coefficient, and then combine the results. However, this approach produced a colossal number of errors, but I continued extracting new and new numbers from it. All these unsuccessful attempts led me to formulate my own hypothesis. Thus, the hypothesis of structural numerical symmetry was born, the essence of which is as follows:

Within the decimal numeral system, for any integer $N \geq 10$, if it is split into $m \geq 2$ blocks, maximally close in digit length (so the number of digits in all blocks differs by no more than one), where $m \in N$ but not exceeding the digit count of N, each block is multiplied by the same natural number k, and then the results are concatenated into a decimal number PQ, then:

* Complete match is possible: PQ = N * k
* The beginning or end may match
* Partial symmetry may be observed: the beginning or end matches
* Both beginning and end match, despite different digit lengths
* Question: Can there exist a number N for which none of the rules holds?

It was tested programmatically on millions of numbers. See file: `Structural_Numerical_Symmetry.jl`(https://colab.research.google.com/drive/1bqUqFugQXDzOB7z9L588UBdktPfHVF3K?usp=sharing). No case without matches was found.

Then I wondered, what if we extend this hypothesis to all natural numbers? i.e., for those where $N<10$. Then I obtained a more generalized formulation:

Within the decimal numeral system, for any natural number N, if: N is split into $m \geq 2$ natural blocks, maximally close in digit length (the difference in the number of digits between any two blocks does not exceed one), $m \in N$, but m should not exceed the digit count of N.

If $N<10$, leading zeros are added before the number so its length becomes equal to m. Then each block is multiplied by the same natural number k.

The results are concatenated into a decimal number PQ. For numbers $N<10$, when comparing with classical multiplication, leading zeros are removed. Then at least one of the following conditions holds:

* Complete match: PQ = N * k.
* The beginning or end matches.
* Both the beginning and end match, but PQ and N * k have different digit lengths.

* Question: Can there exist a number N for which none of the rules holds?

* Answer: No, such a number does not exist! See proof.

Thus, the hypothesis became a phenomenon.

2. History of the Algorithm for Structural Numerical Symmetry

Well, since N itself tends to infinity, and besides, a match only at the beginning was never found, I concluded that if the beginning matches, then the end also matches between the operator PQ and the classical product N * k, with PQ and N * k necessarily having different digit lengths. This is a key difference from a complete match. However, the reverse dependency does not hold. Another phenomenon - in all cases, whether it's a complete match, a match at both beginning and end, or a match only at the end, the following phenomenon is observed: everywhere and always, there is a match at the end. Thus, the idea for an algorithm for Structural Numerical Symmetry arose.

3. The Algorithm Itself

The Structural Numerical Symmetry Algorithm works as follows:

Within the decimal numeral system, for any natural number N, if: N is split into $m \geq 2$ natural blocks, maximally close in digit length (the difference in the number of digits between any two blocks does not exceed one), $m \in N$, but not greater than the digit count of the number N itself. If $N<10$, leading zeros are added before the number so its length becomes equal to m. Then each block is multiplied by the same natural number k, results are concatenated into a decimal number PQ. For numbers $N<10$, when comparing with classical multiplication, leading zeros are removed. Then one of the following conditions necessarily holds: complete match: PQ = N * k, the end matches, both beginning and end match, but PQ and N * k have different digit lengths.

4. Detailed Explanations of Terms

Description of terms:

*** What is SNS?**

This is a method representing an original approach in number theory, aimed at finding patterns and providing a universal description of number structure through their division into blocks and identification of symmetrical properties.

*** What is N?**

N is any natural number.

*** What is k?**

k is absolutely any natural multiplication coefficient.

*** What is m?**

m is the number of blocks into which number N is split.

*** What is PQ?**

PQ is the digital composition (concatenation) of the results of multiplying the blocks of the original number, not an arithmetic sum. This construct lies at the heart of analyzing digital stability and structural matches in this work.

*** What is Nk?**

Nk is the result of classical multiplication of N by k (i.e., N * k).

*** How many digits must match?**

At least 1 digit, or the entire number completely.

*** How exactly to split?**

It is recommended to split from left to right, similar to the positional notation of a number in the decimal system. BUT! You can split however you like, only (!) in accordance with the SNS rules.

1."Within the decimal numeral system".

This means we work with numbers as they are written in our usual form: from 0 to 9, left to right, taking the digit's position into account.

2."Natural number N".

This means:N is any positive integer: 1, 2, 3, ..., 100, ..., 1000000. It is not required to be prime, even, or large.

3."N is split into $m \geq 2$ parts".

This tells us that: We cannot split the number into 1 block — the minimum number of blocks: 2. We can split into 2, 3, 4... , but not more than the number of digits in N.

4."Maximally close in digit length".

This means: During splitting, all blocks should be very similar in digit length. The difference in the number of digits between blocks does not exceed 1.

5."$m \in \mathbb{N}$, but not greater than the digit count of the number N itself".

That is: If N = 1234 (4 digits), then m can be: 2, 3, 4. Cannot split into 5 blocks because the number only has 4 digits.

6."If $N < 10$, then leading zeros are added before the number so its length becomes equal to m".

This is needed so small numbers can also be processed by the same rules as large ones.

7."Each block is multiplied by the same natural number k".

Means: All blocks are multiplied by the same factor. The factor can be any natural number: 1, 2, 3, ...

8."Results are concatenated into a decimal number PQ".

This means: The multiplied blocks are joined as a concatenation, not as a mathematical sum. This is not simple mathematical multiplication, but a structural transformation.

9."Then one of the following conditions necessarily holds".

After all actions, always: The entire number matches completely. Or the end matches. Or both the beginning and the end match, even if the digit length of the numbers is different. This principle has never been violated in millions of checks!

5. Examples for Each Term Explanation

1.This means: We work with numbers as we are used to — in the usual form. Numbers are written with digits from 0 to 9. The position of a digit is important (e.g., 123 ≠ 321 because positions are different).

*** Example:**

* N = 123456789
* m = 3
* k = 7
* splitting → ["123", "456", "789"]
* per-block multiplication: → ["861", "3192", "5523"] concatenation → PQ = "86131925523"
* comparison of numbers:

N * K = 864197523 and PQ = 86131925523

The beginning (8) and the end (3) match.

2. No need for the number to be prime or even. Any positive integer works.

* Example:

* N = 13
* m = 2
* k = 7
* splitting → ["1", "3"]
* per-block multiplication: → ["7", "21"] concatenation → PQ = "721"
* N * K = 91
* comparison of numbers:
* PQ = 721
* N * k = 91
* Only the end (1) matches.

3. Minimum of 2. Can be more: 3, 4, 5... but not more than the digit count of the number

* Example:

* N = 101
* m = 2
* k = 7
* Splitting → ["10", "1"]

* per-block multiplication: → ["70", "7"] concatenation: → PQ = "707"
* Comparison:

  N * K = 707 and PQ = 707

  Complete match.

4. All have very similar digit length. Difference between blocks is no more than 1 digit.

* Example:
  * N = 12345
  * m = 3
  * concatenation: numbers (N, m) → ["12", "34", "5"]

here: 2, 2, 1 → difference does not exceed one.

Condition satisfied.

5. If a number has 4 digits, you cannot divide it into 5 blocks. You can divide into 2, 3, and 4.

* Example:
  * N = 1234
  * digit count(N) = 4
  * m = 5 → Invalid value.
  * m = 4 → Valid value.
  * m = 3 → Valid value
  * m = 2 → Valid value.

6. This allows using small numbers in the full verification system.

* Example:
  * N = 3
  * k = 7
  * m = 2
  * → digit count(N) = 1 → pad to 2 digits → "03"
  * splitting → ["0", "3"]
  * multiplication of parts → ["0×7=0", "3×7=21"] per-block multiplication: → PQ = "021"
  * N * K = 21
  * After removing leading zeros: PQ = 21 → N * k = 21 → Complete match

7. This is important for preserving the structure. where k is a natural number, can be any: 1, 2, 3, ..., 99999999

* Example:
  * N = 1234
  * m = 2
  * k = 4
  * splitting → ["12", "34"]
  * per-block multiplication: → ["48", "136"] concatenation → PQ = "48136"
  * N * K = 1234 × 4 = 4936
  * PQ = 48136
  * compare PQ and N * k

The beginning ("4") and the end ("6") match.

8. This is not just mathematical multiplication. This is a digital transformation: multiply, then join as concatenation.

* Example:
  * N = 899766
  * m = 2
  * k = 4
  * splitting → ["899", "766"]
  * per-block multiplication: → ["3596", "3064"] concatenation → PQ = "35963064"
  * compare PQ and N * k
  * PQ = 35963064
  * N * K = 899766 × 4 = 3599064

The beginning ("3") and the end ("4") match.

9. One of the following will always match: Complete match, Only the end, Both beginning and end. No case without matches has been found.

* Example:
  * N = 11
  * m = 2
  * k = 7
  * splitting → ["1", "1"]
  * per-block multiplication: → ["7", "7"] concatenation: → PQ = "77"
  * PQ = 77

* N * K = 77
* Compare PQ with N * k
* Complete match!

6. Scientific Significance.

The algorithm works for all natural numbers, including primes, composites, large powers. It has a formal proof and is empirically tested on millions of numbers.

It has structural invariance:

If the beginning matches → the end will necessarily match.

But if only the end matches → the beginning may not match.

The proof, since it was derived from the hypothesis, and currently from the phenomenon of SNS (Structural Numerical Symmetry), its proof lies within the framework of SNS. See proof.

This indicates a deep pattern that **can be** used in:

* Number Theory
* Computer Science
* Biology
* Physics
* Chemistry
* Economics
* Medicine
* Astronomy
* Music
* Literature
* History
* Logistics
* Game Theory
* Psychology
* Philosophy
* and other fields.

7. Algorithm Operation Principle

How does it work? (In simple terms)

1. Within the decimal numeral system, take any natural number N.
2. Split it into m ≥ 2 natural blocks, close in digit length.
3. If the number is small (N < 10), pad it with zeros to the required length.
4. Multiply each block by natural number k.
5. Join the results as a concatenation: → get PQ.
6. When comparing for numbers N<10 with classical multiplication, remove leading zeros.
7. Compare PQ with the classical product NK = N × k.
8. There will always be at least a partial match!

Examples of Algorithm Operation:

N	**M**	**k**	**Splitting**	**PQ**	**NK**	**Result**
101	2	7	["10", "1"]	"707"	"707"	Complete match
135	2	7	["13", "5"]	"9135"	"945"	Beginning and end match
13	2	7	["12", "34"]	"721"	"91"	Only the end matches
1234	2	4	["12", "34"]	"48136"	"4936"	Beginning and end match
1001	2	7	["10", "01"]	"7007"	"7007"	Complete match

All these examples show that the beginning and end of numbers maintain a connection, even if the middle part changes.

8. Verification Statistics

Range: from 1 to 10,000,000, m=2, k=7

✅ Complete matches: 1,430,758

🔄 Beginning and end match: 8,560,838

🔄 Only beginning matches: 0

🔄 Only end matches: 8,404

❌ No matches: 0

Download data (https://drive.google.com/file/d/1P1vKiKhUdqpm80JHrs_K-ZRraCqkzc4a/view?usp=sharing)

The same, but with k=99,999,999

📊 Summary statistics:

✅ Complete matches: 10,999

🔄 Beginning and end match: 9,969,075

🔄 Only beginning matches: 0

🔄 Only end matches: 19,926

❌ No matches: 0

Download data (https://drive.google.com/file/d/16pzzA4N52ig8YiQ5YLHuzi9M1XtE4cUO/view?usp=sharing)

9. Examples for Each Rule.

Complete match:

* N = 101
* m = 2
* k = 7
* splitting → ["10", "1"]
* per-block multiplication: → ["70", "7"] concatenation: → PQ = 707
* Comparison of numbers

  NK = 101 × 7 = 707 and PQ = 707

  Result: Complete match

Beginning and end match:

* N = 899766
* m = 2
* k = 4
* splitting → ["899", "766"]
* per-block multiplication: → ["3596", "3064"] concatenation → PQ = 35963064
* compare

  NK = 899766 × 4 = 3599064 and PQ = 35963064
* Match: "3" and "4"
* Beginning and end match

Only the end matches:

* N = 13
* m = 2
* k = 7
* splitting → ["1", "3"]
* per-block multiplication → ["7", "21"] concatenation: → PQ = "721"
* compare

$N * K = 13 \times 7 = 91$ and $PQ = 721$

Only the end ("1") matches.

10. Possible (but non-obvious) Applications in Real-World Tasks

1. Number Theory: https://github.com/Misha0966/New-project/blob/For-English/Number%20Theory.md

2. Computer Science: https://github.com/Misha0966/New-project/blob/For-English/SNS%20in%20Computer%20Science%20(The%20P%20vs%20NP%20Problem).md

3. Biology: https://github.com/Misha0966/New-project/blob/For-English/Biology%20(Genetics).md

4. Physics: https://github.com/Misha0966/New-project/blob/For-English/Application%20in%20Physics%20(Law%20of%20Conservation%20of%20Energy).md

5. Chemistry: https://github.com/Misha0966/New-project/blob/For-English/Application%20of%20SNS%20in%20Chemistry%20(Mendeleev's%20Periodic%20Table).md

6. Economics: https://github.com/Misha0966/New-project/blob/For-English/In%20Economics.md

7. Medicine: https://github.com/Misha0966/New-project/blob/For-English/Application%20in%20Medicine.md

8. Astronomy: https://github.com/Misha0966/New-project/blob/For-English/In%20Astronomy.md

9. Music: https://github.com/Misha0966/New-project/blob/For-English/In%20Music.md

10. Literature: https://github.com/Misha0966/New-project/blob/For-English/In%20Literature.md

11. History: https://github.com/Misha0966/New-project/blob/For-English/In%20History.md

12. Logistics: https://github.com/Misha0966/New-project/blob/For-English/In%20Logistics.md

13. Game Theory: https://github.com/Misha0966/New-project/blob/For-English/In%20Game%20Theory.md

14. Psychology: https://github.com/Misha0966/New-project/blob/For-English/In%20Psychology.md

15. Philosophy: https://github.com/Misha0966/New-project/blob/For-English/In%20Philosophy.md

11. Code implementation.

Julia Code Translation

Visual Studio Code Runtime Environment

```
using Printf
using CSV
using DataFrames
using Base.Threads
using ProgressMeter

function split_number_str(N::Integer, m::Integer)
s = string(N)
if N < 10
s = lpad(s, m, '0')
end
len = length(s)
base_len = div(len, m)
remainder = len % m
parts = String[]
idx = 1
for i in 1:m
current_len = base_len + (i <= remainder ? 1 : 0)
push!(parts, s[idx:idx+current_len-1])
idx += current_len
end
return parts
end
function multiply_preserve_length(part::String, k::Integer)
num = parse(BigInt, part) * k
result = string(num)
```

```
return lpad(result, length(part), '0')
end
function remove_leading_zeros(s::String)
if all(c -> c == '0', s)
return "0"
else
idx = findfirst(c -> c != '0', s)
return s[idx:end]
end
end
function compare_pq_nk(pq::String, nk::String)
if pq == nk
return "Full match"
end
min_len = min(length(pq), length(nk))
prefix_match = 0
for i in 1:min_len
pq[i] == nk[i] ? prefix_match += 1 : break
end
suffix_match = 0
for i in 1:min_len
pq[end - i + 1] == nk[end - i + 1] ? suffix_match += 1 : break
end
if prefix_match > 0 && suffix_match > 0
return "Prefix and suffix match"
elseif prefix_match > 0
return "Prefix matches only"
elseif suffix_match > 0
return "Suffix matches only"
else
```

```
return "No match"
end
end
function check_algoritm(N::Integer, m::Integer, k::Integer)
N_str = string(N)
nk_str = string(N * k)
parts_str = split_number_str(N, m)
multiplied_parts_str = [multiply_preserve_length(p, k) for p in parts_str]
pq_str = join(multiplied_parts_str)
pq_clean = remove_leading_zeros(pq_str)
nk_clean = remove_leading_zeros(nk_str)
result = compare_pq_nk(pq_clean, nk_clean)
return (
N = N,
m = m,
k = k,
parts = string(parts_str),
multiplied_parts = string(multiplied_parts_str),
PQ = pq_clean,
NK = nk_clean,
result = result
)
end
function run_tests_parallel(start_N::Integer, stop_N::Integer, m::Integer, k::Integer)
results21_df = DataFrame(
N = Int[],
m = Int[],
k = Int[],
parts = String[],
multiplied_parts = String[],
PQ = String[],
NK = String[],
result = String[]
)
count_full = Atomic{Int}(0)
count_partial_start = Atomic{Int}(0)
count_partial_end = Atomic{Int}(0)
count_partial_both = Atomic{Int}(0)
count_none = Atomic{Int}(0)
@showprogress "🚀 Testing N ∈ [$start_N, $stop_N], m = $m, k = $k" for N in start_N:stop_N
res = check_algoritm(N, m, k)
Threads.atomic_add!(count_full, res.result == "Full match" ? 1 : 0)
Threads.atomic_add!(count_partial_start, res.result == "Prefix matches only" ? 1 : 0)
Threads.atomic_add!(count_partial_end, res.result == "Suffix matches only" ? 1 : 0)
Threads.atomic_add!(count_partial_both, res.result == "Prefix and suffix match" ? 1 : 0)
Threads.atomic_add!(count_none, res.result == "No match" ? 1 : 0)
push!(results21_df, [
res.N,
res.m,
res.k,
res.parts,
res.multiplied_parts,
res.PQ,
res.NK,
res.result
])
end
full = count_full[]
partial_start = count_partial_start[]
partial_end = count_partial_end[]
```

```
partial_both = count_partial_both[]
none = count_none[]
println("\n Saving results to CSV...")
CSV.write("results2.csv", results21_df)
open("statistics7.txt", "w") do io
write(io, "Structural Numerical Symmetry Hypothesis\n")
write(io, "========================================\n")
write(io, "N range: [$start_N, $stop_N]\n")
write(io, "Number of parts m = $m\n")
write(io, "Multiplier k = $k\n")
write(io, "----------------------------------------\n")
write(io, " Full matches: $full\n")
write(io, " Prefix and suffix match: $partial_both\n")
write(io, " Prefix matches only: $partial_start\n")
write(io, " Suffix matches only: $partial_end\n")
write(io, " No matches: $none\n")
write(io, " Per-number results in 'results2.csv'\n")
end
println("\n Summary statistics:")
@printf(" Full matches: %d\n", full)
@printf(" Prefix and suffix match: %d\n", partial_both)
@printf(" Prefix matches only: %d\n", partial_start)
@printf(" Suffix matches only: %d\n", partial_end)
@printf(" No matches: %d\n", none)
println("\n Statistics saved to 'statistics7.txt'")
println(" Results saved to 'results2.csv'")
return results21_df
end
start_N = 1
stop_N = 10000000
```

m = 2

k = 99999999

run_tests_parallel(start_N, stop_N, m, k)

Code Implementation:

Documentation.

Purpose:

The script tests the behavior of a numeric transformation:

For each integer `N` in the range `[start_N, stop_N]`:

1. Splits the decimal representation of `N` into `m` parts.
2. Multiplies each part by `k`, preserving the original part length (with leading zeros).
3. Concatenates the multiplication results → string `PQ`.
4. Calculates the direct product `N * k` → string `NK`.
5. Removes leading zeros from `PQ` and `NK`.
6. Compares the resulting strings by prefix and suffix.
7. Saves the results to CSV and outputs statistics.

Dependencies:

using Printf

using CSV

using DataFrames

using Base.Threads

using ProgressMeter

Functions:

`split_number_str(N::Integer, m::Integer) → Vector{String}`

Splits the decimal representation of number `N` into `m` substrings of approximately equal length:

- If `N < 10`, pads it with leading zeros up to length `m` before splitting.

- The first `len(N) % m` parts will be longer by 1 character than the others.

- Returns a vector of strings — the number parts from left to right.

`multiply_preserve_length(part::String, k::Integer) → String`

Multiplies the numeric value of string `part` by `k` (using `BigInt`) and returns the result string, padded **with leading zeros** to the length of the original `part` string.

If the result is longer, it is returned without truncation.

`remove_leading_zeros(s::String) → String`

Removes all leading zeros from string `s`.

If the string consists only of zeros — returns `"0"`.

`compare_pq_nk(pq::String, nk::String) → String`

Compares two strings `pq` and `nk`:

- If the strings are identical → `"Full match"`.

- Otherwise, determines the length of the common prefix (match from the start to the first difference) and the common suffix (match from the end to the first difference).

- Returns one of the strings:

  - `"Prefix and suffix match"` — if both matches are non-zero,

  - `"Prefix matches only"` — only prefix,

  - `"Suffix matches only"` — only suffix,

  - `"No match"` — if neither prefix nor suffix match.

`check_algorithm(N::Integer, m::Integer, k::Integer) → NamedTuple`

Performs the full processing cycle for a single number `N`:

- Calculates `NK = string(N * k)`.

- Splits `N` into `m` parts.

- Multiplies each part by `k`, preserving its length.

- Concatenates → `pq_str`.

- Removes leading zeros from `pq_str` and `nk_str`.

- Compares the cleaned strings.

- Returns a named tuple with fields: `N`, `m`, `k`, `parts`,

`multiplied_parts`, `PQ`, `NK`, `result`.

Note: `parts` and `multiplied_parts` are saved as string representations of arrays (via `string(...)`), not as arrays.

`run_tests_parallel(start_N, stop_N, m, k) → DataFrame`

Runs the processing for all `N` from `start_N` to `stop_N` (inclusive):

- Uses multithreading (`Base.Threads`).

- Displays progress via `ProgressMeter`.

- Counts the quantity of each result type using atomic counters.

- Saves all results into a `DataFrame` and writes it to the file `results.csv`.

- Writes summary statistics to `statistics.txt`.

- Outputs statistics to the console.

- Returns the `DataFrame` with results.

Global Parameters (at the end of the script)

start_N = 1

stop_N = 10000000

m = 2

k = 99999999

Tests 10 million numbers, splitting each into 2 parts and multiplying by 99,999,999.

Output Files

- `results.csv` — CSV file with columns:

`N`, `m`, `k`, `parts`, `multiplied_parts`, `PQ`, `NK`, `result`

- `statistics.txt` — text file with statistics:

  - Range of `N`

  - Values of `m` and `k`

  - Count of each match type

Environment Requirements:

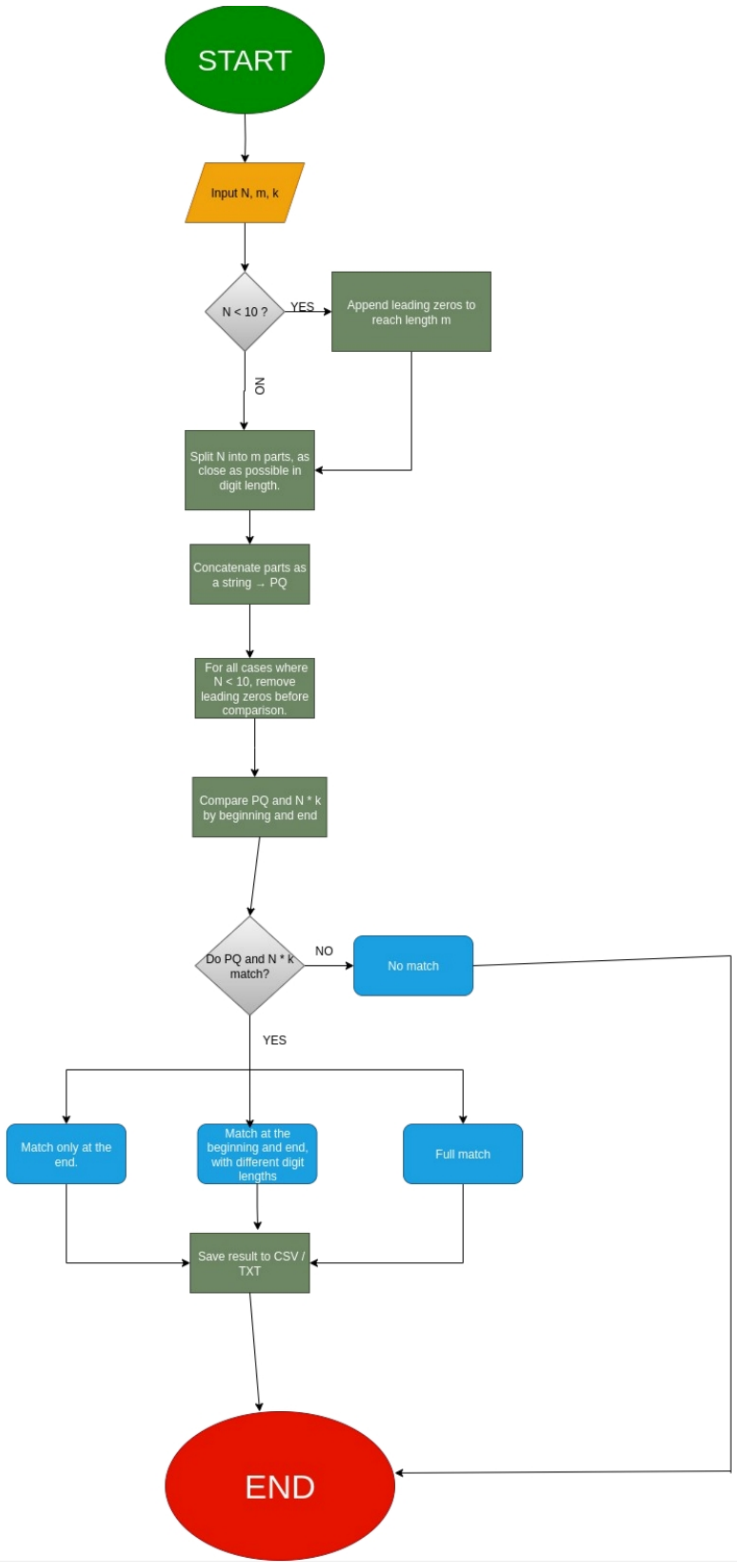
- Julia ≥ 1.6
- Packages: `Printf`, `CSV`, `DataFrames`, `Base.Threads`, `ProgressMeter`
- For multithreading, it is recommended to run with the `--threads=N` flag

License and Authorship:

Code developed in the context of **Structural Number Symmetry (SNS)** research.

Author: Yushchenko Mikhail Yurievich

12. Flowchart


START
Input N, m, k
N < 10 ?
YES
Append leading zeros to reach length m
NO
Split N into m parts, as close as possible in digit length.
Concatenate parts as a string → PQ
For all cases where N < 10, remove leading zeros before comparison.
Compare PQ and N * k by beginning and end
Do PQ and N * k match?
NO
No match
YES
Match only at the end.
Match at the beginning and end, with different digit lengths
Full match
Save result to CSV / TXT
END

13. Proof:

Numbers need to be classified by match type.

For example:

Class F: numbers with a complete match (proven).

Class B: numbers with a match at both beginning and end (not proven/not provable in theory, but has a solid empirical basis!).

Class E: numbers with a match only at the end (proven).

Class S: numbers with a match only at the beginning (refuted).

Class N: numbers with no matches (proven).

Within the SNS algorithm, any natural number N, split into m≥2 parts and subjected to per-block multiplication by a common factor k∈N, generates a result PQ that always belongs to one of the five match classes with the classical product NK=N×k. Below is a strict classification with proofs or justifications for each class.

Notations:

N∈N - original number.

m ≥ 2 - number of parts when splitting.

k∈N - common factor.

N = a1, a2… am — split into non-overlapping decimal blocks (concatenation).

PQ = ($a1 * k$) || ($a2 * k$) || ... || ($am * k$) — result of concatenating multiplied blocks (with length preserved).

NK = N * k - ordinary product.

L = length(am) - number of digits in the last block (L ≥ 1).

Class F - Complete Match

Condition: PQ = NK

Proof:

Complete match is achieved if and only if, during multiplication, no part increases in digit length:

$\forall i$, length ($ai * k$) = length (ai)

If this condition is met, then the positional weights of all blocks in PQ coincide with their contribution to NK, no carries occur between blocks, and the structure is completely preserved.

→Result: PQ = NK → Proven constructively.

Class B - Beginning and End Match

Condition: The prefix and suffix of PQ and NK match, but ≠ PQ = NK.

Status:

Not provable within modern mathematics.

Empirically stable: observed in all cases with various m, k. Theoretically — an open problem.

Significance:

This class constitutes the core of the SNS discovery: despite ignoring carries between parts, the boundaries of the number remain consistent, even when the middle diverges.

Conclusion: → Empirical fact, not following from known theorems.

Class E - Only the End Matches

Condition: Only the least significant (ending) digits of numbers PQ and NK match, while the most significant ones differ.

Proof:

Any natural number N can be represented as:

$N = A * 10 ^ L + am$

where:

am - the last part of the number (last block when splitting),

L - number of digits in this block ($L \geq 1$),

A - everything preceding this block (non-negative integer).

Then the ordinary product:

$NK = N * k = A * k * 10 ^ L + am * k$

Note that the first term $A * k * 10 ^ L$ is divisible by $10 ^ L$, therefore it does not affect the last L digits of the result.

Consequently: $NK \equiv am * k \ (\mathrm{mod} 10 ^ L)$

Now consider PQ - the result of per-block multiplication and concatenation.

The last L digits of PQ are the result of multiplying am * k, written with leading zeros to length L.

This means that:

$PQ \equiv am * k \ (\bmod 10 \ \hat{} \ L)$

From this it immediately follows:

$PQ \equiv NK \ (\bmod 10 \ \hat{} \ L)$

→The last $L \geq 1$ digits always match.

Thus, the match in at least the final digits is not a coincidence but a strict consequence of decimal positional notation.

If the most significant digits differ (which often happens with carries between blocks), then the result belongs to **class E**.

→ **Class E** is possible and provable.

Class S - Only the Beginning Matches

Condition: The prefix (initial digits) of numbers PQ and NK matches, but the suffix (ending digits) differs.

Proof by contradiction:

1. Suppose there exists a number N for which the SNS result PQ and the classical product NK = N * k match only at the beginning, but not at the end.

2. However, as shown in the proof for class E, for any split of number N into parts where the last part has length $L \geq 1$, it always holds: $PQ \equiv NK \ (\bmod 10 \ \hat{} \ L)$. This means the last L digits always match.

3. Consequently, a match at the end is inevitable.

4. But this contradicts the initial assumption that the match is only at the beginning.

→The assumption is false.

Conclusion: **Class S** is logically impossible. No natural number can belong to **class S**.

S = ∅.

Class N - No Matches

Condition: No digits of numbers PQ and NK match — neither at the beginning, nor at the end, nor in the middle.

Proof:

As shown in the analysis of **class E**, the last L digits of the SNS result always match the last L digits of the ordinary product N * k, where L is the number of digits in the last part of N. Since L cannot be less than 1 (the last part always contains at least one digit), at least the last digit always matches. Consequently, a complete absence of matches is impossible.

Conclusion**: Class N** logically cannot exist. The set of numbers belonging to **class N** is empty.

N = ∅.

Final Table

Class	**Possible?**	**Basis**
F	✅ Yes	Constructive condition: block length does not increase.
B	✅ Yes	Empirical phenomenon, theoretically open.
E	✅ Yes	$PQ \equiv NK(\bmod 10^L)$ — strictly proven.
S	❌ No	Contradicts match at the end.
N	❌ No	Contradicts match in at least one digit.

14. License and Contacts:

Author**: Mikhail Yushchenko**

Link: https://github.com/Misha0966

email for contact: misha0966.33@gmail.com

Website: https://structuralnumericalsymmetry.ru

Telegram channel: @structuralnumericalsymmetry

VK group link: https://vk.com/structuralnumericalsymmetry