Consistent Ultrafinitist Logic

Michał J. Gajda
Migamake Pte Ltd, Singapore

Abstract

Ultrafinitism postulates that we can only compute on relatively short objects, and numbers beyond a certain value are not available. This approach would also forbid many forms of infinitary reasoning and allow removing certain paradoxes stemming from enumeration theorems. For a computational application of ultrafinitist logic, we need more than a proof system, but a logical framework to express both proofs, programs, and theorems in a single framework. We present its inference rules, reduction relation, and self-encoding to allow direct proving of the properties of ultrafinitist logic within itself. We also provide a justification why it can express all bounded Turing programs, and thus serve as a "logic of computability".

2012 ACM Subject Classification Theory of computation → Constructive mathematics

Keywords and phrases ultrafinitism, bounded Turing completeness, logic of computability, decidable logic, explicit complexity, strict finitism

Digital Object Identifier 10.4230/LIPIcs.TYPES.2023.5

Related Version Presented on TYPES2022, CiE2022, LAP2021, CLMPST2023, CLA2022
Preprint: `https://arxiv.org/abs/2106.13309`

Acknowledgements We thank Vendran Čačić, Seth Chaiken, Bhupinder Singh Anand, Orestis Melkonian, Anton Setzer for their invaluable feedback. Thanks to Daniel Schwartz for sparkling my interest in Kleene normal form.

Background

Ultrafinitism [42, 61, 79, 22, 47] postulates that we can only reason and compute relatively short objects[1]

Tighter limit can be established using petahertz frequency [9] as a quantum limit for light-based systems giving 10^{33} serial cycles during the lifetime of Earth.] [49, 28, 66, 48, 44, 46], and numbers beyond certain value are not available [79, 66]. Some philosophers also question the physical existence of real numbers beyond a certain level of accuracy [24]. This approach would also forbid many forms of infinitary reasoning and allow removing many from paradoxes stemming from a countable enumeration.

However, philosophers[2] still disagree on whether such a ultrafinitist logic could be consistent [17, 51], while constructivist mathematicians claim that *"no satisfactory developments exist"* [74]. We present a proof system based on the Curry-Howard isomorphism [35] and explicit bounds for computational complexity that answers the question.

This approach invalidates logical paradoxes that stem from a profligate use of transfinite reasoning [6, 55, 67], and assures that we only state problems that are decidable by the limit on input size, proof size, and the number of steps. This explicitly excludes phenomena of undecidability by excluding them from our realm of valid statements [73]. Our approach allows to express all Turing Machine programs that are bounded [34] by proof terms of the logic[3].

[1] For example, a computation run by computer the size of Earth within the lifespan of Earth so far. Of the order of 10^{93} as described by [28].
[2] We cite physical and metaphysical arguments from previous work equally.
[3] Up to fixed emulation overhead, see Emulation Complexity below in section 4.

Explicitly bounding computational complexity also prevents a famous *paradox of inference*. This paradox of classical theory of semantic information [4, 19] unjustly labels all mathematical proofs as "trivial information", because it can be inferred from the axioms.

1 Introduction

By *finitism* we understand the mathematical logic that tries to absolve us from transfinite inductions [42]. *Ultrafinitism*[4] goes even further by postulating a definite limit for the complexity of objects that we can compute with [48, 44, 66, 49, 28, 20]. We assume these without committing ourselves to adopt a fixed number as a limit.

In order to permit only *ultrafinitist* inferences, we postulate *ultraconstructivism*: we permit only *constructive* proofs with a *deadline*. That is constructions that are not just strictly computable, but for which there is a *upper bound* on the amount of computation that is needed to resolve them. That means that we forbid proofs that go for an arbitrarily long time and require *totality* for any proof or computation.

For the sake of generality, we will attach this *deadline* in the form of *bounding function* that takes as arguments *size variables* (*depths of input terms*), and outputs the upper bound on the number of steps that the proof is permitted to make (along with upper bound on the size of the output). *Depths of input terms* are a convenient upper bound on the complexity of normalized proof terms. (Normalized proof terms are those with opportunity for *cut* or β-reduction.)

Our approach is inspired by the *Curry-Howard isomorphism* – the fact that the constructive proofs always correspond to executable programs. It also follows *inverse Curry-Howard isomorphism*: the philosophy that rejects logical inference which do not correspond to programs computable in our universe[5].

The philosophy of *ultraconstructivism* would similarly purport that while transfinitary logics may be consistent, they are correspond to objects *„out-of-this-world'"*[6], since our observable universe is inherently finitary [49].

Contributions

To enumerate chief contributions of this paper:

1. First consistent **ultrafinitist** logic to the knowledge of the author. It allows bounding by any and arbitrarily large computational limit (section 2.1), and consistent reasoning resolving Wang's Paradox[17]. Thus this logic is first formal theory to claim a purely philosophical legacy of ultrafinitism [79, 66, 51, 10].
2. A decidable logic having **meta-theory expressible in itself** (see section 3.4).
3. Clear and comprehensive demonstration that **assumptions of Gödel are too strong** [25, 26] when considering bounded logics. This is because decidability of bounded term can be demonstrated by simple enumeration with a more generous bound. Most proofs are elementary by enumeration. Proof of consistency is done by reduction to widely known intuitionistic logic to make the paper accessible to second year students of computer science or first year graduates in constructive mathematics (section 4).

[4] Also called *strict finitism* by [51].
[5] Given that all formal proofs in mathematically strict formal systems can be considered finite numbers of connected steps in computation or hypercomputation.
[6] Extra-universal.

4. Candidate for a most expressive logic that allows explicitly bounded computable functions. Ideal **logic of computability** must forbid all reasoning about uncomputable, and only allow computable statements (with proofs corresponding to *bounded Turing Machine* programs.) For all bounds that can be computed within the framework, we can also compute the function bounded by these (section 4.3).
5. Placing **ultrafinitism** and **ultraconstructivism** as candidate for realization of *computable foundations for mathematics* programme (discussion in section 6.3).
6. All statements with bounds having a proof without bounds have a proof with bounds too[7] (see section 4.6 theorem 9).

We will further abbreviate the "Consistent Ultra-Finitist Logic" proposed in here as "UFL" when speaking about higher order variant (with dependent Π,Σ types for quantification).

2 Syntax and inference

Due to size bounds and clarity of this paper, we first introduce propositional ultrafinitist logic, and then describe interpretation of universal quantifier in a separate section.

2.1 Bounds

We express bounds as polyvariate functions of the natural numbers, called *sizes*. These explicitly bound our proofs, depending on the size of input terms. While subtraction within natural domain is permitted, only positive results of computation are permitted. All bound functions are increasing with respect to all arguments (monotonic).

The bounds[8] will be standing on one of two roles: as an upper bound on the proof complexity, and there we will use symbol α as a placeholder, or to state an upper bound on the depth of the normal form of the proof indicated by the symbol β. That is because the number of constructors may sometimes bound a recursive examination of the proof of a proposition.

Here ρ^ρ is an exponentation, and $iter\,(\lambda v.\rho_1, \rho_2, \rho_3)$ is an iterated composition of function described by expression ρ_1 with respect to an argument variable v; that iteration happens ρ_2 times, and the function is applied to initial argument ρ_3. The $\rho_1[\![\rho_2/v]\!]$ describes substitution of bound variable v by ρ_2, inside expression ρ_1.

Any total function $f(...)$ over naturals has a bounds $b(...)$ function $\forall x \in \mathrm{Nat}\,.x \leq n \implies f(x) \leq b(n)$: given a range limited [9] by n, we can compute $b(n)$ that is $\max\limits_{x \leq n} f(x)$.

Conjecture Bounds terminate Since all iterations in bounds quantification terminate, all bounds terminate.

2.2 Terms

All terms are explicitly limited, but we avoid labelling terms for which bounds can be easily inferred (see below).

[7] Thanks to anonymous reviewer for pointing importance of this result.
[8] Using a bound on cost and depth of the term for each inference, we independently developed a very similar approach to that used for cost bounding in higher-order rewriting [41].
[9] This is not true for traditional real numbers $\mathbb{R}$: hyperbola $y = \frac{1}{x}$ is unbounded around 0 because $\lim\limits_{x \to 0} \frac{1}{x} = -\infty$.

$$
\begin{array}{lrcl}
\text{Size variables:} & v & \in & V \\
\text{Size values} & n & = & \mathbf{1} \,\big|\, \mathbf{S}(n) \\
\text{Term variables:} & x & \in & X \\
\text{Positive naturals:} & i & \in & \mathbb{N} \setminus \{0\} \\
\text{Upper bounds:} & \rho & ::= & v \,\big|\, i \,\big|\, \mathbf{S}(\rho) \,\big| \\
 & & \big| & \mathit{iter}\,(\lambda v.\rho, \rho, \rho) \,\big|\, \rho[\![{}^{\rho}/_{v}]\!] \,\big|\, \mathit{max}(\rho, \rho)
\end{array}
$$

Iteration is defined as:

$$
\begin{array}{lcl}
\mathit{iter}\,(\lambda v.e, \mathbf{1}, a) & = & e[\![{}^{a}/_{v}]\!] \\
\mathit{iter}\,(\lambda v.e, \mathbf{S}(n), a) & = & \mathit{iter}\,(\lambda v.e, n, e[\![{}^{a}/_{v}]\!])
\end{array}
$$

Later we will explain how bounds expressions can be encoded in the same language as the proof terms. At the level of basic logic we do not need this, but it will become useful when we consider meta-reasoning (and encode entirety of the logic within its own proof terms.)

$$
\begin{array}{lrcl}
\text{Data size bounds:} & \alpha & ::= & \rho \\
\text{Computation bounds:} & \beta & ::= & \rho \\
\text{Types:} & \tau & ::= & v \,\big|\, \tau \wedge \tau \,\big|\, \tau \vee \tau \,\big|\, \tau_v \to^{\alpha}_{\beta} \tau \,\big|\, \bot \,\big|\, \circ \,\big|\, \mathbf{Type} \\
\text{Terms:} & E & ::= & x \,\big|\, \lambda x.E \,\big|\, \mathit{in}_r\,(E) \,\big|\, \mathit{in}_l\,(E) \,\big|\, (E, E) \,\big|\, \cdot \\
 & & \big| & \mathit{case}\ E\ \mathit{of} \begin{cases} \mathit{in}_l\,(x) \to & E; \\ \mathit{in}_r\,(x) \to & E; \end{cases} \\
\text{Environments:} & \Gamma & ::= & v_1 : {\tau^1}_{\beta_1}, ..., v_n : {\tau^n}_{\beta_n} \\
\text{Judgements:} & J & ::= & \Gamma \vdash^{\alpha}_{\beta} E : \tau
\end{array}
$$

There is a special expression **Type** which is syntactically in τ. It is later used when introducing dependent types, since **Type** lives both as a type and as a term. Translation between proofs and types is later described in table 6.

Notation $A_v \to^{\alpha(v)}_{\beta(v)} B$ binds proof variable x with type of A and size variable v, and then bound in bounds $\alpha(v)$ for complexity and $\beta(v)$ for *depth* of the *normalized* term. We use notation $\alpha(v)$ instead of α to emphasize that both $\alpha(v)$ and $\beta(v)$ are functions of size variable v.

We could attach a pair of bounds to each proposition and judgement $A_{(\alpha,\beta)}$ that would describe both complexity α of computing the proof and a maximum depth β of the resulting (normalized) witness. However, in most cases, one of these would be 1 or could be inferred from the remaining information.

The $\mathit{occurs}\,(x, E)$ is a count of free occurrences of variable x in term E. Free variables of E are computed by free(E).

2.3 Inference rules

With any term variable x we need to introduce an associated bound variable v.

$$\frac{\Gamma \vdash^{?}_{?} A\ \mathbf{type} \quad x \in X \quad v \in V}{\Gamma, x_v : A \vdash^{1}_{v} x : A}\ \mathit{var}$$

Sometimes we might want to overestimate proof complexity for the sake of simplicity:

$$\frac{\Gamma \vdash^{\alpha_1}_{\beta_1} e : A \qquad \alpha_1 \le \alpha_2 \qquad \beta_1 \le \beta_2}{\Gamma \vdash^{\alpha_2}_{\beta_2} \mathit{subsume}(e, \alpha_2, \beta_2) : A}\ \mathit{subsume}$$

Table 1 Encoding arithmetic operations, hyperoperation, and Ackermann function

The addition, multiplication, and exponentiation can be defined on bounds using $\mathbf{S}()$ and $iter\,(\lambda x.\rho_1, \rho_2, \rho_2)$:

$$
\begin{array}{lcl}
hyper(a,b,\mathbf{1}) & \equiv & iter\,(\lambda x.\mathbf{S}(x), a, b) \\
hyper(a,b,\mathbf{S}(\mathbf{1})) & \equiv & iter\,(\lambda y.iter\,(\lambda x.\mathbf{S}(x), y, b), \mathbf{1}, \mathbf{1}) \\
 & & \textit{Argument } y \textit{ is ignored in this special case.} \\
hyper(a,b,\mathbf{S}(\mathbf{S}(n))) & = & iter\,(\lambda x.hyper(x,a,n), b, n) \\
h & = & \lambda g.\lambda a.\lambda b.\ case\ b\ of \begin{cases} \mathbf{1} \to & a; \\ \mathbf{S}(c) \to & iter\,(\lambda x.fxa, c, a); \end{cases} \\
hyper(a,b,n) & = & iter\,(\lambda f.h(f), n, \lambda g.g) \\
a+b & = & iter\,(\lambda x.\mathbf{S}(x), a, b) \\
a*\mathbf{1} & = & a \\
a*\mathbf{S}(b) & = & iter\,(\lambda x.x+a, b, a) \\
a^{1} & = & a \\
a^{\mathbf{S}(b)} & = & iter\,(\lambda x.x*a, b, a) \\
a[\mathbf{S}(n)]b & = & case\ b\ of \begin{cases} \mathbf{1} \to & a; \\ \mathbf{S}(c) \to & iter\,(\lambda x.x[n]b, c, a); \end{cases} \\
a+b & \equiv & hyper(a,b,\mathbf{1}) \\
a*b & \equiv & hyper(a,b,\mathbf{S}(\mathbf{1})) \\
a^{b} & \equiv & hyper(a,b,\mathbf{S}(\mathbf{S}(\mathbf{1}))) \\
\mathbf{S}(\mathbf{S}(\mathbf{S}(ack(m,\mathbf{S}(\mathbf{S}(\mathbf{S}(n)))))))) & \equiv & 2[m](\mathbf{S}(\mathbf{S}(\mathbf{S}(n))))
\end{array}
$$

Here $=$ is for definition, and $\equiv$ states equivalence of expressions. To avoid definition of predecessor function, we use equivalence to express Ackermann function.
$hyper(a,b,n)$, and $a[n]c$ are alternative ways of introducing hyperoperations.
We use hyperoperations for clarity, showing that we can indeed express Ackermann function as bounded iteration of function compositions.

Note that subsumption is necessary for *case*-expressions. Below we have typical rules for construction and destruction of basic types:

$$\frac{}{\Gamma \vdash^{1}_{\beta} \cdot : \circ}\ unit$$

$$\frac{\Gamma \vdash^{\alpha}_{\beta} e : A}{\Gamma \vdash^{\alpha+1}_{\beta+1} in_l\,(e) : A \vee B}\ inl \qquad \frac{\Gamma \vdash^{\alpha}_{\beta} e : B}{\Gamma \vdash^{\alpha+1}_{\beta+1} in_r\,(e) : A \vee B}\ inr$$

$$\frac{\Gamma \vdash^{\alpha_\vee}_{\beta_\vee+1} a : L \vee R \qquad \Gamma, x_{\beta_\vee} : L \vdash^{\alpha_l}_{\beta_l} l : B \qquad \Gamma, y_{\beta_\vee} : R \vdash^{\alpha_r}_{\beta_r} r : B}{\Gamma \vdash^{\alpha_\vee + max(\alpha_l,\alpha_r)+1}_{max(\beta_l,\beta_r)} case\ a\ of \begin{cases} in_l\,(x) \to & l; \\ in_r\,(y) \to & r; \end{cases} : B}\ case$$

$$\frac{\Gamma \vdash^{\alpha_a}_{\beta_a} a : A \quad \Gamma \vdash^{\alpha_b}_{\beta_b} b : B}{\Gamma \vdash^{\alpha_a+\alpha_b+1}_{max(\beta_a,\beta_b)+1} (a,b) : A \wedge B}\ pair$$

$$\frac{\Gamma \vdash^{\alpha}_{\beta+1} e : A \wedge B \quad i \in \{l,r\}}{\Gamma \vdash^{\alpha+1}_{\beta} prj_i e : A}\ prl \qquad \frac{\Gamma \vdash^{\alpha}_{\beta+1} e : A \wedge B \quad i \in \{l,r\}}{\Gamma \vdash^{\alpha+1}_{\beta} prj_r e : B}\ prr$$

Please note that notation $A_v \to^{\alpha}_{\beta} B$ has a size variable v declared as a depth of *normal form proof term* having type A, and then bounds α and β apply to the computation of the result.

$$\frac{\Gamma, x_v : A \vdash^{\alpha}_{\beta} e : B \quad x, v \notin \Gamma}{\Gamma \vdash^{\alpha[\![1/v]\!]+1}_{\beta[\![1/v]\!]+1} \lambda x.e : A_v \to^{\alpha}_{\beta} B} \; abs$$

Note that abstraction increases term depth by one, and application decreases it by one[10]. All introduction rules (*abs*, *pair*, *inl*, *inr*) increase β by at least one[11]. Likewise all *non-functional* (data) elimination rules (*case*, *prl*, *prr*) decrease depth expected from the resulting normal form β by one.

$$\frac{\Gamma \vdash^{\alpha_1}_{\beta_1} e : A_v \to^{\alpha_2}_{\beta_2} B \quad \Gamma \vdash^{\alpha_3}_{\beta_3} a : A}{\Gamma \vdash^{\alpha_1+\alpha_2[\![\beta_3/v]\!]+\alpha_3}_{\beta_2[\![\beta_3/v]\!]} e\ a : B} \; app$$

Please note that these rules all maintain bounded depth with no unbounded recursion. We add an explicit bounded recursive definition (like the definition of the closure) with this rule:

$$\frac{\Gamma \vdash^{\alpha_1}_{\beta_1} f : A_v \to^{\alpha_2}_{\beta_2} A \quad \Gamma \vdash^{\alpha_3}_{\beta_3} k : B \quad \Gamma \vdash^{\alpha_4}_{\beta_4} a : A}{\Gamma \vdash^{\alpha_1+iter(\lambda v.\alpha_2,\beta_3,\beta_4)+\alpha_3+\alpha_4}_{\beta_1[\![iter(\lambda v.\beta_2,\beta_3,\beta_4)/v]\!]} rec(f,k,a) : B} \; rec$$

Here the depth of the term must decrease at each step of the recursion. With the exception of *subsume*, and *rec* these are all reinterpretations of rules for intuitionistic logic [11, 75, 72], enriched with bounds on the proof length α and normalized depth of term t namely $|t|_d$ as depth expression β. The rule *rec* allows for explicitly *bounded recursion*, as opposed to traditional approaches that rely on an unbounded fixpoint[12].

Note that we may quantify on higher order values, but we cannot recurse indefinitely: there is always a limit to a number of function compositions allowed. Power of bounded function composition gives an explicit limit to Peano Arithmetic induction[38]: any computational application of Peano induction is unbounded. At the same time, we can use multiple recursions over bounded number of functions, terms, not just natural numbers. Wired-in explicit bounding also allows us to prove termination of arbitrary "towers" of function compositions, like hyperoperations [7, 33, 71, 62][13], including Goodstein functions that cannot be proven *within* PA itself[27].

2.3.1 Implicit universal quantification

In the propositional logic above, provability allows us to confirm statements with $\forall$ for all variables on top. Given that statement of existence of bounded proof term x for witness bounded by result size v can be interpreted in the following way in unbound logic $\exists v \in \mathbb{N}^+ . \exists x . |x|_d \leq v$.

[10] This allows us to correctly treat Church encoding.

[11] For *unit*, the inner proof term would have null depth, since there is no term there. Thus depth is $0 + 1$ instead of $\beta + 1$.

[12] Fixpoint may not exist, thus leading not only to arbitrarily long computation, but also to undecidability in cases where computation may never end.

[13] See table 1 to see how hyperoperations and Ackermann function can be encoded using *iter* for bounded iteration of function composition.

So $a_v \to^{\alpha}_{\beta} b$ becomes the following statement in unbounded logics $\forall v.\forall a.(|a|_d < v) \to a \to b \wedge |b|_d < \beta \wedge c(b) < \alpha$. That is: we can infer that fact for all a below an arbitrarily large depth, and bound the depth and computational complexity of the resulting witness.

This concludes our treatment of Ultrafinitist Propositional Logic (UFPL).

2.3.2 Quantification with dependent types

It is customary in constructive mathematics and theorem proving to use dependent types instead of usual universal and existential quantifiers [52].

Please note that just like one can define intuitionistic propositional logic with just implication and then encode both sum and product types[36], so Π type can express both universal quantification and plain implication, while Σ type can express both existential quantification and product type. Since implication can already express sum and product types in polymorphic calculus, we will only show how to modify rules for implication and lambda to make the Π type that corresponds to universal quantification.

While it is usual to introduce universal quantification directly in calculi without proof terms, we will introduce them with Π types, like is now customary in dependently typed languages.

First we need a rule to introduce a type variable:

$$\frac{\Gamma \vdash^{\alpha}_{\beta} t\ \mathbf{type}}{\Gamma, v < \beta, x_v : t \vdash^{1}_{v} x : t}\ \mathit{tyvar}$$

This rule allows us to use variables at type level, and together with Π and Σ types allow to express quantification.

For the inequalities, it suffices to ensure that they are not cyclic and thus unsatisfiable. Note that inequality stems from the fact that value is always no longer than its encoding as a type.

$$\frac{\Gamma \vdash^{\alpha_1}_{\beta_1} A\ \mathbf{type} \quad \Gamma, x_v : A \vdash^{\alpha_2(v)}_{\beta_2(v)} B\ \mathbf{type} \quad x, v \notin \Gamma}{\Gamma \vdash^{\alpha_1+\alpha_2[\![1/v]\!]}_{max(\beta_1,\beta_2[\![1/v]\!])+1} \Pi(x_v : A) \to^{\alpha_2}_{\beta_2} B\ \mathbf{type}}\ \mathit{forall\text{-}form}$$

Please note that similarly to the treatment of lambda abstraction as proof of implication, we estimate the computational cost of dependent product by substituting free variables with 1, but now we still need to consider the same substitution in the resulting Π type.

Treatment of universal quantifier bears usual similarity[52] to *abs* and *app* rules:

$$\frac{\Gamma, x_v : A \vdash^{\alpha}_{\beta} e : B}{\Gamma \vdash^{\alpha[\![1/v]\!]+1}_{\beta[\![1/v]\!]+1} \lambda(x : A).e : \Pi(x_v : A) \to^{\alpha}_{\beta} B}\ \mathit{forall\text{-}intro}$$

Elimination works the same way as for usual application, since computation works after type erasure.

$$\frac{\Gamma \vdash^{\alpha_1}_{\beta_1} e : A_v \to^{\alpha_2}_{\beta_2} B \quad \Gamma \vdash^{\alpha_3}_{\beta_3} a : A}{\Gamma \vdash^{\alpha_1+\alpha_2[\![\beta_3/v]\!]+\alpha_3}_{\beta_2[\![\beta_3/v]\!]} e\ a : B[\![A/x]\!]}\ \mathit{forall\text{-}app}$$

This allows us to replace implication and by extension, all UFPL types. It also allows for quantification of higher order values.

We leave introducing Σ-types to interested students of type theory, since they are not essential to our argument that we may have a decidable higher order logic.

Since introduction creates the proof term in the same way, the proof terms can be enumerated in the same way as shown in section 4.4 on page 13.

Table 2 Simplication of bounds. May rewrite left to right.

(1)	$a * x^e + b * x^f$	$\leq$	$(a+b) * x^f$
(2)	$a * x^e * y^g$	$\leq$	$a * x^f * y^h$
(3)	$iter\,(\lambda v.e, g, x)$	$\leq$	$iter\,(\lambda w.f, h, x)$
(4)	$iter\,(\lambda v.v * a, e, x)$	$=$	$a^e * x$
(5)	$iter\,(\lambda v.v + a, e, x)$	$=$	$x + a * e$
(6)	$iter\,(\lambda v.v^e, g, x)$	$=$	x^{e^g}

Assumptions:
- $x, y \geq 1$ are data size variables in the environment,
- $1 \leq e \leq f$ and $1 \leq g \leq h$ are arbitrary positive and increasing expressions,
- $a, b, c... \geq 1$ are constants.

3 Application of the logic

3.1 Using proofs

Each proof ultimately leads to a judgment $\Gamma \vdash^{\alpha}_{\beta} e : A$. We may resolve all upper bound variables $v_1, v_2, ..., v_n$ in the α to get an upper bound on computational complexity of the statement, and in β to get an upper bound on normalized term resulting from the proof. This way all proofs are ultra-finitary statements: Only as long as α is less than our assumed limit, we will consider the proof valid and proof computation to be available within the given time.

3.2 Simplifying upper bounds

Our inference rules rely on computing upper bounds and their inequality. Here we note a few inequalities that simplify reasoning about these bounds, albeit at the cost of making them somewhat looser.

First, we note that all variables are positive naturals because they represent the data of non-zero size: $x \geq 1$.

That means that the following laws are true, assuming that $x, y, ... \geq 1$ are data size variables in the environment, $1 \leq e \leq f$ and $1 \leq g \leq h$ are arbitrary positive expressions, and $a, b, c... \geq 1$ are constants. For easier use, the rules are presented in left-to-right order, just like conventional rewrite rules.

We may thus use these rules to loosen the bound in such a way as to reduce the size of the bound expression and make it a sum of a single term in all variables and an additional constant term. This reduction may be delayed until we have bound to verify.

We may use inference rules leaving "type holes"[57] instead of bounds, which could be named "bound holes", and let them be filled by the framework interpreter.

3.3 Reduction

Reduction relation is defined as small step semantics [60] in order to preserve number of computational steps made over the course of evaluation. See table 3 on page 9.

When performing application, we expect substitution to take work proportional to the number of occurrences of the variable, like changing links on directed acyclic graph of the term.

Table 3 Reduction rules.

$$\frac{e \Downarrow_k e'}{case\ e\ of \begin{cases} in_l\,(x) \to & b; \\ in_r\,(y) \to & c; \end{cases} \Downarrow_k case\ e'\ of \begin{cases} in_l\,(x) \to & b; \\ in_r\,(y) \to & c; \end{cases}}\ eval\text{-}case\text{-}arg$$

$$\frac{k = occurs\,(x, b)}{case\ in_l\,(a)\ of \begin{cases} in_l\,(x) \to & b; \\ in_r\,(y) \to & c; \end{cases} \Downarrow_k b[\![a/x]\!]}\ eval\text{-}case\text{-}left$$

$$\frac{k = occurs\,(y, c)}{case\ in_r\,(a)\ of \begin{cases} in_l\,(x) \to & b; \\ in_r\,(y) \to & c; \end{cases} \Downarrow_k c[\![a/y]\!]}\ eval\text{-}case\text{-}right$$

$$\frac{k = occurs\,(x, e)}{(\lambda x.e) f \Downarrow_k e[\![f/x]\!]}\ eval\text{-}app$$

$$\frac{e \Downarrow_n e' \quad i \in \{l, r\}}{in_i(e) \Downarrow_n in_i(e')}\ eval\text{-}sum$$

$$\frac{}{prj_l\,(a, b) \Downarrow_1 a}\ eval\text{-}prl \qquad \frac{}{prj_r\,(a, b) \Downarrow_1 b}\ eval\text{-}prr$$

$$\frac{a_l \Downarrow_n a_l'}{(a_l, a_r) \Downarrow_n (a_l', a_r)}\ eval\text{-}pairleft$$

$$\frac{a_r \Downarrow_n a_r'}{(a_l, a_r) \Downarrow_n (a_l, a_r')}\ eval\text{-}pairright$$

For discussion of efficient reduction of lambda terms please read [45, 5], since here we focus on demonstration with a simplified cost model.

3.4 Self-encoding

3.4.1 Natural numbers

In this section we will encode bounds, propositions (types) and proof terms as proof terms within UFPL. Thus $[\![..]\!]$ corresponds to LISP quote.

Below we use notation $\mathbb{B}\,(v)$ for de Brujin index of the variable [15].

Table 4 Encoding natural numbers.

Nat_β	$=$	$\left(rec(x, [\![\circ \vee x]\!], \beta)[\![\circ]\!] \right)$		
zero	$=$	$in_l\,(\cdot)$	$:^1_1$	Nat_1
succ	$=$	$\lambda x_v \to^1_{v+1} in_r\,(x)$	$:^1_{v+1}$	$\mathrm{Nat}_v \to Nat_{v+1}$

Table 5 Encoding bounds.

Var_β	$=$	Nat_β
$\mathrm{Bound}_{\beta+1}$	$=$	$\mathrm{Var} \vee \mathrm{Nat}_\beta \vee \circ \vee (\mathrm{Bound}_\beta, \mathrm{Bound}_\beta)$
	$\vee$	$(\mathrm{Bound}_\beta, \mathrm{Bound}_\beta)$
	$\vee$	$(\mathrm{Bound}_\beta, \mathrm{Bound}_\beta)$
	$\vee$	$(\mathrm{Bound}_\beta, (\mathrm{Bound}_\beta, \mathrm{Var}))$
	$\vee$	$(\mathrm{Bound}_\beta, (\mathrm{Var}, \mathrm{Bound}_\beta))$
$[\![v]\!]$	$=$	$in_l\,(in_l\,(in_l\,(\mathbb{B}\,(v))))$
$[\![i]\!]$	$=$	$in_l\,(in_l\,(in_r\,(i)))$
$[\![\cdot]\!]$	$=$	$in_l\,(in_l\,(in_r\,(\cdot)))$
$[\![\rho_1 + \rho_2]\!]$	$=$	$in_l\,(in_r\,(in_r\,(([\![\rho_1]\!], [\![\rho_2]\!]))))$
$[\![\rho_1 * \rho_2]\!]$	$=$	$in_r\,(in_l\,(in_l\,(([\![\rho_1]\!], [\![\rho_2]\!]))))$
$[\![\rho_1^{\rho_2}]\!]$	$=$	$in_r\,(in_l\,(in_r\,(([\![\rho_1]\!], [\![\rho_2]\!]))))$
$[\![iter\,(\lambda v.\rho_1, \rho_2, \rho_3)]\!]$	$=$	$in_r\,(in_r\,(in_l\,((\mathbb{B}\,(v), [\![\rho_1]\!], ([\![\rho_2]\!], [\![\rho_3]\!])))))$
$[\![\rho_1\ [\![\rho/v]\!]]\!]$	$=$	$in_r\,(in_r\,(in_r\,(([\![\rho_1]\!], ([\![\rho_2]\!], \mathbb{B}\,(v))))))$

3.4.2 Encoding bounds

Now we may encode bounds (table 5), types (table 6), and proof terms (table 7).

This encoding allows us to make operations on types akin to generic programming in Haskell [50].

Our inference rules rely on computing bounds and their inequalities. Given that all variables are positive naturals because they represent the data of non-zero size: $x \geq 1$, we may simplify these bounds with a set of simple inequalities.

3.4.3 Encoding proof terms

Note that every type term in *normal form* is longer than its own type.

▶ **Theorem 1** (Encoding). *All bound, type, proof, or proposition of UFPL can be encoded as a proof term of UFPL.*

Details are visible in the tables 4-7.

Table 6 Encoding types.

$[\![A \vee B]\!]$	$=$	$in_l\,(in_l\,(([\![A]\!], [\![B]\!])))$
$[\![A \wedge B]\!]$	$=$	$in_l\,(in_r\,(([\![A]\!], [\![B]\!])))$
$[\![A_v \to^\alpha_\beta B]\!]$	$=$	$in_r\,(in_l\,((\lambda x : A.[\![B]\!], (\lambda v : \mathrm{Nat}_v\,.[\![\alpha]\!], \lambda v : \mathrm{Nat}_v\,.[\![\beta]\!]))))$
$[\![\circ]\!]$	$=$	$in_r\,(in_r\,(\cdot))$

Table 7 Encoding terms.

$[\![x_v]\!]$	$=$	$in_l\,(in_l\,(in_l\,((\mathbb{B}\,(x), v)))))$
$[\![subsume(A,B)]\!]$	$=$	$in_l\,(in_l\,(in_l\,(in_r\,(([\![B]\!]_{\text{Bound}}, [\![A]\!])))))$
$[\![unit]\!]$	$=$	$in_l\,(in_l\,(in_r\,(in_l\,(\cdot))))$
$[\![in_l\,(A)]\!]$	$=$	$in_l\,(in_l\,(in_r\,(in_r\,(A))))$
$[\![in_r\,(A)]\!]$	$=$	$in_l\,(in_r\,(in_l\,(in_l\,(A))))$
$[\![prj_l A]\!]$	$=$	$in_l\,(in_r\,(in_l\,(in_r\,(A))))$
$[\![prj_r A]\!]$	$=$	$in_l\,(in_r\,(in_r\,(in_l\,(A))))$
$[\![(A,B)]\!]$	$=$	$in_l\,(in_r\,(in_r\,(in_r\,(([\![A]\!], [\![B]\!])))))$
$[\![AB]\!]$	$=$	$in_r\,(in_l\,(in_l\,(in_l\,(([\![A]\!], [\![B]\!])))))$
$[\![\lambda x_v.A]\!]$	$=$	$in_r\,(in_l\,(in_l\,(in_r\,(((\mathbb{B}\,(x), \mathbb{B}\,(v))\,, [\![A]\!])))))$
$[\![rec(v,A,B)C]\!]$	$=$	$in_r\,(in_l\,(in_r\,(in_l\,(((\mathbb{B}\,(v), [\![A]\!])\,, ([\![B]\!], [\![C]\!])))))))$

4 Properties of the logic

When implementing the computation seems straightforward, we will just establish the finite limit for the computation that should be taken as a proof. That is what we describe as *problem is decidable by the limit of* a given complexity. This approach explicitly describes undecidable problems as those that require an infinite number of steps to solve.

4.1 Consistency

Here we will only use well-known proof of consistency of intuitionistic logic [11, 75, 72][14]. We do not use the self-encoding presented in section 3.4.

▶ **Theorem 2** (Consistency of UFL). *UFPL is consistent, if intuitionistic propositional logic is consistent.*

Proof. After elision of bounds[15], we interpret the rule *subsume* as $id = \lambda x.x$. Then we see the standard proof rules for intuitionistic logic. The consistency follows from the consistency of intuitionistic logic. ◀

4.2 Expressivity

▶ **Theorem 3** (At least as expressive as PRA.). *UFL can express all Primitive Recursive programs.*

Proof. It is easy to show that our logic can emulate bounded loop programs[53] which has power equivalent to primitive recursive functions[64]. Every bounded *loop* can be encoded by $iter\,(\lambda v.loop, x, n)$, then every flat logical statement can be encoded with a tuple containing states of the variables. ◀

One could muse that this class does not cover all Bounded Turing Machine[34] programs. In order to support these, we would need to define more general bounding functions.

[14] The proof above is totally independent of previous conjectures.
[15] Elision of bounds is only used once to prove consistency.

One can replace upper bound expressions with arbitrary bounding functions expressed in simply typed lambda calculus (see section 3.4). These are the operations used in inference rules. However, such functions are more difficult to bound and compute themselves.

It has been proven that any function whose complexity is *bounded* by primitive recursive function is also primitive recursive[16], which means that estimating our complexities could become an impossibly long endeavour, but logically consistent one.

To give an example of simplified Ackermann function which is the best known example of function beyond PRA [70, 3], evaluation takes $A(5) = 2^{2^{2^{2^{16}}}} - 3$[40]. That means that these evaluations quickly get out of hand and indeed outside of any reasonable limits.

The encoding of Ackermann function is through hyperoperation in table 1.

4.3 Bounded Turing completeness

An evidence of stronger expressivity may be found by encoding bounded Turing Machine programs in UFPL. This proof uses encoding similar to 3.4, but for a Turing Machine. For a reference on encoding of Turing machines in lambda calculus see [2].

For any complexity bound $f(x)$ expressible in the language of ultrafinitist logic, and an algorithm that satisfies it and emulation function with complexity of $e(f)(x)$ – that is an encoding $e(f)$ of f, applied to the argument x of f – which we can encode this emulation as a bound.

▶ **Theorem 4** (Emulation complexity). *Assume a time complexity $c(x)$ for program (or proof) s that can be encoded as UFL bounds. If we can emulate (encode evaluation) of $f(x)$ with an overhead e for each step, then we can prove that complexity of evaluating s is $e * c(x) + cc(x)$.*

Where $cc(x)$ is complexity of evaluating complexity bounds for the encoding $c(x)$.

Proof. Given each step of emulation encoded as $s(x)$, where x is a current state, emulation with a complexity function encoded as $f(x)$ can be executed by $iter\,(\lambda v.s, f(x), x)$.

Assuming that $e(f)$ is function emulation in UFPL, we can write proof expression $iter\,(\lambda x.e(f)(x), e(c)(x), x)$. This expression evaluated encoded s and has exactly the assumed complexity ◀

The most complex part of the proof may be logically inferring the right complexity $c(x)$ and totality of the function f within this number of steps.

▶ **Theorem 5** (Bounded Turing Machine emulation). *For programs of Bounded Turing Machine f over alphabet size $|a|$ and number of states $|s|$ with complexity that can be encoded in UFL, we can prove time complexity of $\lg_2(|a|) + \lg_2(|s|) * |c| + |cc|$ with UFPL. Note that $|cc|$ is cost to evaluate complexity function itself.*

Proof. We use emulation argument for Bounded Turing machines that may be limited by bounds described above, it is too with $\mathcal{O}\left(\log^2(a) * f(x)\right)$, where a is the bound on the size of alphabet and number of states of the machine.

For the Bounded Turing machine we encode tape as pair of lists, with current position at the top of both lists.

Then we encode the following steps:

- examine the alphabet character: $\mathcal{O}\left(\log(|a|)\right)$
- examine finite state machine for a character: $\mathcal{O}\left(\log(|s|)\right)$
- move one step up or down the tape by moving the top from one line list to the other: $\mathcal{O}\left(1\right)$;

- if we want to write at the current position, we take the top element from the right list, and put the new one.

Together they make a single step of the Turing machine at the cost of $\mathcal{O}(\log(|a|) + \log(|s|))$. ◀

We encode variable bindings as a dictionary with cost of $\mathcal{O}(\lg_2 |\operatorname{Var}|)$, where Var is number of variables used. All operations not involving substitution should remain at $\mathcal{O}(1)$ complexity within emulation.

▶ **Lemma 6** (Self-emulation). *ULF self-emulation of function with integral bound $|cc|$ is feasible within $\mathcal{O}(|cc| * \lg_2 |\operatorname{Var}|)$.*

Overall we can infer that for each algorithm of bounded complexity B that we may encode in ULF, we may use ULF self-emulation to find a proof with complexity of at most $e * B * \lg_2 |\operatorname{Var}|$. All steps of ULF are $\mathcal{O}(1)$ with respect to inferred bounds on computational complexity, with the exception of function application and variable substitution which are $\mathcal{O}(\lg_2 |\operatorname{Var}|))$

▶ **Theorem 7** (Emulation completeness). *If the bounds that can be encoded within the bounds function, the UFPL is complete for proving its own bounds up to the cost of self-emulation e.*

Since we can encode any statement in UFPL in UFPL itself, this likely would mean the proof of emulation completeness can be written in the UFPL itself.

There are complex ways of proving completeness that apply in the realm of non-idempotent intersection type systems, but they use a more abstract notion of complexity[1].

4.4 Decidability of bounded statements

▶ **Theorem 8** (Decidability). *Every valid proposition with a* fixed bound on input *n can be checked by enumerating inputs, and is thus decidable.*

This comes at the cost of complexity that increases by $\alpha(n) * a^n$, where n is the depth of input, since we need to enumerate all inputs of depth n. Proof follows directly from enumeration, and bounds.

Proof. Let's try to enumerate the terms that can be constructed for a given bounds, without using *subsume* rule:

Bounds function will contain a given number n of successor functions, and *iter* expressions. Each time we make a single inference rule, and construct a slightly more complex term, we add a successor function or *iter* expression, the number of different proof tree shapes equals to the number of ternary trees constructible with n nodes. This number is defined as OEIS A001764 [39] and given by the algebraic term $\frac{\binom{3n}{n}}{2n+1}$. Given that for each of n nodes we can choose one of the 12 rules (12^n different choices), we have at most $\mathcal{O}\left(\frac{12^n \binom{3n}{n}}{2n+1}\right)$ different proofs with the complexity given by a term of n nodes.

With the inclusion of subsumption rule, we can only decrease the n, so at most:

$$\sum_{i=1..n} \mathcal{O}\left(\frac{12^n}{2n+1}\binom{3n}{n}\right) = \mathcal{O}\left(12^n \binom{3n}{n}\right)$$

Since number of proofs is finite, we can decide the provability after they are exhausted. ◀

Thus all judgements with bounds are decidable. This property is shared with some other *resource bounded logics* [1].

Given that exponential lower bound has been established for implicational intuitionistic logic [37], we expect that lower bound for ultrafinitist logic will also be exponential and thus the proposed bound is asymptotically tight.

4.5 Paradox of undecidability

Expressing any statements about undecidability implicitly requires unbounded computational effort. Since all our proofs and arguments are explicitly bounded, there is no room to state undecidability. Thus we conclude that this paradox is removed from ultrafinitist logic: statement of undecidability is *invalid* as a proposition. All *valid propositions* are decidable.

This is not as outrageous as it superficially seems, since we already know that computation models that would allow transfinite number of steps would also make all functions computable [30].

4.6 Finitary completeness

Let's assume we have upper bounds on all variables within an intuitionistic theorem.

Can we prove it with UFL?

▶ **Theorem 9** (Preservation of bounded intuitionistic theorems). *Any intuitionistic theorem bounded by definite integers in UFL can be proven in UFL.*

Proof. Let's enumerate complexities of computing intuitionistic proof for a given set of inputs bounded by given value. We may enumerate these proofs, and thus take maximum length of the computation. This maximum length will be upper bound on all proofs. ◀

This proof uses 4.4, and 3.4. Naturally this means that all statements with bounds but proof without bounds will also have proof with bounds.

5 Related work

The philosophical problem with transfinite arguments has been spotted long before [42, 61, 79, 22, 47]. Automatic theorem provers like Coq require a monotonically decreasing bounding function in the ordinal domain for each inductive definition [56, 58, 8]. This makes all recursive definitions *well-founded*[56], but since transfinite ordinals are permitted, it also allows theories outside computable universe.

The computation of a bounding function may turn out to take unfeasible amount of time. Cost calculi for functional languages attempt to assign cost to certain operations in order to reason about time and space complexity [65]. But these approaches do not require all proofs and propositions to carry the cost as we do.

Philosophers have postulated distinction between feasible computations and unfeasible ones [79], however it was considered unclear whether it is possible to realize this distinction on the basis of a logic [74], with some claiming that such a logic could not be consistent [17, 51].

There exist logics that implicitly constrain computational complexity of the proofs, for example Bounded Arithmetic [12, 43, 14] that is restricted to computations in polynomial time. However, most of them are significantly weaker than class of primitive recursive functions, which is widely considered to contain most useful programs. This would put the logician in a position of trying to state a widely known facts about objects that are inexpressible within the logic.

6 Discussion

6.1 Explicit bounds versus implicit structural recursion

It is long known that unbounded logics may give rise to paradoxes [23, 13], and the use of implicit techniques [14], including bounded recursion [43], structural recursion [59], well-founded sets [56], or predicative bounding [21] were developed.

Using explicit bounds provides a more obvious solution, which is easier to prove correct, and parallels development of an explicit mathematical limit [69, 29], starting from Eudoxus' method of exhaustion [18], through implicit notion of terminus [76] to a modern concept of a mathematical limit of a function [78][16].

6.2 Open problem of directly proving bounded Turing completeness

Note that the proof above mentions Turing completeness, if we can prove that all bounds can be expressed by the bounds functions defined above. While the usual examples of fast growing functions like Ackermann [70, 3, 40], or Goodstein [27, 38] are expressible by bounded composition of functions, the clarity is still elusive. (Of course such fast growing functions would quickly surpass any reasonable limit.)

We still search for a proof of bounded Turing completeness that would not use a recursive argument, where we replace bound functions with arbitrary bounded lambda expressions. That is because our induction principle would have to be more complex to include the latter.

Interested student may prove Turing completeness by encoding to Kleene normal form with iteration on top [31]. In this logic, Kleene normal form may be made explicit.

6.3 Computability as foundation of mathematics

Finite descriptions of the proofs and their objects are most rational foundations of mathematics. These objects are all definable by bounded Turing computability.

Attempts to define *hypercomputation* beyond bounded Turing machine immediately lead to physical impossibilities [54]. At the same time computability or bounded Turing machine and total computable functions have been translated between multiple mathematical models. Hence we conjecture that the only mathematical proof principle that is immune to rational doubt is the bounded Turing machine, and ultrafinitist logic.

A logic that allows expression of any bounded Turing function and nothing else could be rightly called a logic of computable functions, and a best candidate for encoding foundations of mathematics. Alternative attempts to narrow set theory by predicativism [21, 68] are subject to critique[77] that motivates further search.

That is because we can encode axioms that are incomputable as function parameters with assumed types, and use these to prove or disprove theorems of traditional axiomatic theories without endangering consistency of the underlying logical framework.

[16] Interestingly delta-epsilon definition is formalizable for computable functions, by assuming that as n approaches the limit the smaller computable environment is taken. Of course both delta and epsilon would have to be a finite expansions (approximations) instead of possibly transcendental value. This would give a definition of "computable limit".

6.4 Automated theorem proving

Interesting avenue for future work would be to define a full type theory, dependently typed language and an automatic prover for these inference rules. Improving on the bound of $\mathcal{O}\left(\binom{3n}{n}\right)$ for deciding subtheorems would be possible, since we only need to consider normal forms. It would be exciting to prove metatheoretic results about the UFL in itself, and verify it with an automatic theorem prover.

Since meta-reasoning always results in longer proofs than original theorems, the UFL may also allow us to prove consistency of ultrafinitist arithmetic, enabling to second Hilbert problem [32], and potentially allowing self-verifiable formalization of mathematics.

Theories for uncomputable are only indirectly formalisable within such framework as functions taking uncomputable actions (like infinite recursor of Peano arithmetic) as arguments. Previously created theories are prone to high complexity and errors due to difficulty at maintaining expressivity and consistency together. Simplicity of proving the hierarchy of universes as hierarchy of complexities, and expanding ultrafinitist logic with strongly normalizable dependent types gives us hope that such automated theorem proving framework would be simpler.

Since the logic includes upper bounds for all functions, we may use these and proof irrelevance to automatically and safely optimize proofs as well. For example, we could automatically replace computation of naturals defined by successor function with computation defined on positional binary numbers.

6.5 Proving decidability in strictly finite domain

The explicit bounding of all objects, including proofs in this work is used to prevent undecidability within finite domain [63].

7 Conclusion

We have shown a possible consistent logic for inference with a strictly bounded number of steps. This allows us to limit our statements by the length of acceptable proof, and thus define statements that are both true, and computable within Bremermann-Gorelik limit [28] This inference system explicitly bounds both the length of the resulting proof, and the bounds on the depth of the normalized result term. This allows avoiding inconsistencies suggested by philosophical work, and at the same time steers away from relatively weak logics with implicit complexity like Bounded Arithmetic [43], which capture polynomial time hierarchy. It also shows how much we gain by making explicit bounds, since these may be tighter than with implicit complexity approaches. While Emulation Complexity is the powerful approach to proving expressivity of this logic, it would be nice to see a proof with tighter bounds on what we may prove with it[17].

We strive to prove that all bounded computable functions are expressible within this framework, and thus we propose this logic as a "logic of practical computability".

[17] A promising avenue of work would be proving that *amortized complexity* by replacing single bound variable by a vector of monotonic bound variables. Another approach would be attempt to obtain tighter bounds directly by separately counting beta-reduction steps and substitutions, instead of all reduction steps and substitutions together [1].

References

1 Beniamino Accattoli, Stéphane Graham-Lengrand, and Delia Kesner. Tight typings and split bounds. *Proc. ACM Program. Lang.*, 2(ICFP), July 2018. doi:10.1145/3236789.

2 Beniamino Accattoli, Ugo Dal Lago, and Gabriele Vanoni. A log-sensitive encoding of turing machines in the λ-calculus, 2023. arXiv:2301.12556.

3 Wilhelm Ackermann. Zum hilbertschen aufbau der reellen zahlen. *Mathematische Annalen*, 99(1):118–133, December 1928. doi:10.1007/BF01459088.

4 Yehoshua Bar-Hillel and Rudolf Carnap. Semantic information. *British Journal for the Philosophy of Science*, 4(14):147–157, 1953. doi:10.1093/bjps/IV.14.147.

5 Pablo Barenbaum and Eduardo Bonelli. Optimality and the Linear Substitution Calculus. In Dale Miller, editor, *2nd International Conference on Formal Structures for Computation and Deduction (FSCD 2017)*, volume 84 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 9:1–9:16, Dagstuhl, Germany, 2017. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.FSCD.2017.9.

6 Jose Benardete. *Infinity: An Essay in Metaphysics*. Clarendon Press, 1964.

7 Albert A. Bennett. Note on an operation of the third grade. *Annals of Mathematics. Second Series*, 17 (2):74–75, December 1915. doi:10.2307/2007124.

8 Yves Bertot and Vladimir Komendantsky. Fixed point semantics and partial recursion in Coq. In *MPC 2008*, Marseille, France, July 2008. URL: https://hal.inria.fr/inria-00190975.

9 Tobias Boolakee, Christian Heide, Antonio Garzon-Ramirez, Heiko B. Weber, Ignacio Franco, and Peter Hommelhoff. Light-field control of real and virtual charge carriers. *Nature*, 605(7909):251–255, May 2022. doi:10.1038/s41586-022-04565-9.

10 Manuel Bremer. Varieties of finitism. *Metaphysica*, 8:131–148, October 2007. doi:10.1007/s12133-007-0012-9.

11 l. e. j. Brouwer. *Over de grondslagen der wiskunde*, volume 1 of *mc varia*. Mathematisch Centrum, Amsterdam, 1981. including unpublished fragments, correspondence with D. J. Korteweg and reviews by G. Mannoury, edited and with an introduction by D. Van Dalen.

12 Samuel R. Buss. The polynomial hierarchy and intuitionistic bounded arithmetic. In Alan L. Selman, editor, *Structure in Complexity Theory, Proceedings of the Conference hold at the University of California, Berkeley, California, USA, June 2-5, 1986*, volume 223 of *Lecture Notes in Computer Science*, pages 77–103. Springer, 1986. doi:10.1007/3-540-16486-3_91.

13 Thierry Coquand. An analysis of girard's paradox. In *Proceedings of the First Annual IEEE Symposium on Logic in Computer Science (LICS 1986)*, pages 227–236. IEEE Computer Society Press, June 1986.

14 Ugo Dal Lago. Implicit computation complexity in higher-order programming languages: A survey in memory of martin hofmann. *Math. Struct. Comput. Sci.*, 32(6):760–776, 2022. doi:10.1017/S0960129521000505.

15 Nicolaas Govert De Bruijn. *Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem*, volume 75, pages 381–392. Elsevier, 1972.

16 P. J. Denning, J. B. Dennis, and J. E. Qualitz. *Machines, Languages, and Computation*. Prentice-Hall, 1978.

17 Michael Dummett. Wang's paradox. *Synthese*, 30(3/4):301–324, 1975. URL: https://link.springer.com/article/10.1007/BF00485048.

18 Liu Dun. A comparison of archimedes' and liu hui's studies of circles. In *Chinese Studies in the History and Philosophy of Science and Technology 179*, pages 279–287. Kluwer Academic Publishers, 1966.

19 Marie Duzi. The paradox of inference and the non-triviality of analytic information. *Journal of Philosophical Logic*, 39:473–510, October 2010. doi:10.1007/s10992-010-9127-5.

20 Lisa Dyson, Matthew Kleban, and Leonard Susskind. Disturbing implications of a cosmological constant. *Journal of High Energy Physics*, 2002(10):011, November 2002. doi:10.1088/1126-6708/2002/10/011.

21 Solomon Feferman. Systems of predicative analysis. *Journal of Symbolic Logic*, 29:1–30, 1964. URL: https://api.semanticscholar.org/CorpusID:35126801.

22 Amanda Gefter. Mind-bending mathematics: Why infinity has to go. *New Scientist*, 219(2930):32–35, 2013. doi:10.1016/S0262-4079(13)62043-6.

23 Jean-Yves Girard. *Interprétation fonctionnelle et Élimination des coupures de l'arithmétique d'ordre supérieur*. PhD thesis, Université Paris Diderot - Paris 7, 1972. URL: https://www.cs.cmu.edu/~kw/scans/girard72thesis.pdf.

24 Nicolas Gisin. Indeterminism in physics, classical chaos and bohmian mechanics. are real numbers really real?, 2019. arXiv:1803.06824.

25 Kurt Gödel. Über formal unentscheidbare sätze der principia mathematica und verwandter systeme i. *Monatshefte für Mathematik und Physik*, 38(1):173–198, December 1931. doi:10.1007/BF01700692.

26 Kurt Gödel, S. Feferman, J.W. Dawson, S.C. Kleene, G. Moore, R. Solovay, and J. van Heijenoort. *Kurt Gödel: Collected Works: Volume I: Publications 1929-1936*. Collected Works of Kurt Godel. OUP USA, 1986. URL: https://books.google.pl/books?id=5ya4A0w62skC.

27 R. L. Goodstein. On the restricted ordinal theorem. *The Journal of Symbolic Logic*, 9(2):33–41, 1944. doi:10.2307/2268019.

28 Gennady Gorelik. Bremermann's limit and cgh-physics, 2010. arXiv:0910.3424.

29 Judith V. Grabiner. Who gave you the epsilon? cauchy and the origins of rigorous calculus. *The American Mathematical Monthly*, 90(3):185–194, 1983. URL: http://www.jstor.org/stable/2975545.

30 Joel David Hamkins. Every function can be computable! Blog post., March 2016. URL: http://jdh.hamkins.org/every-function-can-be-computable/.

31 Hans Hermes. *Enumerability, decidability, computability - an introduction to the theory of recursive functions*. Springer, 1969.

32 David Hilbert. Mathematical problems. *Bulletin of the American Mathematical Society*, 8(10):437–479, 1902.

33 David Hilbert. Über das unendliche. *Mathematische Annalen*, 95:161–190, 1926. doi:10.1007/BF01206605.

34 John E. Hopcroft and Jeffrey D. Ullman. Relations between time and tape complexities. *J. ACM*, 15(3):414–427, July 1968. doi:10.1145/321466.321474.

35 William A. Howard. The formulae-as-types notion of construction. In J. Hindley and J. Seldin, editors, *To H. B. Curry: Essays on Combinatory Logic, λ-calculus and Formalism*, pages 479–490. Academic Press, 1980.

36 Jan Martin Jansen. *Programming in the λ-Calculus: From Church to Scott and Back*, pages 168–180. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. doi:10.1007/978-3-642-40355-2_12.

37 Emil Jeřábek. A simplified lower bound for implicational logic, 2023. arXiv:2303.15090.

38 Laurie Kirby and Jeff Paris. Accessible independence results for peano arithmetic. *Bulletin of the London Mathematical Society*, 14(4):285–293, 1982. doi:10.1112/blms/14.4.285.

39 Don Knuth, Frank Ellerman, and Natan Arie Consigli. A001764 – a(n) = binomial(3*n,n)/(2*n+1) (enumerates ternary trees and also noncrossing trees). (formerly m2926 n1174), 2016. URL: https://oeis.org/A001764.

40 Don Knuth, Frank Ellerman, and Natan Arie Consigli. A046859: Simplified ackermann function (main diagonal of ackermann-péter function), 2016. URL: https://oeis.org/A046859.

41 Cynthia Kop and Deivid Vale. Tuple Interpretations for Higher-Order Complexity. In Naoki Kobayashi, editor, *6th International Conference on Formal Structures for Computation and Deduction (FSCD 2021)*, volume 195 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 31:1–31:22, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.FSCD.2021.31.

42 Andras Kornai. Explicit finitism. *International Journal of Theoretical Physics*, 42:301–307, February 2003. doi:10.1023/A:1024451401255.

43 Jan Krajicek. *Bounded Arithmetic, Propositional Logic and Complexity Theory*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1995. doi:10.1017/CBO9780511529948.

44 Lawrence M. Krauss and Glenn D. Starkman. Universal limits on computation. Preprint arXiV., 2004. arXiv:astro-ph/0404510.

45 John Lamping. An algorithm for optimal lambda calculus reduction. In *Proceedings of the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '90, pages 16–30, New York, NY, USA, 1989. Association for Computing Machinery. doi:10.1145/96709.96711.

46 Jean-Luc Lehners and Jerome Quintin. A small universe, 2023. arXiv:2309.03272.

47 Jonathan Lenchner. A finitist's manifesto: Do we need to reformulate the foundations of mathematics?, 2020. arXiv:2009.06485.

48 Seth Lloyd. Ultimate physical limits to computation. *Nature*, 406(6799):1047–1054, August 2000. doi:10.1038/35023282.

49 Seth Lloyd. Computational capacity of the universe. *Physical review letters*, 88 23:237901, 2002.

50 José Pedro Magalhães, Atze Dijkstra, Johan Jeuring, and Andres Löh. A generic deriving mechanism for haskell. *SIGPLAN Not.*, 45(11):37–48, September 2010. doi:10.1145/2088456.1863529.

51 Ofra Magidor. Strict finitism refuted? *Proceedings of the Aristotelian Society*, 107(1pt3):403–411, 2007. doi:10.1111/j.1467-9264.2007.00230.x.

52 Per Martin-Löf. *Intuitionistic type theory*, volume 1 of *Studies in proof theory*. Bibliopolis, 1984.

53 Albert R. Meyer and Dennis M. Ritchie. The complexity of loop programs. In *Proceedings of the 1967 22nd National Conference*, ACM '67, pages 465–469, New York, NY, USA, 1967. Association for Computing Machinery. doi:10.1145/800196.806014.

54 Aran Nayebi. Practical intractability: A critique of the hypercomputation movement. *Minds and Machines*, 24:275–305, 2012. URL: https://api.semanticscholar.org/CorpusID:9328713.

55 Daniel Nolan. Lessons from infinite clowns (1st edition). In Karen Bennett and Dean Zimmerman, editors, *Oxford Studies in Metaphysics Vol. 14*. Oxford University Press, forthcoming.

56 B. Nordström. Terminating general recursion. *BIT*, 28(3):605–619, July 1988. doi:10.1007/BF01941137.

57 Cyrus Omar, Ian Voysey, Ravi Chugh, and Matthew A. Hammer. Live functional programming with typed holes. *Proc. ACM Program. Lang.*, 3(POPL), January 2019. doi:10.1145/3290327.

58 Christine Paulin-Mohring. Inductive definitions in the system coq - rules and properties. In Marc Bezem and Jan Friso Groote, editors, *Typed Lambda Calculi and Applications, International Conference on Typed Lambda Calculi and Applications, TLCA '93, Utrecht, The Netherlands, March 16-18, 1993, Proceedings*, volume 664 of *Lecture Notes in Computer Science*, pages 328–345. Springer, 1993. doi:10.1007/BFb0037116.

59 Rózsa Péter. Über die verallgemeinerung der theorie der rekursiven funktionen für abstrakte mengen geeigneter struktur als definitionsbereiche. *Acta Mathematica Academiae Scientiarum Hungaricae*, 12(3):271–314, May 1964. doi:10.1007/BF02023919.

60 Gordon D. Plotkin. A structural approach to operational semantics, 2004.

61 Karlis Podnieks. Towards a real finitism? Web page published online., December 2005. URL: http://www.ltn.lv/~podnieks/finitism.htm.

62 António G. Porto and Armando B. Matos. Ackermann and the superpowers. *SIGACT News*, 12(3):90–95, September 1980. doi:10.1145/1008861.1008872.

63 Pavel Pudlák. Incompleteness in the finite domain. *Bulletin of Symbolic Logic*, 23(4):405–441, 2017. doi:10.1017/bsl.2017.32.

64 Raphael M. Robinson. Primitive recursive functions. *Bulletin of the American Mathematical Society*, 53(10):p. 925–942, 1947. doi:bams/1183511140.

65 David Sands. *Calculi for time analysis of functional programs*. PhD thesis, University of London, 1990.

66 Vladimir Yu. Sazonov. On feasible numbers. In Daniel Leivant, editor, *Logic and Computational Complexity*, pages 30–51, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

67 Matthias Schirn and Karl-Georg Niebergall. Finitism = pra? on a thesis of w. w. tait. *Reports on Mathematical Logic*, January 2005.

68 Kurt Schütte. Predicative well-orderings. *Studies in logic and the foundations of mathematics*, 40:280–303, 1965. URL: https://api.semanticscholar.org/CorpusID:117343586.

69 Galina Iwanowna Sinkiewicz. On history of epsilontics. *Antiquitates Mathematicae*, 10(0):183–204, 2017. doi:10.14708/am.v10i0.805.

70 G. Sudan. Sur le nombre transfini ω^ω. *Bulletin Mathématique de la Société Roumaine des Sciences*, 30:11–30, 1927.

71 Yngve Sundblad. The ackermann function. a theoretical, computational, and formula manipulative study. *BIT Numerical Mathematics*, 11(1):107–119, March 1971. doi:10.1007/BF01935330.

72 Morten Heine b. Sørensen and Pawel Urzyczyn. *Lectures on the Curry-Howard isomorphism*. Studies in Logic and Foundations of Mathematics. Elsevier, 1998. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.17.7385.

73 Alfred Tarski, Andrzej Mostowski, and Raphael M. Robinson. Undecidable theories. *Philosophy*, 30(114):278–279, 1955.

74 A. S. Troelstra. *Constructivism in Mathematics: An Introduction*. Elsevier, 1988. URL: https://www.sciencedirect.com/bookseries/studies-in-logic-and-the-foundations-of-mathematics/vol/121/suppl/C.

75 Dirk Van Dalen. *Intuitionistic Logic*, pages 225–339. Springer Netherlands, Dordrecht, 1986. doi:10.1007/978-94-009-5203-4_4.

76 Herman Van Looy. A chronology and historical analysis of the mathematical manuscripts of gregorius a sancto vincentio (1584–1667). *Historia Mathematica*, 11(1):57–75, 1984. doi:10.1016/0315-0860(84)90005-3.

77 Nik Weaver. What is predicativism? Unpublished manuscript online., 2013. URL: https://api.semanticscholar.org/CorpusID:40352024.

78 Karl Weierstraß. *Rechnen mit komplexen Zahlen*, pages 25–44. Vieweg+Teubner Verlag, Wiesbaden, 1861-1862. doi:10.1007/978-3-663-06846-4_3.

79 Aleksandr S Yessenin-Volpin. The ultra-intuitionistic criticism and the antitraditional program for foundations of mathematics. In *Studies in Logic and the Foundations of Mathematics*, volume 60, pages 3–45. Elsevier, 1970.