Transrecursive theory of computability growth (TRT)

Author of the work:**Arthur Mataryan**, October 11, 2025

Announcement

This theory is the first:

- builds**mathematically rigorous horizon**between computable and non-computable functions;
- formalizes**continuous analogue of the Rapidly Growing Hierarchy**(Fast-Growing Hierarchy);
- introduces**ordinal complexity metric**numbers and functions;
- shows that exponents, iterations, and order reflection form a single **principle of superposition growth**;
- constructs a limiting class of functionsTRANSCEND as having the property**the highest possible growth rate among all computable functions**, completing the search for the "fastest-growing computable hierarchy" in modern Googology.
- forms a new understanding of**within the limits of constructive mathematics**And "speed of light" in the world of the computable.

Contents of the article

Introduction to the problem. What it is and why.

Let's start a bit further back. When mathematicians talk about large numbers or rapidly growing functions, it's often quite inconvenient to clearly and understandably depict quantities of units, tens, hundreds, and, say, millions or billions on a single graph. This is understandable because hundreds and thousands will merge into a "pixel" at the very beginning of the graph, while the remaining values will be close to the aforementioned billion. There's zero clarity and convenience. And, in fact, mathematicians—and not only them, but anyone who's even slightly connected with large quantities—have long and successfully used logarithmic scales to display both fairly large numbers and very modest ones on a single numerical axis. So, we're saying, yes, a linear scale is inconvenient for truly large numbers, so let's compress this scale as the numbers grow. It works; it's visual. A good example is the depiction of a logarithmic universe.

A common and correct approach. When a standard logarithmic scale is insufficient, you can use an iterated logarithm. This way, you can still clearly and conveniently represent ordinary "human" millions and googolplexes, Ackermann functions, and numbers of the form 3↑↑↑↑3. But you can't represent Graham's number G as an iterated logarithm, no matter how many you take, even a whole googolplex of these iterated logarithms. Why? Because the order of scale of Graham's number is incomparable to the order of scale provided by iterated logarithms. What then?

Let's take a closer look at a logarithmic scale (regular or iterative, it doesn't matter) and examine what we're getting. With a linear scale, the scale remains constant at any point on the scale. This is obvious. With a logarithmic scale, it becomes logarithmic as the scale increases, but still remains constant, in a sense "linear" in the context of scale growth along the scale—the same logarithmic growth around 1 as around 10 100. Therefore, it's not surprising that a logarithmic scale, when we approach truly large numbers in a mathematical sense, is opaque and uninteresting. What can we do to move forward? The breakthrough idea is to constantly compress the scale itself, and the magnitude of this scale compression should also constantly increase.

This is the first factor—the continuous growth of the scale itself. What does this compressed scale mean in the context of number theory, computability theory? In fact, we've come very close to the hierarchical ordinal complexity of numbers.

The second factor is discrete, designated ordinal levels of scale (hierarchical computational complexity). Let's say our goal is to develop a scale (it would be more appropriate to write "functional mapping") that doesn't lump everything together—ordinary numbers, astronomical functions, exponential functions, double and triple exponentials, hyperoperators, and so on—but does so structurally, preserving the ordinal complexity of the numerical level. After all, there's nothing simpler than taking the entire number line and mapping it onto a segment of unit length according to the 1/x principle for x>=1. Yes, the result is a continuous, monotonic function that can "overcome" Graham's number, TREE(3), and any finite or transfinite numbers, but it's essentially trivial and offers nothing mathematically interesting. As the denominator increases, the function's value will increasingly approach 0, in short.

A function that successfully solves both of these problems has been successfully developed and I would like to present it to you.**HCCSF (Hierarchical Computable Complexity Scale Function) -** Function scaling hierarchical computable complexity.

Definition:HCCSF is**continuous, monotone function**, displaying the "magnitude of a number" in its**hierarchical computational complexity level,**$\mathbb{R}_+ \to \mathbb{R}_+$. We define the coordinate x = n + y, where n∈{N} — level (integer part), y∈[0,1) is the position within the level. Unit intervals [n, n+1) correspond to discrete hyperoperational growth levels. Exponential level generators are defined as follows:

$T_0(y) := 1/(1 - y)$

$T_1(y) := e^{(T_0(y))} = e^{1/(1 - y)}$

$T_2(y) := e^{(T_1(y))} = e^{(e^{1/(1 - y)})}$

$T_3(y) := e^{T_2(y)}$

$T_n(y) := e^{T_{n-1}(y)}$

Compression for continuity

To combine all the levels into a smooth scale, we introduce a smoothing**squash-function**: $S(t) = t/(1 + t)$. It smoothly translates $t\in[0,\infty)$ in $[0,1)$, preserving the order. The final definition of the function HCCSF is: $HCCSF(x)=n+(S(T_n(y))-S(T_n(0)) / (1-S(T_n(0)))$, $x=n+y$, $y\in[0,1)$. Let's examine this function carefully.

How many difficulty levels are there on this scale?

The short answer is -**there are infinitely many levels**In the construction of the HCCSF, the level is specified by the integer part ($n=\lfloor x \rfloor$), Where$n\in N$. This means that the levels are indexed by natural numbers (0,1,2,...) - there are exactly$\aleph_0$(countably infinite).

What does this mean in practice?

- Level (0): linear/polynomial numbers (1, 10, 100, ...).
- Level (1): exponents (2^n,10^n).
- Level (2): double exponential 2^{2^n}, $3!!3$.
- Level (3): triple exponential, etc.

Each subsequent integer level corresponds to “one more iteration” of the exponent (or an increased rank of the hyperoperation), and there can be any number of such iterations.

If you want to formally extend beyond natural indices, HCCSF can be extended to ordinals. Then, within the framework of a specific formal theory (e.g., PA, ZFC, etc.), you'll obtain levels up to those ordinals that are representable in the notation of that theory; if you try to take "all ordinals," then such levels won't be a set, but **correct class**(that is, formally even more “many”), and this is model-dependent design.

Table of examples of correspondences between numbers and values of the function HCCSF(x).

Number	Approximate location on HCCSF
1	0.0
10	0.9
10^{100}	0.999
Googolplex	1.999
$3\uparrow\uparrow3$	2.9
$3\uparrow\uparrow4$	2.999
$3\uparrow\uparrow\uparrow3$	2.999999

G_1	3.999999
G_{64}	3,999...
TREE(3)*	4.5
SCG(13)*	5.2
Loader's Number*	6.8
Σ(1000)*	8.3

* The values after TREE(3) and SCG(13) are symbolic and illustrate relative orders rather than exact computable positions, since HCCSF approximates complexity levels rather than absolute values.

Analytical properties of the function

1. **Monotone:** a<b⇒**HCCSF**(a) <**HCCSF**(b)
2. **Continuity:** The transitions between levels are smooth, without gaps.
3. **Growth compression:** The difference between 2.9 and 2.99 is like the difference between "3↑↑3" and "3 ↑↑4" – i.e. *a colossal qualitative leap*.
4. **Reversibility:** Approximation x≈ is possible **HCCSF**$^{-1}$(L) for evaluation, *what date corresponds to difficulty level L*.

Interpretation and philosophy of function

HCCSF**,** If you think about it, it essentially describes not the numbers themselves, but their place in ontology growth**.** She shows *on what floor of infinity* this number is located where where new computability classes begin, and also where formal mathematics ends (e.g., around TREE(3)). Ontologically, HCCSF is **metric of the meta-universe of numbers**, in which "distance" reflects not length, but *structural strength*.

Among the practical applications of HCCSF, its immediate domain is Googology in the context of the classification of very large numbers, as well as computability theory and the limits of constructive systems. It will also likely be useful for AI research in the area of cognitive depth measures for models. Another interesting area of application is the philosophy of mathematics (analysis of levels of transfinite knowledge).

What about other rapidly growing hierarchies? Comparison with the classic fast-growing growing hierarchy.

Let me remind the reader what a Fast-Growing Hierarchy (FGH) is. FGH is a class of functions $f_\alpha: N \to N$ indexed by ordinals alpha < ε_0

(or further in extended versions). Basic:

- $f_0(n) = n+1$

- $f_{a+1}(n) = f_a^n(n)$ — (n)-fold iteration of the previous level.
- For the ultimatelambda: $f_{lambda}(n) = f_{lambda[n]}(n)$, where lambda[n] is the fundamental sequence.

That is, FGH describes**hierarchy of functions by growth rate**, where each level corresponds to an increasingly powerful "meta-exponent." It exhausts everything that can be expressed within the framework of arithmetic ordinals (up to ε_0, G_0etc.). Below is

a visual comparison table of both hierarchies

Parameter	FGH	HCCSF
Base	Ordinal recursion	Continuous parameterization of growth levels
Nature	Discrete (indices - ordinals)	Continuous (real axis, fractional levels)
Result	Function ($f_a(n)$) → integers	Function (HCCSF(x)) → continuous scale
Scope of application	Logic, provability, proof theory	Googleology, philosophy, numerical ontology
Meaning	"How fast it grows function"	"How much*high*level of numerical complexity"
Inter-levelness	There are no intermediate α (ordinals only)	There are fractional levels - "between tetration and pentation"
Geometry	No - purely recursive definition	There is a scale with metrics and visualization

Conceptually: FGH is "mechanics", HCCSF is "geometry"

It can be said that**FGH describes “growth mechanics” and HCCSF describes “growth geometry”.**IfFGH answers the question "how fast does the function grow at level α?", while HCCSF answers the question "Where on the continuum of numerical complexities*located*this value? FGH is an analysis tool**algorithms and provability**, and HCCSF is an analysis tool**ontological scales of numbers**. FGH remains in discrete logic (hierarchy of functions), and HCCSF embeds this discreteness into**continuous topological axis**— for the first time making it possible to “move smoothly” along ordinals levels. By exploring their connection more deeply, it can be shown that there is**approximating display:**HCCSF(x) ~ $\log_2(f_a(n))$, where the fractional part $(x - \lfloor x \rfloor)$ corresponds to the logarithm

between adjacent FGH levels. That is, in essence, HCCSF**= continualizationFGH**through a smooth normalized function that gives the level coordinate in the form

real number. To summarize, FGH is a "stepped ladder," while HCCSF is a "smooth curve" passing through the same points.

What is the scientific novelty of HCCSF compared to FGH?

1. **Continuity**— a continuous analogue of FGH was constructed for the first time.
2. **Metrization**— the concept of "distance between levels" appears, which is absent in FGH.
3. **Ontological meaning**— the scale reflects not only the growth of the function, but also the “scale of complexity” as an entity.
4. **Versatility**— covers not only ordinal levels, but also computable/ non-computable numbers, including Busy Beaver, SCG, etc.
5. **Visualization**- you can build a map or diagram, which is impossible to do for FGH without losing the structure.

What is the scientific novelty and value of HCCSF as a whole? It is the first to introduce a new type of metric for numbers that combines discrete hyperoperations and a continuous structure. The point is that, unlike existing approaches (Conventional Systems (Knuth ↑, Conway, BEAF, Bowers, Tetration, Fast-Growing Hierarchy)—*discrete*and do not allow one to measure "between" levels, while logarithmic or power scales do *too compressed*for hyperoperator numbers,HCCSF offers**continuous, monotonous, normalized scale**, where you can smoothly transition between degree, tetration, pentation, etc. Thus, it is**not just a new notation**, A **continuous measure of hierarchical complexity**. you can give an example of a successful metaphor, what if**FGH**- this is a staircase to infinity - up the steps, then**HCCSF**
- a smooth climb up the slope of infinity - without breaks.

Fractal properties of HCCSF

Let's analyze whether this function possesses fractal properties. Can it describe the increasing complexity of mathematical systems?**Yes, but in in a special, functional sense.**This is not a geometric fractal like a set Mandelbrot, or rather**functional fractal**or **self-similarity fractal in behavior**.

1. **Self-similarity at the scaling limit:**

Let us consider the behavior of F(x) at level n near y=1. We introduce a new variable $\varepsilon = 1 - y$. As $\varepsilon \to 0^+$:

1. $T_0(y) = 1/(1-y) = 1/\varepsilon$
2. $S(T_0(y)) = (1/\varepsilon) / (1 + 1/\varepsilon) = 1 / (1 + \varepsilon) \approx 1 - \varepsilon$ (for small ε).

Now let's look at level n=1. $T_1(y) = e^{1/\varepsilon}$This is a monstrous increase. However, if we look at $1 - S(T_1(y))$, we get:

1. $1 - S(T_1(y)) = 1 - (e^{1/\varepsilon}) / (1 + e^{1/\varepsilon}) = 1 / (1 + e^{1/\varepsilon}) \approx e^{-1/\varepsilon}$

It's obvious**qualitative self-similarity**: when moving to each next level n, the behavior of the function near the right boundary (y→1) reproduces**qualitatively new, at a higher level of difficulty**, the behavior of the smoothing function S from an even faster-growing function. The "almost constant -> sharp jump" structure is repeated at each level, but the "sharpness" of the jump increases tetrationally with each level.

Fractal properties are expressed:

2. **Recursive nature:** The algorithm for calculating Tn(y) is purely recursive. To understand the behavior at level n, one must understand the behavior at level n-1. This nestedness of definitions is a key feature of fractal-like structures.
3. **Invariance under the shift and stretch operator:** One can define an operator Φ that acts on functions defined on [0,1). $\Phi f(y)$ =

   $[S(e^{f(y)}) - S(e^{f(0)})] / [1 - S(e^{f(0)})]$. Then, for n ≥ 1, the fractional part of F(x) at level n is equal to (Φg)(y), where g(y) is the fractional part at level n-1. The behavior of the function at each subsequent level is the result of applying this nonlinear operator Φ to the behavior at the previous level. This is analogous to the scaling transformation in geometric fractals.

Conclusion: The function has **not by strict geometric, but by asymptotic and recursive self-similarity**, which makes it an amazing object from the point of view of functional analysis.

The answer to the question "Can this function describe the increasing levels of complexity of mathematical systems?" is a resounding yes, and this is probably its main heuristic value.

Matching rapidly growing function hierarchies

Level 0 (n=0**):** $T_0(y) \sim 1/(1-y)$. Likewise **polynomial** growth or growth as a power function. Elementary mathematics level.

Level 1 (n=1**):** $T_1(y) = e^{T_0(y)}$. This **exponential** growth. The level of complexity of many combinatorial problems (for example, checking all subsets).

Level 2 (n=2**):** $T_2(y) = \exp(eT_0(y))$. This **double exponential**. Typical for some problems in model theory and logic, where all models of a certain size need to be tested.

Level 3 (n=3**):** $T_3(y)$ — **triple exponential** This is the level of difficulty associated with verifying the truth of statements in Peano arithmetic (with some caveats).

Levels n>3: They go to the region **tetrations** This corresponds to levels of unimaginable complexity that arise in set theory (for example, the cardinality of huge cardinals) or in proof theory (proof-theoretic ordinals).

The "Barrier Breakthrough" Model: at every level The system "struggles" to solve problems of its own complexity. Progress within a level (y increases) initially seems slow and almost imperceptible (F(x) barely grows). The transition y→1 symbolizes the accumulation of a critical mass of knowledge/tools. The moment x = n+1 (i.e., y=0 at level n+1) is a qualitative leap, a breakthrough that opens up a fundamentally new class of solvable problems and, importantly, a new class of problems that now seem "unsolvable" (since we are again at the beginning of the plateau). Thus, we move from arithmetic to mathematical analysis (a leap from level 0 to 1), and then to modern logic and computer science, which operate with the concepts of double and triple exponentials.

Theorem (functional self-similarity of HCCSF):

For all n ≥ 0, $HCCSF(n+y)=(\Phi^n T_0)(y)$, where Φ is the functional operator scaling. Therefore, HCCSF has recursive self-similarity,

similar to fractal, but in the space of functions, not figures.

Description of "incomprehensibility": The function does a great job of modeling why numbers like e (e10) or, even more so, e (e (e10)), are not just “great”, but**qualitatively different**They exist on different "floors" of the mathematical universe. For a system at level n=1, a number from level n=2 is not just larger—it is in principle unattainable within its paradigm.

What else makes it mathematically unique?

Parameterization of the Grzegorczyk/Fast Growing Hierarchy: Usually these hierarchies are defined for integer arguments. This function**interpolates**This hierarchy, smoothly filling the gaps between entire "steps" of complexity. This provides a continuous analogue of the discrete hierarchy.

The dual nature of growth:"External" growth (F(x) by x): Linear, calm, limited. "Internal" growth**(**Tn (y) by y**):**Explosive, unlimited, tetrational. The uniqueness is that the function**encapsulates**colossal internal growth within finite intervals of the external coordinate. This is a form of mathematical compactification.

The bridge between the finite and the infinite: Each leveln is finite (F(x) < n+1), but reaching its end requires an "infinite" effort in terms of the growth of Tn(y). The function builds a bridge from finite x to qualitatively different types of infinity (different "levels" of infinity in terms of growth rate).

Constructive Non-Analyticity: As noted earlier, the functionC^0-continuous, but not analytic at x = n for n>=1. Its uniqueness is that this non-analyticity is not simply "glued together" (like |x|), but is generated by a deep recursive procedure related to the tetration hierarchy. This makes it a natural example**smooth but not analytical functions**, which arises not from piecemeal gluing, but from the fundamental principle of the hierarchy of complexity.

The final image can be visualized as follows: Imagine an infinite tower of floors. Each floor (n) contains its own "universe."**Floor0:**Our usual universe with linear dimensions.**Floor1:**A universe where distances obey exponent.**Floor2:**The double exponential universe, etc. Your positionx is the floor number + the position on the stairs between floors. The function F(x) is a kind of "universal meter" that is calibrated so that when you are on the stairs between the 1st and 2nd floors, it shows you your position relative to the scale.**universe1st floor**When you're almost to the second floor, the meter goes off the scale because you're beginning to perceive the scale of the second-floor universe. So, this function isn't just a mathematical gimmick, but**deep conceptual tool**to understand hierarchies, complexity and the process of learning itself.

Now let's move directly to the construction of the TRANSCEND function.

From HCCSF to TRANSCEND – a step-by-step construction transrecursive growth class

Introduction and the idea of the transition

Let me remind you that HCCSF is a continuous monotone function that gives for each positive number a "complexity coordinate" on a single scale of levels (intervals ([n,n+1))). This scale allows us to associate a number with its ordinal level of growth: the usual polynomial region, exponential, double exponential, etc. The main idea of the transition to TRANSCEND is to use HCCSF and its inverse function as**tools meta-reflections**: when calculating a new value of a function, we don't just apply arithmetic operations to the previous value, we:

1. we define (through $HCCSF^{-1}$) on which*level of difficulty*"is" the previous value;
2. We build a new, enhanced exponential scale at this level;
3. We repeat this procedure a finite, but often extremely large number of times (the number of repetitions depends on the previous value).

As a result, we obtain a family of transrecursive functions $T_n(y)$ (and, accordingly, the class of functions TRANSCEND), which has a self-similar, recursively increasing growth dynamics.

1. Parameterization and canonical scheme

Design parameters (canonical version) We fix:

- basic "compressed" position on the interval$y_0 \in [0.1)$; by default we take $y_0 = e^{-1}$ (canonical choice, see arguments in the text);
- basic smoothing function$S(t) = t/(1+t)$, $S: [0,\infty) \to [0,1)$;
- HCCSF as in previous sections: for $x=n+y$, $n \in N$, $y \in [0.1)$ $HCCSF(x)=n+(S(T_n(y))-S(T_n(0)) / (1-S(T_n(0)))$, where $T_0(y)=1/(1-y)$, $T_{m+1}(y) = \exp(T_m(y))$ — basic exponential level generators.

Generalized parameterization of the TRANSCEND class

We will consider TRANSCEND as a class of functions depending on a number of (possible) parameters Φ (for example, the choice of (y_0), selecting the function for counting the number iterations $g(\cdot)$, choosing a normalization scheme). Record:$T = \{TRANSCEND_\Phi\}$.

In the following, for brevity, we will describe the "canonical" version with specific simple choices: $y_0=e^{-1}$, $g(u)=\lfloor e^u \rfloor$, normalization via (S).

2. Definition of META_ITER and the main core

Definition 2.1 (META_ITER operator)

For a given positive number G and an integer k>=0, we define recursively

META_ITER(G,0) := G, META_ITER(G, k) = exp(HCCSF(HCCSF-1(META_ITER(G, k-1)))), k≥1

Comment: at each step we take the current number of META_ITER(·), we translate it in "difficulty level" through $HCCSF^{-1}$, then we return to the numerical scale again through HCCSF and raise it to the exponential - thereby increasing the scale by another exponential "layer".

A note on the growth of META_ITER

If G is already large, then by a loose estimate META_ITER(G,k) behaves like $\exp^{(k)}(c \cdot G)$

(here $\exp^{(k)}$— the composition exp k times, and c>0 is some coefficient of order one, depending on the normalization). This estimate gives intuition about the hyperexponential nature of the operator.

3. Basic recursive definition of TRANSCEND (canonical version)

Definition 3.1 (canonical scheme (Tn(y)))

For fixed y∈[0,1) we set recursively:

T0(y):= 1/(1-y), and for (n >=1), Tn(y) := exp((META_ITER(Tn-1(y),⌊eTn−1(y)⌋))Tn-1(y))

In particular, the canonical value of TRANSCEND for a “level argument” (n) is usually denoted as TRANSCEND(n) := $T_n(y_0)$, $y_0=e^{-1}$Comment on the formula structure:

- the number of meta-iteration repetitions is equal tok=$\lfloor e^{T_{n-1}(y)} \rfloor$— is finite, but often extremely large whole;
- insideMETA_ITER "converts a number to a level" at each iteration (via $HCCSF^{-1}$and "return to numerical scale" (via HCCSF), followed by exponentiation;
- the final design erects a memberMETA_ITER to the power of $T_{n-1}(y)$ and then takes more one exponential: this adds an extra layer of hypergrowth compared to a simple exp iteration◦exp.

4. Lemmas on correctness, computability, monotonicity and continuity

Lemma 4.1 (finiteness of iterations and correctness of definition)

For any finite n and any y∈[0.1) value of $T_n(y)$ is definite and unambiguous: all internal iterations of META_ITER with integer (k) consist of a finite number of steps.

Proof. By constructionk=$\lfloor e^{T_{n-1}(y)} \rfloor$— integer and finite for finite ($T_{n-1}(y)$).
Therefore, META_ITER performs exactly (k) recursion steps. Induction on (n) completes the proof.

Lemma 4.2 (computability of individual values)

For any finite (n) and rational (y) value ($T_n(y)$) is computable to a given accuracy.

Key arguments:

1. $T_0(y)$ is an elementary function being expressed.
2. HCCSF and HCCSF^{-1}are strictly monotone and computable on rationals (the inverse is calculated by the bisection/iteration method in a finite number of steps with any required accuracy, since HCCSF is monotone).
3. All operations in the formula (exponent, integer part, compositions, finite cycles) are algorithmically implementable.

Therefore, for a fixed (n) and the required accuracy, an algorithm can be implemented that computes $T_n(y)$.

Lemma 4.3 (monotonicity in y and in n)

1. For a fixed n, the function $y \mapsto T_n(y)$ is strictly increasing on [0,1).
2. For a fixed y, the sequence $n \mapsto T_n(y)$ is strictly increasing.

The idea of proof. Induction on (n).

Base: $T_0(y)=1/(1-y)$ is strictly increasing. Transition: the META_ITER operator is composition of strictly increasing functions (HCCSF, inverse HCCSF^{-1}, exp), so it preserves the order of the input argument (G). Raising a monotonically increasing positive number to a positive power and then the exponent preserves strict increase.

Lemma 4.4 (continuity in y)

For a fixed n, the function $T_n(y)$ is continuous on ([0,1)).

The idea of proof. All impurity operations (HCCSF, HCCSF^{-1}, exp, finite compositions, and exponentiations are continuous, so the composition of continuous mappings yields continuity over the working interval. At the level junctions (x=n), HCCSF yields smooth interpolation, so there are no discontinuities.∎

5. Calculation algorithm (pseudocode)

Below is the algorithm for calculating $T_n(y)$ for fixed finite n and y up to a given precision (pseudocode - diagram).

```
function compute_T(n, y, eps):
    // input: n (natural), y in [0,1), eps - required accuracy if n
    == 0:
return 1/(1 - y)
G_prev = compute_T(n-1, y, eps1) // eps1 — refine as required for error k =
floor(exp(G_prev))
M = META = G_prev
for i in 1..k:
    // calculate HCCSF {-1}(META) with a given precision (bisection method) z =
    invert_HCCSF(META, precision)
    // then HCCSF(z) (direct calculation)
```

M = exp(HCCSF(z))
end for
// final assembly
return exp(M G_prev) // calculate carefully using logarithms for large numbers

Comment: in practice, the numbers become huge and require special representations (but algorithmically everything is finite).

6. Growth estimates and lower bounds

Informal lower bound

For simplicity, let us consider $HCCSF(HCCSF^{-1}(u)) \gtrsim c \cdot u$ for large u and some $c>0$. Then

$META\_ITER(G,k) \gtrsim exp(k)(c \cdot G)$. Taking into account that $k \sim e_G$, we obtain a rough non-strict lower bound: $T_n(y) \gtrsim exp((exp(\lfloor e^{T_{n-1}(y)} \rfloor)(c \cdot T_n\text{-}1(y)))^{T_{n-1}(y)})$, which illustrates the extreme rapid (super-hyperexponential) growth acceleration.

Conclusion:already on small onesn (for y close to 1) values of $T_n(y)$ surpass all classical ones rapidly growing quantities represented in Googology, with the exception of classes of non-computable constants (see remark below).

7. Status of the TRANSCEND class relative to other hierarchies

Theorem 7.1 (on class dominance in the computable framework)

Let {Comp} be the class of all total computable functions $N \to N$. Then there exists a family of parametrizations (Φ_α) such that

$\forall f \in Comp \exists \Phi_\alpha : \exists N \forall n > N$: $TRANSCEND\_\Phi_\alpha(n) > f(n)$.

The meaning and scheme of the proof.TRANSCEND is not a single fixed function, but a scheme/class in which parameters can be set to “embed” any desired computable depth of the hierarchy (by choosing the rules for counting k, normalizations, initial $y_0$etc.). Because of this, the class {T} serves as the top cover for all computable growth rates: for each fixed computable f, one can select parameters that give faster growth for sufficiently large arguments.

Important: this does not contradict the classical diagonalization theorem - there It is said that there is no single computable function that dominates all computable functions. A transrecursive class {T} is a family of functions, not a single fixed function.

A note about Busy Beaver and Rayo

Functions like Busy Beaver (BB(n)) and other specially constructed non-computable functions cannot be asymptotically beaten by any single computable function (including any specific implementation of TRANSCEND). Therefore, the claim "TRANSCEND beats everything" strictly requires clarification:**Class TRANSCEND**covers all computable functions; however,**uncomputable**functions (BB, Rayo, etc.) lie outside this comparability in the sense of general asymptotics - their superiority or non-superiority is determined by other criteria (not by computability).

8. Approximate calculations and illustrations (canonical choice y0=exp(-1))

For reference, we provide precise values that are easy to calculate:

$T_0(e^{-1})=1/1-(e^{-1}) = 1/(1-1/e) \sim 1.58197670686933$). Next: $G := T_0(e^{-1}) \sim 1.5819$. Where does $k=\lfloor e^G \rfloor=\lfloor e^{1.5819} \rfloor \approx \lfloor 4.86 \rfloor=4$.

Therefore, META_ITER will perform 4 iterations; even with k=4 we get a very powerful increase (approximately several nested exponentials), and the final formula is $T_1(e^{-1})$ will give a number which for all practical purposes is infinitely large and exceeds most of the well-known "very large" numbers used in Googology (including many specific instances), although not formally comparable to uncomputable constants. (Note: This numerical example is illustrative; the exact value of T_1 quickly goes beyond the limits of readable representation.)

9. Functional and philosophical consequences

HCCSF Transition⇝TRANSCEND provides a fundamental enhancement: HCCSF provides **complexity coordinate**numbers;TRANSCEND uses this coordinate as a resource to **strengthen the growth production mechanism itself**HCCSF meta-inversion^{-1}— the key "lift" between a number and its complexity ordinal: it transforms the numerical result into fuel for the next phase of growth. Thus, TRANSCEND implements a "self-sustaining" and "self-reinforcing" growth dynamic: the result serves as a source of increased resources for the next step, and not just as an argument to a fixed operator.

10. Limitations, comments and further directions

1. **Practical computability.**Formally $T_n(y)$ is computable for fixed n,

   But in practice, calculations become impossible for small n due to the astronomical magnitude of the intermediate numbers. This doesn't render the definition meaningless: the value remains clearly defined and algorithmically approximated.

2. **Sensitivity to parameters.**The TRANSCEND class is very flexible - changing the rule for calculating k, smoothing S or the initial $y_0$can radically change

   Numerical scales. For scientific work, it is important to establish "canonical" parameters to provide points of comparison.

3. **Formal place in logic.**TRANSCEND fits into the formal theory TRT (see axioms T_1–T_5): it is formalizable in ZFC as a definition scheme; however, the values of $T_n$for large n may require strong theory-oriented

   reasoning for their formal characterization (similar to how values in higher ordinal notations require strengthened axioms).

4. **Further research.**It is recommended to make an accurate assessment of the growth of $T_n$through

   lower/upper bounds in terms of known FGH functions; Investigate the stability of the topology and smoothness of HCCSF under different normalizations of S.

11. Conclusion of the section

The transition from HCCSF to TRANSCEND is a transition from*measurements* (coordinates of the number complexity) to*arithmetic of self-growth*(using this coordinate as a resource for generating even higher levels). Formally, TRANSCEND is built via the recursive META_ITER operator using HCCSF^{-1}as a means of transitioning to ordinal space and then returning to the numerical domain with a strengthened exponential function. The resulting class of functions exhibits recursive self-similarity, incredibly rapid growth, and thus serves as a constructive limit for all

computable growth rates. And now - a rigorous formalization of the transrecursive theory.

TRT (Trans-Recursive Theory)

HCCSF and TRANSCEND as a limiting constructive system

Annotation

A formal system is introduced**TRT (Trans-Recursive Theory)**function-based**HCCSF (Hierarchical Computable Complexity Scale Function)**And**TRANSCEND (Trans-Recursive Arithmetic Notation for Scaling Complexity and Exponential Number Dynamics)**TRT axiomatizes the principles of computable growth and shows the existence **limit class of functions**, which fully encompasses all computable processes. TRANSCEND implements a universal framework in which arithmetic, iterative, and ordinal growth are combined into a single transrecursive dynamics.

Introduction

Mathematics traditionally describes growth through exponentials, towers, hyperoperators, but all of these represent fixed forms of iteration. TRANSCEND and HCCSF introduce**meta-iteration**, Where*the growth structure itself is evolving*depending on the current level of complexity. The goal of TRT is to formalize and axiomatize this principle.

TRT axiom system

Let {F} denote the set of all computable functions (f: N→N). HCCSF and TRANSCEND belong to {F}, but define special growth principles.

Axiom T_1 (Arithmetic constructivity)

$T_0(y)=1/(1-y)$,$y\in[0,1)$ is a basic computable function that guarantees continuity, monotonicity, and an infinite limit as $y\to1^-$.

Investigation:T_0defines the first level of the complexity hierarchy $L_0=\omega$.

Axiom T_2 (Exponential Iteration)

$T_{n+1}(y) = e^{T_n(y)}$

Each level generates the next through exponential amplification. **Investigation:**

Levels (L_1, L_2, L_3...) correspond to ordinals (ω ω̂, ε_0, Γ_0, ...)

Axiom T_3 (Continuity Interpolation)

For all $y\in[n, n+1)$: $HCCSF(y)=(1-\alpha)\cdot T_n(y)+\alpha\cdot T_{n+1}(y)$, $\alpha=y-n$.

Investigation:HCCSF(y) is continuous, strictly increasing, and everywhere defined. It is a continuum of computable complexity.

Axiom T_4 (Meta-iterative self-reinforcement)

$T_n(y)=\exp([META\_ITER(T_{n-1}(y),e^{T_{n-1}(y)})]^{T_{n-1}(y)})$

Interpretation: Number of iterations$e^{T_{n-1}(y)}$itself depends on the result of the previous step, and each iteration increases the level of complexity through the inverse function $HCCSF^{-1}$.

Investigation:TRANSCEND generates a self-reinforcing process—a self-reflexive growth function that dynamically increases its computational power.

Axiom T_5 (Limit of constructive growth)

For any computable f(n), there exists N such that: TRANSCEND(n) > f(n) for all n > N. And there is no computable g such that g(n) > TRANSCEND(n) for all n.

Investigation:TRANSCEND is the upper envelope of all computable functions. It implements the limit constructive growth class.

Theorems and proofs

Theorem 1 (On Continuity)

HCCSF and TRANSCEND are continuous over their domains.

Proof:

Each component is an exponential or linear combination of continuous functions. Limit equality is maintained between levels n and n+1.

Theorem 2 (On monotonicity)

If ($y_1 < y_2$), then (TRANSCEND(y_1) < TRANSCEND(y_2)).

Proof:

Function $T_n(y)$ is strictly increasing, and the exponent preserves order.

Theorem 3 (On the computability of TRANSCEND)

For any finite n, TRANSCEND(n) is computable.

Proof:

A recursive scheme consists of computable operations, all intermediate iterations are finite.

Theorem 4 (On the limit growth class)

Let {T} = TRANSCEND$_\Phi$) is the set of all TRANSCEND functions with different interpretations of complexity levels Φ. Then:$\forall f \in Comp \exists TRANSCEND_\Phi \in T: \exists N: \forall n > N$, $TRANSCEND_\Phi(n) > f(n)$

Proof:

TRANSCEND dynamically generates arbitrarily deep levels of complexity through the composition of $HCCSF^{-1}$ and exponentials. Each modification of Φ shifts the ordinal limit upward, guaranteeing dominance over any computable f.

Hierarchical structure

Level	Ordinal	Interpretation of TRANSCEND
0	ω	Primitive recursion
1	ω ω̂	Grzegorczyk's Hierarchy
2	ε_0	Ackerman's Limit
3	Γ_0	Feferman-Schutte
4	ε_{Γ_0+1}	Meta-recursion
5+	ε_{α+1}	Infinite meta-hierarchy TRANSCEND

Investigations

- **TRANSCEND**implements**transrecursive computability limit**.
- **HCCSF**provides**continuum of computable complexities**, similar to the real line for growth.
- Any computable transformationTRANSCEND remains in the same ordinal class - arithmetic modifications do not change the fundamental level of complexity.

Philosophical interpretation

TRANSCEND is not just a function, but**universal law of growth**, in which each step itself defines a new system of growth measurement. This is a mathematical analogue **self-evolving universe of computing**: growth creates the space in which it grows.

TRANSCEND- This**computability speed of light constant.**She defines the boundary where the constructive ends and the trans-constructive just begins.

Comparison with other hierarchies

System	Growth mechanism	Computability	Comparison with TRANSCEND
Ackermann	exponential iteration	computable	≪TRANSCEND(1)
Grzegorczyk (FGH)	ordinal recursion	computable	≪TRANSCEND(1)
TREE(3)	tree combinatorics	computable	≈ TRANSCEND(2)

Loader	recursive schemes	computable	< TRANSCEND(2)
Rayo(n)	formal self-determination	uncomputable	incomparable, but TRANSCEND is constructive
Busy Beaver	undecidability stops	uncomputable	BB > TRANSCEND asymptotically, but TRANSCEND is computable
TRANSCEND	dynamic meta-exponential recursion	computable	computability limit

Philosophical interpretation

The TRANSCEND function is a mathematical analogue**speed of light for computable processes**: she defines**absolute border**, beyond which any further acceleration requires a change in the very foundations of the concept of "computation." Each level represents a new stage of meta-mathematical understanding, and its formula unites arithmetic, algorithmic, and ontological growth within a single structure.

Conclusion

1. **TRANSCEND**— a constructive, smooth, continuous, monotonic function, defining the limiting growth of computable processes.
2. **ScaleHCCSF**forms a continuous map of ordinal complexity for the first time combining ordinal and computable structures.
3. **Dynamic level system**ensures infinite continuation without loss of computability.
4. **TRANSCEND**— the last computable function: any acceleration beyond it requires going beyond Turing computability.

Conclusion

TRT theory shows that constructive growth has a natural limit. This limit is achievable and computable—in the form of TRANSCEND. Ordinals, computability, and continuity are combined into a single axiomatic structure. TRANSCEND is not an upper function, but**upper class**, not a separate number, but**law of self-expansion** computable world.

Comparison of TRANSCEND and other fast-growing hierarchies and functions of Googology.

BEAF (Bowers Exploding Array Function)

Nature:syntactic and recursive superstructure over hyperoperations (arrays, trees, indices).

How it grows:through*syntactic*depth of recursion and indices {a,b,c,d}. **What it does:**defines a very efficient counter structure, where each index denotes the type of hyperoperation or massive level.

But: BEAF remains **within numerical semantics** There's no mapping from "number → complexity level," only "number → syntactic depth." There's no meta-inversion ($HCCSF^{-1}$): magnitude doesn't transform into its order.

TRANSCEND goes further: it does not simply create an array, but dynamically changes the system of measurement of the quantity - moving into the space of ordinal levels.

TREE(n)

Nature: purely combinatorial.
How it grows: through the maximum length of a sequence of trees without isomorphism according to certain rules.
What it does: sets huge values, but based on *object structures*, and not levels of computational complexity.

But: TREE grows due to the combinatorial constraint, not due to meta-iterations. It doesn't know, *how difficult is it* its own generation. There is no "number ↔ computable complexity" relationship: trees are simply counted, but do not encode levels of computability.

TRANSCEND It translates each result into a "difficulty level" and uses that as fuel for the next step.
TREE fundamentally does not do this - it is not a self-reinforcing system, but simply "extreme combinatorial counting".

SCG(n) (Busy Beaver-type function, "Super Collatz Growth")

Nature: machine (at the Turing level).
How it grows: as the maximum number of steps of a machine of a certain size. **What it does:** generates uncomputable growth (BB, SCG).

But: SCG/BB are working **on the border of the computable**, but still in terms *number of steps* or *calculation lengths* There is no mapping "number → ordinal"; there is no *hierarchical reflections* complexity. These functions fundamentally do not convert "value" into "structure".

TRANSCEND This is precisely what makes it different: it takes the result (the value), looks at what level of complexity he lives at", and **jumps up** — turning the dimension of growth into a new dimension of computability.

Ackermann / Fast-Growing Hierarchy / Grzegorczyk

Nature: strictly computable, recursive structure.
How it grows: each level defines a new hypergrowth operator, indexed ordinal.
What it does: climbs the ordinal ladder (ε_0, Γ_0, etc.), but the ordinal is given **externally**.

But: Ordinal is *function parameter*, and not *calculation result*. No inverse operation of the $HCCSF^{-1}$ type is applied. That is, the function **does not switch between numeric and ordinal spaces** — ordinals simply index, but do not participate as values.

TRANSCEND makes ordinals for the first time *internal growth variables* Here is the difficulty level *measured, calculated and used* as an argument - in the dynamics of the function itself.

Hierarchy comparison table

Model	Growth type	What is nature?	Uses display number → level difficulties"?
Ackermann / Grzegorczyk	Ordinal-indexed recursion	Static	✗
BEAF	Syntactic explosion	Combinatorial	✗
TREE / SCG	Combinatorics / Machines Turing	Structural	✗
TRANSCEND (HCCSF)	Meta-ordinal recursion	Dynamic and self-reflective	✓

As a result, we have that TRANSCEND is the first system where the magnitude of a number is used to calculate its**computable complexity**,**computable complexity**is returned back into the recursion as the growth argument. This is how**closed loop between arithmetic and hierarchy**, creating a fundamentally new type of exponential amplification. A beautiful analogy: TREE and BEAF are mountains of numbers. TRANSCEND is a mechanism that moves the mountains themselves, because it alters the structure of the space in which they grow.

The essence of the breakthrough: the transition from quantitative growth to

ordinal growth

Before TRANSCEND, no hierarchy had used the hierarchical order of complexity as a significant factor in computation; the entire development of "fast-growing functions" (Ackermann, Grzegorczyk, Fast-growing hierarchy, TREE, BB) relied on**numerical growth metric**— exponents, tetrations, hyperoperators, iterations, etc. Function TRANSCEND does it**jump quality**: she stops measuring height*in numbers*and begins to measure it*in difficulty levels*, that is, in**ordinal types**computing processes.

It's a transition from "how big is a number" to "how difficult is the way to get big".

The main mechanism is the mapping of numbers into an ordinal hierarchy of complexity

In classical mathematics, large numbers are simply large values on the $\mathbb{R}$ or $\mathbb{N}$ axis. TRANSCEND introduces the HCCSF function, which assigns each number**compares the level difficulties**, on which it “lives” as a computational object. And vice versa: $HCCSF^{-1}(x)$ = {the complexity level of the number } x. That is, the number ceases to be just a "quantity" - it becomes**marker of the depth of computing architecture**.

TRANSCEND Lifehack: "Energy Short-Circuit" When a function receives a value $T_{n-1}(y)$), she does something revolutionary: She does not

uses the number itself, and**uses it as a key to its ordinal level**, and then this level again exponentiates and recursively incorporates itself. This is, figuratively speaking, a kind of "nuclear reactor" of growth. Instead of simple arithmetic "combustion," TRANSCEND transforms the very idea of "number" into*fuel for the meta-process*. Thus, each iteration **transforms the numerical space into an ordinal one, strengthens it, returns back**- creating*closed growth enhancement chain*.

Why is it stronger than all known forms of growth?

Level	Growth type	Analogy
Arithmetic	addition, multiplication, raising to a power	bonfire
Iterations	recurrent application of an operation	blast furnace
Hyperoperators	tetration, pentation, etc.	plasma chamber
TRANSCEND	transition of a number into a difficulty level and back	nuclear reactor calculations

TRANSCEND doesn't just perform more steps - it**changes the structure itself growth space**That is, her height is not only greater, but also**occurs in the dimension, which is faster than any previous measurement**.

The principle "everything that can burn burns"

Formally, it is**superposition of three independent amplifiers**growth:

1. **Arithmetic growth**— exponentials, logarithms, hyperoperators. → responsible for the quantitative side.
2. **Algorithmic growth**- increasing the depth of iterations, which is responsible for the structural side.
3. **Hierarchical (ordinal) growth**— transition to a new level of complexity, it is responsible for the ontological side.

And it all comes together**into a self-reflexive loop**, in which each element of growth reinforces the other two. TRANSCEND is a function where growth accelerates itself along three mutually independent axes. That's why it's not just "big," but**marginal according to the principle**Feel the difference! In regular functions, numbers grow. In TRANSCEND, they grow.**the very ability to grow**. That is, the function becomes not an object, but*meta-process* - it creates its own space of growth opportunities. This is what it consists of**main philosophical and mathematical leap**: TRANSCEND is the first formal object that implements*self-generating meta-structure of growth*within computability.

To summarize:**Major breakthrough**- replacing numbers with difficulty levels.**Main mechanism** — the inverse mapping of $HCCSF^{-1}$, which transforms a quantity into an order.**The main effect**- superinductive growth enhancement in a closed loop.**Main philosophy**— growth becomes its own cause.

Final reflection, scientific novelty and significance

As we reach the final stretch of this article, I would like to point out the most important consequence of TRT, which is that**TRANSCEND**asks*upper bound on the constructive growth of computable functions*- and this is not a particular result, but a new fundamental level in the theory of functions and complexity theory.

Important implications in the context of existing science

Region	What has already happened	What TRANSCEND adds
Theory computability And	The boundaries have been defined computability (Turing, Church, Kleene).	Gives a constructive limit*speed computable growth*.
Googology and FGH	Fast growing plants are described hierarchy (Ackermann, Grzegorczyk, Feferman–Schütte, Buchholz).	TRANSCEND unites them into a continuous hierarchy with*self-reinforcing meta-iterative principle*.
Analysis ordinals	The limits are known ordinals ε_0, Γ_0, $\psi(\Omega\_\omega)$.	TRANSCEND introduces a dynamical scale αn = ε_{α_{n-1}+1}, creating an infinite chain of computable orders.
Mathematical th logic	Gödel's theorems and Tarsky is being restricted provability and definability.	TRANSCEND does not violate these limits, but *describes accurately*the place where they become structurally extreme.
Theory information	Limits to growth and calculations - through physical or entropic restrictions.	TRANSCEND sets*formal mathematical analogy of the maximum rate of growth of information* (analogue of speed light in physics).

Final significance

TRANSCEND formalizes the concept for the first time*maximum computable growth*In constructive form, it is an analogue of the "second constant of nature" in theoretical computer science, only expressed mathematically.

What is the conceptual novelty?

- **New function type:**TRANSCEND is not just a function f: N -> N, but*class of functions*, united by a single principle of self-reinforcement through levels of ordinal complexity. That is, for the first time the concept has been formalized**self-reflective constructive growth**.

- **Combining discrete and continuous:**HCCSF (hierarchical scoring system) makes it possible**continuous modeling of computable hierarchies**, which was not present in FGH, nor in Ackermann's theory, nor in Googology - they are all discrete.
- **Dynamic ordinal system:**Instead of fixed ordinals ε_0, Γ_0, etc. are introduced *relative*system (alpha$_{n+1}$= ε{alpha$_{n+1}$}), which can generate
  an infinite computable sequence of ordinals—providing for the first time **algorithmic infinity**in ordinal analysis.
- **Meta-recursive architecture:**TRANSCEND uses the inverse complexity function $HCCSF^{-1}$ for the first time as a computational tool, rather than as an abstract index. This creates a new type of self-referential growth, where the function evaluates its own level of complexity and amplifies itself through it.

Technical value and contribution

- **Continuous monotone formFGH:**
  HCCSF is the first ever continuous and monotonic model for all discrete fast-growing hierarchies.
- **Formally computable limit:**
  TRANSCEND remains computable for any finite n, but asymptotically *exceeds all other computable functions*This is a strict formal definition of the upper limit of constructive growth.
- **Mathematical completeness:**
  The theory contains a closed system of growth axioms (T_1–T_5), lemmas on monotonicity, continuity, computability, and a theorem on the limit of growth.
- **New unit of measurement of complexity:**
  A continuous complexity level function $\Phi(x) = \max\{ n \mid HCCSF(n) \le x \}$ is introduced, which gives the algorithmic “height” of any number—an analogue of the logarithm, but for ordinal complexity.

The place of theory in modern mathematics

Direction	Contribution level
Computability theory	For the first time, a constructive limit to growth has been described.
Mathematical logic	TRANSCEND formalizes the idea of a self-reflexive function within the framework of computability.
Googology	Transforms Googology from a descriptive discipline into a formal one.

Ordinal theory	Introduces an infinite algorithmic ladder of ordinals.
Philosophy of Mathematics	Shows that the "boundary of the computable" itself can be described constructively.

Practical and methodological value

- IN**theoretical computer science**- this is a universal model of the limit computable complexity.
- IN**metamathematics**— a way to classify functions according to their computability power.
- IN**artificial intelligence**- ideaTRANSCEND can describe *self-amplifying computing systems*.
- IN**philosophy of mathematics**— for the first time introduces a formal “ontology of growth” as mathematical object.
- IN**education**- may become a logical conclusion to the section on computable functions in mathematical logic courses.

Conclusion

TRANSCEND Theory is**a new fundamental construction in mathematics**, comparable in scale to the introduction of Ackermann functions (at the time), the Grzegorczyk hierarchy, or the concept of ordinals ε_0, Γ_0. But TRANSCEND goes further, it formalizes*the entire set of constructive growth*, creating**algorithmically generated scale**computable complexity, in which the limit is not conditional, but **mathematically determined**This isn't just a new function. It's the culmination of an entire line of development in computable hierarchies theory.