# Consistent Ultrafinitist Logic

Michał J. Gajda

Migamake Pte Ltd
mjgajda@migamake.com

Ultrafinitism(Kornai 2003; Podnieks 2005; Yessenin-Volpin 1970; Gefter 2013; Lenchner 2020) postulates that we can only reason and compute relatively short objects(Seth Lloyd 2000; Krauss and Starkman 2004; Sazonov 1995; S. Lloyd 2002; Gorelik 2010), and numbers beyond certain value are not available. Some philosophers also question the physical existence of real numbers beyond a certain level of accurracy(Gisin 2019). This approach would also forbid many forms of infinitary reasoning and allow removing many from paradoxes stemming from a countable enumeration.

However, philosophers still disagree on whether such a finitist logic could be consistent(Magidor 2007), while constructivist mathematicians claim that *"no satisfactory developments exist"*(Troelstra 1988). We present preliminary work on a proof system based on Curry-Howard isomorphism(Howard 1980) and explicit bounds for computational complexity.

This approach invalidates logical paradoxes that stem from a profligate use of transfinite reasoning(Benardete 1964; Nolan forthcoming; Schirn and Niebergall 2005), and assures that we only state problems that are decidable by the limit on input size, proof size, or the number of steps(Tarski, Mostowski, and Robinson 1955).

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(Vale and Kop 2021).

## 1 Introduction

By *finitism* we understand the mathematical logic that tries to absolve us from transfinite inductions(Kornai 2003). *Ultrafinitism* goes even further by postulating a definite limit for the complexity of objects that we can compute with(Seth Lloyd 2000; Krauss and Starkman 2004; Sazonov 1995; S. Lloyd 2002; Gorelik 2010). We assume these without committing to a particular limit.

In order to permit only *ultrafinitist* inferences, we postulate *ultraconstructivism*: we permit proofs, or constructions that are not just strictly computable, but for which there is a bound on amount of computation that is needed to resolve them. That means that we forbid proofs that go for an arbitrarily long time and require a *deadline* 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 *depths of input terms*, and outputs the upper bound on the number of steps that the proof is permitted to make. *Depths of input terms* are a convenient upper bound on the complexity of normalized proof terms (those without the cut.)

## 2 Syntax and inference rules

| | | | |
|---|---|---|---|
| Size variables: | $v$ | $\in$ | $V$ |
| Term variables: | $x$ | $\in$ | $X$ |
| Positive naturals: | $i$ | $\in$ | $\mathbb{N} \setminus \{0\}$ |
| Polynomials: | $\rho$ | $::=$ | $v \mid i \mid \rho + \rho \mid \rho * \rho \mid \rho^{\rho} \mid iter(\rho, \rho, v) \mid \rho[\![^{v}/_{\rho}]\!]$ |
| Data size bounds: | $\alpha$ | $::=$ | $\rho$ |
| Computation bounds: | $\beta$ | $::=$ | $\rho$ |
| Types: | $\tau$ | $::=$ | $v \mid \tau \wedge \tau \mid \tau \vee \tau \mid \forall x_v : \tau \rightarrow^{\alpha}_{\beta} \tau \mid \bot \mid \circ$ |
| Terms: | $E$ | $::=$ | $v \mid \lambda v.E \mid in_r(E) \mid in_l(E) \mid (E, E) \mid ()$ |
| | | $\mid$ | $case\, E\, of \begin{array}{lll} in_l(v) & \rightarrow & E; \\ in_r(v) & \rightarrow & E; \end{array}$ |
| Environments: | $\Gamma$ | $::=$ | $v_1 : {\tau^1}_{\beta_1}, ..., {\tau^n}_{\beta_n}$ |
| Judgements: | $J$ | $::=$ | $\Gamma \vdash^{\alpha}_{\beta} E : \tau$ |

Here $\rho^{\rho}$ is an exponentation, and $iter(\rho_1, \rho_2, \rho_3)v$ is an iterated composition of function described by expression $\rho_1$ with respect to an argument variable $v$. The iteration happens $\rho_2$ times. The $\rho_1[\![^{v}/_{\rho_2}]\!]$ describes substitution, inside of $\rho_1$, of all instances of bound variable $v$ with $\rho_2$.

The polynomials will be standing on one of two roles: as an upper bound on the proof complexity, and there we will use symbol $\alpha$ as a placeholder, or to state an upper bound on the number of constructors in the proof indicated by the symbol $\beta$. That is because the number of constructors may sometimes bound a recursive examination of the proof of a proposition.

Notation $\forall x_v : A \implies {}^{\alpha(v)}_{\beta(v)} B$ binds proof variable $x$ with type of $A$, and then bound in polynomials $\alpha(v)$ for complexity and $\beta(v)$ for depth of the normalized term.

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

### 2.1 Inference rules

$$\frac{\Gamma \vdash^{\alpha}_{\beta} y_{\beta} : A \quad v \in V}{\Gamma, x_{\beta} : A \vdash^{1}_{\beta} x : A}\ var \qquad \frac{}{\Gamma \vdash^{1}_{\beta} () : \circ}\ unit$$

$$\frac{\Gamma \vdash^{\alpha_1}_{\beta_1} a^1 : A^1 \quad \Gamma \vdash^{\alpha_2}_{\beta_2} a^2 : A^2}{\Gamma \vdash^{\alpha_1+\alpha_2}_{\max(\beta_1,\beta_2)+1} (a^1, a^2) : A^1 \wedge A^2}\ pair \qquad \frac{\Gamma \vdash^{\alpha}_{\max(\beta_1,\beta_2)} e : A^1 \wedge A^2 \quad i \in \{1,2\}}{\Gamma \vdash^{\alpha+1}_{\beta-1} prj_i\ \ e : A^i}\ prj_i$$

$$\frac{\Gamma \vdash^{\alpha}_{\beta} e : A^i \quad i \in \{l, r\}}{\Gamma \vdash^{\alpha+1}_{\beta+1} in_i(e) : A^1 \vee A^2}\ inj \qquad \frac{\Gamma \vdash^{\alpha_1}_{\beta_1} e : A \qquad \alpha_1 \leq \alpha_2 \qquad \beta_1 \leq \beta_2}{\Gamma \vdash^{\alpha_2}_{\beta_2} e : A}\ subsume$$

$$\frac{\Gamma \vdash^{\alpha_\vee}_{\beta_\vee} a : A^1 \vee A^2 \qquad \Gamma, x : {A^1}_{\beta_\vee - 1} \vdash^{\alpha_1}_{\beta_1} b : B \qquad \Gamma, y : {A^2}_{\beta_\vee - 1} \vdash^{\alpha_2}_{\beta_2} c : B}{\Gamma \vdash^{\alpha_\vee + max(\alpha_1,\alpha_2)+1}_{\max(\beta_1,\beta_2)} case\, a\, of \begin{array}{lll} in_l(x) & \rightarrow & b; \\ in_r(y) & \rightarrow & c; \end{array} : B}\ case$$

$$\frac{\Gamma, x_v : A \vdash^{\alpha(v)}_{\beta(v)} e : B}{\Gamma \vdash^{\alpha(1)+1}_{\beta(1)+1} \lambda x.e : \forall a_v : A {\to}^{\alpha(v)}_{\beta(v)} B}\ abs \qquad \frac{\Gamma \vdash^{\alpha_1}_{\beta_1} e : \forall a : A_v {\to}^{\alpha_2(v)}_{\beta_2(v)} B \quad \Gamma \vdash^{\alpha_3}_{\beta_3} a : A}{\Gamma \vdash^{\alpha_1+\alpha_2(\beta_3)+\alpha_3}_{\beta_2(\beta_3)} e\ a : B}\ app$$

Please note that notation $\forall x_v : A \to^{\alpha(v)}_{\beta(v)} B$ has a size variable $v$ declared as a depth of term variable $x$, and then bound in polynomials $\alpha(v)$ and $\beta(v)$ The notation $\alpha(1)$ is a shortcut for $\alpha[\![{}^1/v]\!]$ in the rules *abs* and *app*.

With the exception of *bound*, and *rec* these are all reinterpretations of rules for intuitionistic logic(Brouwer 1981; Van Dalen 1986; Sørensen and Urzyczyn 1998), enriched with bounds on the proof length $\alpha$ and normalized term depth $\beta$.

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:

$$\frac{\Gamma \vdash^{\alpha_1}_{\beta_1} f : A_v {\to}^{\alpha_2(v_1)}_{\beta_2(v_2)} A \quad \Gamma \vdash^{\alpha_3}_{\beta_3} k : B \quad \Gamma \vdash^{\alpha_4}_{\beta_4} a : A}{\Gamma \vdash^{\alpha_1+\alpha_3+iter(\alpha_2,\beta_3,v_1)[\![{}^{\beta_4}/v_1]\!]+\alpha_4}_{\beta_1[\![{}^{iter(\beta_2,\beta_3,v_2)[\![{}^{\beta_4}/v_2]\!]}/v]\!]} rec(f,k,a) : B}\ rec$$

Here the depth of the term must decrease at each step of the recursion.

## 2.2 Simplifying bound polynomials

Our inference rules rely on computing polynomial 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.

$$\begin{array}{lrcl}
(1) & a * x^e + b * x^f & \leq & (a+b) * x^f \\
(2) & a * x^e * y^g & \leq & a * x^f * y^h \\
(3) & iter(e, g, x) & \leq & iter(f, h, x) \\
(4) & iter(a * x, e, x) & = & a^e * x \\
(5) & iter(x + a, e, x) & = & x + a * e \\
(6) & iter(x^e, g, x) & = & x^{e^g}
\end{array}$$

We may thus use these rules to loosen the bound in such a way as to reduce the size of the polynomial 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 polynomials as the holes to be filled by the framework interpreter.

## 2.3 Reduction

Reduction relation is defined as small step semantics(Plotkin 2004) in order to preserve number of computational steps made over the course of evaluation.

$$\frac{}{case\, in_l(a)\, of \begin{array}{ccc} in_l(x) & \rightarrow & b; \\ in_r(y) & \rightarrow & c; \end{array} \Longrightarrow_1 b[\![{}^a/_x]\!]}\ eval-case-left$$

$$\frac{}{case\, in_r(a)\, of \begin{array}{ccc} in_l(x) & \rightarrow & b; \\ in_r(y) & \rightarrow & c; \end{array} \Longrightarrow_1 c[\![{}^a/_x]\!]}\ eval-case-right$$

$$\frac{occurs(x,e)=k}{(\lambda x.e)f \Longrightarrow_k e[\![{}^f/_x]\!]}\ eval-app \qquad \frac{e \Longrightarrow_n e' \quad i \in \{l,r\}}{in_i(e) \Longrightarrow_1 in_i(e')}\ eval-sum$$

$$\frac{}{prj_r(a,b) \Longrightarrow_1 a}\ eval-prr \qquad \frac{}{prj_l(a,b) \Longrightarrow_1 b}\ eval-prl$$

$$\frac{a_l \Longrightarrow_n a_l'}{(a_l,a_r) \Longrightarrow_n (a_l',a_r)}\ eval-pairleft \qquad \frac{a_r \Longrightarrow_n a_r'}{(a_l,a_r) \Longrightarrow_n (a_l,a_r')}\ eval-pairright$$

## 3 Sketching the proofs

### 3.1 Consistency

After elision of bounds and rule *subsume* we see the rules for intuitionistic logic. Thus consistency can be proved by the consistency of intuitionistic logic(Brouwer 1981; Van Dalen 1986; Sørensen and Urzyczyn 1998).

### 3.2 Strong normalization

Please note that all rules increase polynomial bound on computation effort: $\alpha$. Thus if we are able to infer a judgement with a valid bound, we get a bounded judgement.

The depth bound $\beta$ is used for delimiting recursion depth since it has to decrease at each stage of loop application *rec*.

### 3.3 Valid proposition with a fixed bound

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 $n$.

### 3.4 Expressivity

It is easy to show that our logic can emulate bounded loop programs(Meyer and Ritchie 1967) which have power equivalent to primitive recursive functions(Robinson 1947).

One could muse that this class does not cover all Bounded Turing Machine(Hopcroft and Ullman 1968) programs. In order to support these, we would need to define more general bounding functions.

One can replace iterated polynomials with arbitrary bounding functions, as long as they are additive and substitutable. 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(Denning, Dennis, and Qualitz 1978), which means that estimating our complexities would become an impossibly long endeavour.

To give an example of simplified Ackermann function which is the best known example of function beyond PRA, evaluation takes $A(5) = 2^{2^{2^{2^{16}}}} - 3$(Knuth, Ellerman, and Consigli 2016). That means that these evaluations quickly get out of hand and indeed outside of any reasonable limits.

## 4 Related work

Note that the problem with transfinite arguments has been spotted long before(Kornai 2003; Podnieks 2005; Yessenin-Volpin 1970; Gefter 2013; Lenchner 2020). Automatic theorem provers like Coq require a bounding function for each inductive definition(Nordström 1988; Paulin-Mohring 1993; Bertot and Komendantsky 2008). This bounding function is required to monotonically decrease with each recursive call. This makes all recursive definitions *well-founded*(Nordström 1988).

However, the bounding function may be very large, and impossible to compute within reasonable time. Cost calculi for functional languages attempt to assign cost to certain operations in order to reason about time and space complexity (Sands 1990). They do not require all proofs and propositions to carry the cost as we do.

Philosophers have long postulated distinction between feasible computations and unfeasible ones(Yessenin-Volpin 1970), however it was considered unclear whether it is possible to realize this distinction on the basis of a logic(Troelstra 1988), with some claiming that such a logic could not be consistent(Magidor 2007).

There exist logics that implicitly constrain computational complexity of the proofs, for example Bounded Arithmetic(Krajicek 1995) that is restricted to computations in polynomial time. However, most of them are significantly weaker than class of primitive recursive functions which is 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.

## 5 Future work

In future work, we will attempt to characterize the realm of ultrafinitist theorems. For example, the famous undecidability example that refers to an input and a program of an arbitrary length seems impossible to define in this framework. In particular any program bounded by $\alpha(x)$ for any input $\beta(x)$ might be checked for termination within a significantly larger but still bounded limit of $\alpha^{\beta(x)}(x)$. Another avenue of future work would be to define a full type theory, dependently typed language and an automatic prover for these inference rules. Since polynomials $\alpha(x)$ are monotonic, it is likely that the proof strategy will be also bounded in this case. We thus aim to prove that bounded proofs invalidate transfinite arguments like Cantor's diagonalization.

## 6 Discussion

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(Gorelik 2010)[1] 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(Krajicek 1995), which capture polynomial time hierarchy.

Please note that we avoid any *implicit* treatment of computational complexity here so that the power of the logic is *not* constrained implicitly.

## Acknowledgments

We thank Vendran Čačić and Orestis Melkonian for their invaluable feedback.

## References

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

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

Brouwer, L. E. J. 1981. *Over de Grondslagen Der Wiskunde.* Vol. 1. MC Varia. Amsterdam: Mathematisch Centrum.

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

Gefter, Amanda. 2013. "Mind-Bending Mathematics: Why Infinity Has to Go." *New Scientist* 219 (2930): 32–35. https://doi.org/https://doi.org/10.1016/S0262-4079(13)62043-6.

Gisin, Nicolas. 2019. "Indeterminism in Physics, Classical Chaos and Bohmian Mechanics. Are Real Numbers Really Real?" https://arxiv.org/abs/1803.06824.

Gorelik, Gennady. 2010. "Bremermann's Limit and cGh-Physics." https://arxiv.org/abs/0910.3424.

Hopcroft, John E., and Jeffrey D. Ullman. 1968. "Relations Between Time and Tape Complexities." *J. ACM* 15 (3): 414–27. https://doi.org/10.1145/321466.321474.

Howard, William A. 1980. "The Formulae-as-Types Notion of Construction." In *To h.b. Curry: Essays on Combinatory Logic, λ-Calculus and Formalism*, edited by J. Hindley and J. Seldin, 479–90. Academic Press.

Knuth, Don, Frank Ellerman, and Natan Arie Consigli. 2016. "A046859: Simplified Ackermann Function (Main Diagonal of Ackermann-Péter Function)." Edited by N. J. A. Sloane. 2016. https://oeis.org/A046859.

Kornai, Andras. 2003. "Explicit Finitism." *International Journal of Theoretical Physics* 42 (February): 301–7. https://doi.org/10.1023/A:1024451401255.

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

Krauss, Lawrence, and Glenn Starkman. 2004. "Universal Limits on Computation," May.

Lenchner, Jonathan. 2020. "A Finitist's Manifesto: Do We Need to Reformulate the Foundations of Mathematics?" https://arxiv.org/abs/2009.06485.

Lloyd, S. 2002. "Computational Capacity of the Universe." *Physical Review Letters* 88 23: 237901.

[1]A computation run by computer the size of Earth within the lifespan of Earth so far. Of the order of $10^{93}$.

Lloyd, Seth. 2000. "Ultimate Physical Limits to Computation." *Nature* 406 (6799): 1047–54. https://doi.org/10.1038/35023282.

Magalhães, José Pedro, Atze Dijkstra, Johan Jeuring, and Andres Löh. 2010. "A Generic Deriving Mechanism for Haskell." *SIGPLAN Not.* 45 (11): 37–48. https://doi.org/10.1145/2088456.1863529.

Magidor, Ofra. 2007. "Strict Finitism Refuted?" *Proceedings of the Aristotelian Society* 107 (1pt3): 403–11. https://doi.org/10.1111/j.1467-9264.2007.00230.x.

Meyer, Albert R., and Dennis M. Ritchie. 1967. "The Complexity of Loop Programs." In *Proceedings of the 1967 22nd National Conference*, 465–69. ACM '67. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/800196.806014.

Nolan, Daniel. forthcoming. "Send in the Clowns." In *Oxford Studies in Metaphysics*, edited by Karen Bennett and Dean Zimmerman. Oxford: Oxford University Press.

Nordström, B. 1988. "Terminating General Recursion." *BIT* 28 (3): 605–19. https://doi.org/10.1007/BF01941137.

Paulin-Mohring, Christine. 1993. "Inductive Definitions in the System Coq - Rules and Properties." In *Typed Lambda Calculi and Applications, International Conference on Typed Lambda Calculi and Applications, TLCA '93, Utrecht, the Netherlands, March 16-18, 1993, Proceedings*, edited by Marc Bezem and Jan Friso Groote, 664:328–45. Lecture Notes in Computer Science. Springer. https://doi.org/10.1007/BFb0037116.

Plotkin, Gordon D. 2004. "A Structural Approach to Operational Semantics."

Podnieks, Karlis. 2005. "Towards a Real Finitism?" 2005. http://www.ltn.lv/~podnieks/finitism.htm.

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

Sands, David. 1990. "Calculi for Time Analysis of Functional Programs." PhD thesis, University of London.

Sazonov, Vladimir Yu. 1995. "On Feasible Numbers." In *Logic and Computational Complexity*, edited by Daniel Leivant, 30–51. Berlin, Heidelberg: Springer Berlin Heidelberg.

Schirn, Matthias, and Karl-Georg Niebergall. 2005. "Finitism = PRA? On a Thesis of w. W. Tait." *Reports on Mathematical Logic*, January.

Sørensen, Morten Heine B., and Pawel Urzyczyn. 1998. "Lectures on the Curry-Howard Isomorphism." http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.17.7385.

Tarski, Alfred, Andrzej Mostowski, and Raphael M. Robinson. 1955. "Undecidable Theories." *Philosophy* 30 (114): 278–79.

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

Vale, Deivid, and Cynthia Kop. 2021. "Tuple Interpretations for Higher-Order Rewriting." *CoRR* abs/2105.01112. https://arxiv.org/abs/2105.01112.

Van Dalen, Dirk. 1986. "Intuitionistic Logic." In *Handbook of Philosophical Logic: Volume III: Alternatives in Classical Logic*, edited by D. Gabbay and F. Guenthner, 225–339. Dordrecht: Springer Netherlands. https://doi.org/10.1007/978-94-009-5203-4_4.

Yessenin-Volpin, Aleksandr S. 1970. "The Ultra-Intuitionistic Criticism and the Antitraditional Program for Foundations of Mathematics." In *Studies in Logic and the Foundations of Mathematics*, 60:3–45. Elsevier.

# Appendix: Facilities

Here we discuss possible extensions that will make it easier to use the framework.

## Inductive types

Inductive types can be encoded with finite pairs and sums above, but we can also have mutually recursive declarations of the form:

$$I ::= data\ \ t\ \tau^* = [c\ a^*]^+$$

Where:

- $t \in T$ is a type identifier,
- $\tau = V$ is a type variable,
- $c \in C$ is a constructor identifier,
- $a ::= \tau \,\big|\, t\ a^*$ which means that argument to a constructor is either a type variable, or a type constructor and a list of arguments.

For each type constructor we have a fixed arity of type arguments, and ditto for any constructor $c$ for this type. We also need to compute a *minimum depth of the type*, which is a minimum of *minimal depths for each constructor*. Minimal depth for each constructor is a maximum of all its arguments, or 1 if there are no arguments. For a set of data types to be valid, they have to all possess a *minimum depth* that is a finite number.

Please note that this definition explicitly forbids inductive types that are empty, or do not have a finite construction.

$$\frac{\Gamma_c \vdash^{\alpha_\vee}_{\beta_\vee} a : I\ a^1\ ...\ a^n \qquad \Gamma_c \vdash^{\beta_1(v)}_{e_1 : B} \quad ... \quad \Gamma_c \vdash^{\alpha_n(v)}_{\beta_n(v)} e_n : B}{\Gamma_c \vdash^{\alpha_\vee + max_k(\alpha_1(\beta_{\vee-1}),...,\alpha_n(\beta_{\vee-1}))}_{\max(\beta_1(\beta_{\vee-1}),...,\beta_n(\beta_{\vee-1}))} case\,a\,of \begin{array}{ccc} C_1\ r^{1,1}\ ...r^{1,k_1} & \rightarrow & e_1; \\ ... & & \\ C_n\ r^{n,1}\ ...r^{n,k_n} & \rightarrow & e_n; \end{array} : B}\ case$$

## Example terms

$$\begin{array}{rcll} \text{Nat}_\beta & = & rec(\circ \vee x, \beta, x) \circ & \\ \text{zero} & = & in_l(()) & :^1_1 \quad \text{Nat}_1 \\ \text{succ} & = & \lambda x_v \rightarrow^1_{v+1} in_r(x) & :^1_{v+1} \quad \text{Nat}_v \rightarrow Nat_{v+1} \\ induction & = & \lambda(s_v : A \rightarrow^{\alpha_s(v)}_{\beta_s(v)} A) \rightarrow \lambda(b_w : A_w) \rightarrow \lambda(n_u : Nat_u) \rightarrow rec(s, b, n) & \end{array}$$

## Metareasoning

We may now encode the type constructors:

$$\begin{array}{lcl} [\![A \vee B]\!] & = & in_l(in_l(([\![A]\!], [\![B]\!]))) \\ [\![A \wedge B]\!] & = & in_l(in_r(([\![A]\!], [\![B]\!]))) \\ [\![\forall x_v : A.B]\!] & = & in_r(in_l((\lambda x : A.[\![B]\!], ([\![\alpha]\!], [\![\beta]\!])))) \\ [\![\circ]\!] & = & in_r(in_r(())) \end{array}$$

This encoding allows us to make operations on types akin to generic programming in Haskell(Magalhães et al. 2010)

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

Note that every type term in the *normal form* is longer than its own type.
In order to encode types, we also need to encode polynomials:

$$\begin{array}{lcl}
[\![v]\!] & = & in_l(in_l(in_l(\mathbb{B}\,(v)))) \\
[\![i]\!] & = & in_l(in_l(in_r(i))) \\
[\![\rho_1 + \rho_2]\!] & = & in_l(in_r(in_l(([\![\rho_1]\!], [\![\rho_2]\!])))) \\
[\![\rho_1 * \rho_2]\!] & = & in_l(in_r(in_r(([\![\rho_1]\!], [\![\rho_2]\!])))) \\
[\![\rho_1^{\rho_2}]\!] & = & in_r(in_l(in_l(([\![\rho_1]\!], [\![\rho_2]\!])))) \\
[\![iter(\rho_1, \rho_2, v, ]\!],) & = & in_r(in_l(in_r(([\![\rho_1]\!], ([\![\rho_2]\!], [\![v]\!]))))) \\
[\![\rho_1\ [\![^v/_{\rho_2}]\!]]\!] & = & in_r(in_r(in_l(([\![\rho_1]\!], ([\![\rho_2]\!], \mathbb{B}\,(v))))))
\end{array}$$

Above we assume that $\mathbb{B}\,(v)$ is de Brujin index of the variable.