Zaionc paradox revisited

Pierre Lescanne

Zaionc paradox revisited (Part I)

Pierre Lescanne

University of Lyon, École Normale Supérieure de Lyon,
LIP (UMR 5668 CNRS ENS Lyon UCBL),
46 allée d'Italie, 69364 Lyon, France
pierre.lescanne@ens-lyon.fr

Abstract

Canonical expressions are representative of implicative propositions up-to renaming of variables. In this paper we explore, using a Monte-Carlo approach, the model of canonical expressions in order to confirm the paradox that says that *asymptotically almost all classical theorems are intuitionistic*.

Keywords: intuitionistic logic, classical logic, combinatorics, asymptotic, random generation, Bell number, Catalan number, Monte-Carlo method

1 Introduction

In 2007, Marek Zaionc coauthored two papers [7, 5], corresponding to two models of the calculus of implicative propositions and presenting the following paradox, namely that *asymptotically almost all classical theorems are intuitionistic*, which is called here *Zaionc paradox*. In the current paper, we focus on the model of [7], which we call *canonical expressions*. They have been introduced by Genitrini, Kozik and Zaionc [7] and more recently by Tarau and de Paiva [13, 14]. A canonical expression is a representative of a class of implicative expressions that differ only by the name assigned to the variables. Whereas Genitrini, Kozik and Zaionc addressed the mathematical aspect of this model, Tarau and de Paiva tried to explicitly generate all the canonical expressions of a given size and faced up to combinatorial explosion, because canonical expressions grow super exponentially in size. In this paper, I check experimentally Zaionc paradox, adopting a Monte-Carlo approach to observe how this paradox emerges. Indeed I designed a linear algorithm to randomly generate canonical expressions. Therefore I can consider large samples of random canonical expressions and count how many canonical expressions in that samples are intuitionistic theorems or classical theorems. The experiments, centred around canonical expressions of size 100, show that the numbers we get for both sets are very close confirming experimentally the paradox. As a by product we obtain programs generating large random canonical expressions, large random intuitionistic theorems or large random classical theorems.

The programs used in this paper can be found on GitHub.

2 Intuitionistic vs classical theorems

In this paper we deal only with implicative propositions. An implicative proposition is a binary expression with propositional variables, which has only one binary operator namely the implication written $\to$. This can be seen as the type of a function in functional programming or in λ-calculus. Among the implicative propositions, some can be proven, using a proof system. Let us consider natural deduction. There are three rules used to prove *intuitionistic theorems*.

$$\mathsf{Axiom}\ \frac{}{\alpha \vdash \alpha} \qquad \to\text{-}\mathsf{Elim}\ \frac{\Gamma \vdash \alpha \to \beta \qquad \Gamma \vdash \alpha}{\Gamma \vdash \beta} \qquad \to\text{-}\mathsf{Intro}\ \frac{\alpha, \Gamma \vdash \beta}{\Gamma \vdash \alpha \to \beta}$$

Classical theorems are proved by adding the axiom:

$$\mathsf{Pierce}\ \frac{}{\vdash ((\alpha \to \beta) \to \alpha) \to \alpha}$$

which is called *Peirce law*. Usually one uses valuations, which assign booleans to variables. Let ρ be an assignment of booleans to variables. Valuations of expressions are defined by:

$$\begin{array}{rcl} [\![x]\!]_\rho & = & \rho(x) \\ [\![e \to e']\!]_\rho & = & [\![e']\!]_\rho \vee \overline{[\![e]\!]_\rho} \end{array}$$

where $b \mapsto \overline{b}$ is the negation. An expression e is a classical theorem or a tautology, if for all valuation ρ, $[\![e]\!]_\rho = \mathsf{True}$.

3 The model of canonical expressions

We call *canonical expression* the representative of an equivalence class of binary expressions up-to renaming of variables. In other words, a canonical expression is a binary expression, in which variables are named canonically, from right to left. That means that the rightmost variable is x_0, then if processing to the left, the next new variable is x_1, then the next new variable, which is neither x_0 nor x_1 is x_2 etc. Recall that in an expression, a variable corresponds to a position into the expression. In other words a variable in an equivalence class of positions. Therefore naming canonically a variable corresponds to naming canonically an equivalence class in the set of position. Therefore if a variable belongs to the i^{th} class it will be named α_i and vice-versa, if a class is the class of α_i, it is the i^{th} class. In canonical expressions, the classes are numbered from right to left. For instance, assume an expression of size 10, i.e., with 10 occurrences of variables. This is an expression with 10 positions of variables, like :

$$x \;\; y \;\; y \;\; x \;\; y \;\; x \;\; z \;\; x \;\; x \;\; x$$

or

$$\beta \;\; \alpha \;\; \alpha \;\; \beta \;\; \alpha \;\; \beta \;\; \gamma \;\; \beta \;\; \beta \;\; \beta$$

In the first expression we see 3 variables namely $\{x, y, z\}$, hence 3 congruence classes. As said above, for technical reasons, not hard to guess, variables are numbered from right to left, starting at 0. Hence x which corresponds to positions $\{1, 4, 6, 8, 9, 10\}$ is class 0, z which corresponds to positions $\{7\}$ is class 1 and y which corresponds to positions $\{2, 3, 5\}$ is class 2. Therefore the list of variables canonically named associated with the above list of variables is.

$$\alpha_0 \;\; \alpha_2 \;\; \alpha_2 \;\; \alpha_0 \;\; \alpha_2 \;\; \alpha_0 \;\; \alpha_1 \;\; \alpha_0 \;\; \alpha_0 \;\; \alpha_0$$

The above congruence class is canonically represented by the string 0220201000. Notations for canonically representing congruence classes over a set of n elements by strings of natural numbers of size n are known . They are called *restricted growth strings* by Knuth ([8], fascicle 3, §7.2.1.5, p. 62) and *irregular staircases* by Flajolet and Sedgewick [4] (p. 62-63). In this paper, we consider them from right to left wherever the cited authors consider them from left to right, but this is a detail. Intuitively, a restricted growth string is a string whose last item is 0 and when one progresses to the left, one meets items that have been met already or if not the new item is just the successor of the largest item met until this point. Here $[0..i]^*$ is the set of strings made of integers k such that $0 \leq k \leq i$.

Definition 1 (Restricted growth string)**.** *The set* $\mathcal{W}_n$ *of* n-restricted right to left growth strings *is defined as follows*

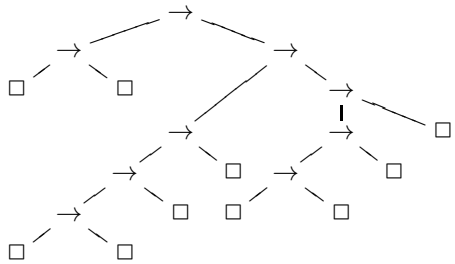
- $\mathcal{W}_0 = [0..0]^*$,
- $\mathcal{W}_{n+1} = [0..(n+1)]^*\,(n+1)\,\mathcal{W}_n$

For instance $W_2 = [0..2]^*\,2\,\mathcal{W}_1 = [0..2]^*\,2\,[0..1]^*\,1\,\mathcal{W}_0 = [0..2]^*\,2\,[0..1]^*\,1\,[0..0]^*$ One sees that $0220201000 \in \mathcal{W}_2$ where items larger that those on the right are put in red.

Once the variables are chosen, how operators $\to$ are associated has to be done. Here we are interested in parenthesised expressions with the only binary operator $\to$. For instance, for an expression of size 10, we look for a binary tree with 10 external leaves, we get

$$((\square\to\square)\to((((\square\to\square)\to\square)\to\square)\to(((\square\to\square)\to\square)\to\square)))$$

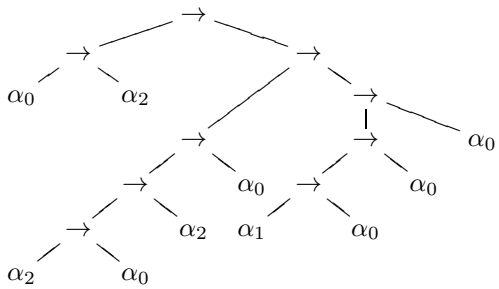
which can be drawn as the tree :



To get a canonical expression one matches a restricted growth string and a binary tree. In our case, we get by matching the above restricted tree and the above parenthesised expression, the following canonical expression

$$((\alpha_0 \to \alpha_2) \to ((((\alpha_2 \to \alpha_0) \to \alpha_2) \to \alpha_0) \to (((\alpha_1 \to \alpha_0) \to \alpha_0) \to \alpha_0)))$$

which corresponds to the tree:



Canonical expressions are therefore pairs of binary tree and restricted left to right growth strings, counted by $K_n = C_{n-1}\varpi_n$ where C_n are *Catalan numbers* (counting binary trees) and ϖ_n are *Bell numbers* (counting restricted growth strings). This corresponds to sequence A289679 in the *Online encyclopedia of integer sequences* [11]. Asymptotically,

$$C_{n-1} \sim \frac{4^{n-1}}{\sqrt{\pi(n-1)^3}}$$

$$\varpi_n \sim n! \frac{e^{e^r-1}}{r^n \sqrt{2\pi r(r+1)e^r}}$$

where $r \equiv r(n)$ is the positive root of the equation $re^r = n+1$. Therefore

$$K_n \quad \sim \quad n! \frac{4^{n-1} e^{e^r-1}}{\pi \sqrt{2(n-1)^3 r(r+1)e^r}}$$

The first values of K_n are $1, 2, 10, 75, 728, 8526, 115764, 1776060, 30240210, ...$ and $K_{100} \sim 9, 62.10^{168}$.

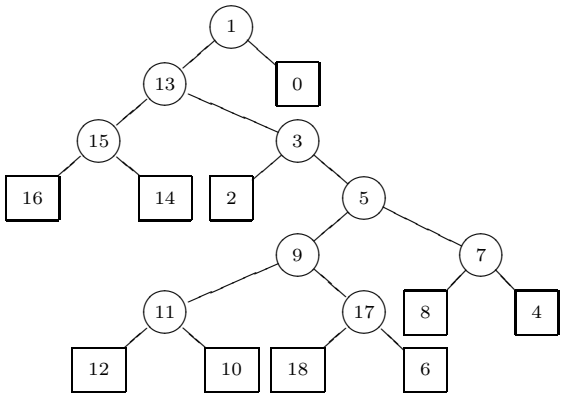
4 Random canonical expressions

Since canonical expressions are pairs of well-known combinatorial objects, namely binary trees and congruence classes, we can use well-known algorithm to generate each constituents of the pairs.

4.1 Random binary trees

For generating random binary trees, I use *Rémy algorithm* [10] which is linear. This algorithm is described by Knuth in [8] § 7.2.1.6 (pp. 18-19). I have taken his implementation. The idea of the algorithm is that a random binary tree can be built by iteratively and randomly picking an internal node or a leaf in a random binary tree and inserting an new internal node and a new leaf either on the left or on the right. A binary tree of size n has $n-1$ internal nodes and n leaves. Inserting a node in a binary tree of size n requires throwing randomly a number between 1 and $4n-2$ (a random number between 0 and $4n-3$ in my Haskell implementation). This process can be optimised by representing a binary tree as a list (a vector in Haskell), an idea sketched by Rémy and described by Knuth. In this vector, even locations are for internal nodes and odd locations are for leaves. Here is a vector representing a binary tree with 10 leaves and its drawing.

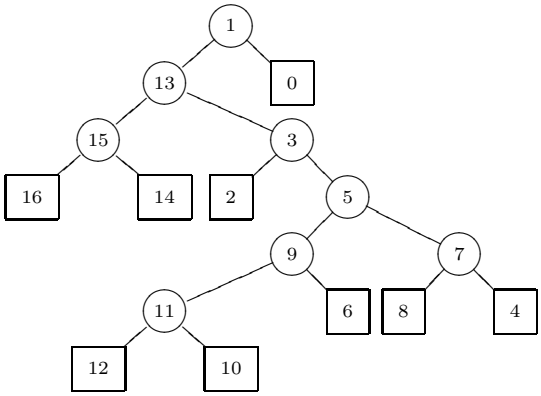
indices	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
values	1	13	0	2	5	9	7	8	4	11	17	12	10	15	3	16	14	18	6



This tree was built by inserting the node 17 together with the leaf 18 in the following tree.

indices	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
values	1	13	0	2	5	9	7	8	4	11	6	12	10	15	3	16	14

coding the tree



This was done by picking a node (internal node or leaf) and by inserting above this node a new internal node (labelled 17) and a new leaf (labelled 18). The new leaf is attached to the new node. This double action (inserting the internal node and attaching the leaf) is done by choosing a number in the interval [0..33] (in general, in the interval $[0..(4n-3)]$). Assume that in this case the random generator returns 21. 21 contains two informations : its parity (a boolean) and its half. Half of 21 is 10, which tells that the new node 17 must be inserted above the 10^{th} (in the array) node namely 6. Since 21 is odd, the rest of the tree (here reduced to the leaf 6) is inserted on the right (otherwise it would be inserted on the left). A new leaf 18 is inserted on the left (otherwise it would be inserted on the right).

The algorithm works as follows. If $n = 0$, Rémy's algorithm returns the vector starting with 0 and filled with anything, since the whole algorithm works on the same vector with the same size. In general, say that, for $n-1$, Rémy's algorithm returns a vector v. One picks a random integer x between 0 and $4n-3$. Let k be half of x. In the vector v one replaces the k^{th} position by $2n-1$ and one appends two elements, namely the k^{th} item of v followed by $2n$ if x is even and $2n$ followed by the k^{th} item of ℓ if x is odd.

If we admit that given a `seed` and a positive integer `n`, `randForRemy seed n` returns a random integer between 0 and $4n-3$ inclusive, the program in Haskell of the function `rbtL` which yields a random binary tree of size n coded as a vector of length $2n$ is

```
rbtV :: Int -> Int -> Vector Int
rbtV seed 0 = Data.Vector.replicate sizeOfVector (-1) // [(0,0)]
rbtV seed n =
  let x = randForRemy seed n -- a random value between 0 and 4n-3 inclusive
      v = rbtV seed (n-1)
      k = x `div` 2
  in case even x of
    True -> v // [(k,2*n-1),(2*n-1,v!k),(2*n,2*n)]
    False -> v // [(k,2*n-1),(2*n-1,2*n),(2*n,v!k)]
```

4.2 Random restricted growth string

For generating random partitions or random restricted growth strings an algorithm due to A. J. Stam [12] and described by Knuth in [9] § 7.2.1.3 (p. 74) was implemented. The implementation requires, for each value of n (the size of the underlying set – for us, this is number of

variables or the size of the expression –), a preliminary construction of a table of reals in which indices are looked up (the number M of classes). Those reals are probabilities

$$\frac{m^n}{em!\varpi_n}$$

that an n-partition has m classes. Thus in my program, I implemented the algorithm for size $n = 10, 25, 50\ 100, 500$ and 1000. From this table and a randomly chosen number between 0 and 1, one gets a random number M of equivalence classes. Thereafter, for each element in $[0..n]$ one picks up randomly uniformly and independently individuals in $[0..(M-1)]$. This method yields *class descriptions* (classes are a priori numbered from 0 to $M-1$ and the elements 0,..., $n-1$ are distributed in those classes), but one wants *restricted growth strings* as described in Section 3. So a function that transforms a class description into a restricted growth string was implemented.

Putting together those two algorithms, namely binary tree random generation and restricted growth string random generation, produces an algorithm for canonical expression random generation.

5 Selecting intuitionistic theorems

Once a canonical expression is randomly generated, one has to check whether it is an intuitionistic theorem, a classical theorem, or not a theorem of those sorts

The program selects two kinds of trivial intuitionistic expressions. At first glance this selection looks coarse, but from experience, the first one (*simple* theorems) collects a large majority of the expressions and the second selects (*Elim* theorems) most of the others, because it is associated with a trick which consists in cleaning expressions by removing recursively trivial subexpressions that are theorems. Indeed a "cleaned" sub-expression can become trivial and be removed in turn. This might allow cleaning an expression where a trivial premise appears, which might be removed in turn.

5.1 Simple intuitionistic theorems

Let us call *simple intuitionistic theorem* (see [7] Definition 1), a theorem, in which the goal is among the premises. In other words, this is a theorem of the form:

$$... \to \alpha_i \to ... \to \alpha_i$$

5.2 Elim intuitionistic theorems

Let us call *Elim intuitionistic theorem* (for $\to$-*eliminating* theorem), a theorem which is a direct application of the modus ponens aka $\to$-elimination. This is a theorem with goal α_i and two premises α_j and $\alpha_j \to \alpha_i$. Therefore it has the form:

$$... \to (\alpha_j \to \alpha_i) \to ... \to \alpha_j \to ... \to \alpha_i$$

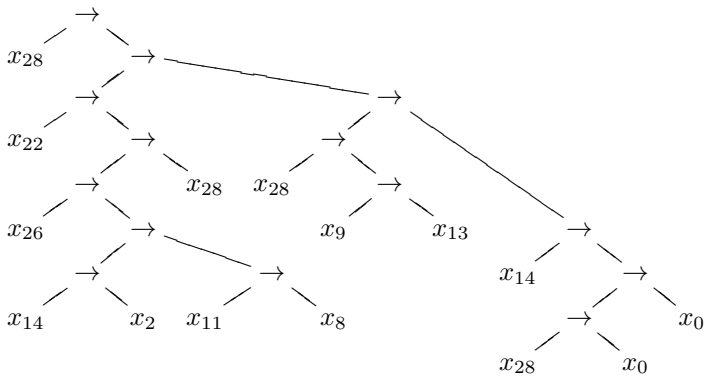
or

$$... \to \alpha_j \to ... \to (\alpha_j \to \alpha_i) \to ... \to \alpha_i$$

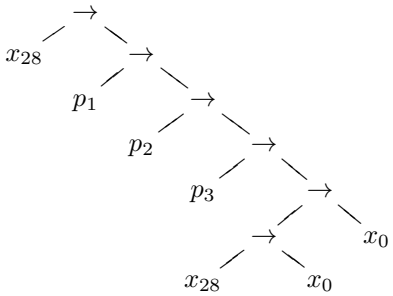
During the experiments I met, for instance, the term, which is not a canonical expression, but obtained by cleaning:

$$(x_{28} \to ((x_{22} \to ((x_{26} \to ((x_{14} \to x_2) \to (x_{11} \to x_8))) \to x_{28})) \to ((x_{28} \to (x_9 \to x_{13})) \to (x_{14} \to ((x_{28} \to x_{0)} \to x_0)))))$$

which can be drawn as the labelled binary tree:



which can be written



It is clearly an intuitionistic theorem and isElim checks it. The reader is invited to look at the Haskell implementation to see the two variants of isElim.

5.3 Easy intuitionistic theorems

Let us call *Easy intuitionistic theorems*, expressions that are simple or Elim.

5.4 Removing trivial premises

In intuitionistic logic if a premise is a theorem, it can be removed. Consider the predicate $\vdash p$ that says that p is a theorem. Clearly under the assumption $\vdash p$, the two statements $\vdash p \to q$ and $\vdash q$ are equivalent. Note that p is not necessarily the first premise of the implication. Hence if an expression becomes *easy after removing trivial premises*, it is an intuitionistic theorem.

In the process of "cleaning" expressions, expressions that are easy are removed inside-out. This way easy expressions that can be removed recursively are detected.

5.5 Silly intuitionistic theorems

A silly theorem is a theorem of the form ... $\to p \to$ ... $\to p$. Detecting such expressions has a cost, I decided to no detect silly intuitionistic theorem recursively, but only after easy subexpressions have been removed recursively.

5.6 Cheap intuitionistic theorems

Let us call *cheap intuitionistic theorems*, expressions that are silly or easy after removing (recursively) easy premises. Actually, my experiments lead naturally to the statement that classical theorems are 96% cheap intuitionistic (see Section 7).

6 Classical tautologies

The selection of classical tautologies is as usual, by valuations. Indeed if all the valuations of a given expression yield True this expression is a classical tautology. But this method is obviously intractable [3]. It should be applied only to expressions on which other more efficient methods do not work and with a limitation of the number of variables in expressions[1].

6.1 Trivial non classical propositions

Before applying valuations, some trivial non classical propositions must be eliminated. The function trivNonClass checks in quadratic time that a given expression is not a tautology. Only if an expression is not trivNonClass, boolean valuations check whether it is classical. For more efficiency, trivNonClass is applied on expressions in which trivial premises have been recursively removed, like for intuitionistic expressions.

An expression e is trivially non classical if it is of the form

$$... \to e_i \to ... \to x_0$$

where the premises e_i are of the form

$$\to ... \to x_i \qquad x_i \neq x_0$$

i.e., have a goal which is not x_0 or are of the form

$$... \to x_0 \to ... \to x_0$$

i.e., are simple with goal x_0. One sees easily that applying the valuation ρ such that $\rho(x_0) =$ False and $\rho(x_i) =$ True for $i \neq 0$ to $[\![e]\!]_\rho = False$. Therefore e is not classical.

In [7], Genitrini, Kozik and Zaionc consider only the first case, namely the case where the premises have a goal which is note x_0. They call such expressions, *simple non tautologies*. To avoid stacks of *non*, I suggest to call them *simple antilogies*.

6.2 Expressions with too many variables

Despite of trivNonClass check, there are still expressions intractable by the valuation method, because they have too many variables, i.e., they have an index which is too large. In the forthcoming experiment with expressions of size 100, an index is too large if it is larger than 31. Fortunately those expressions are rare and one may expect that there is a valuation that rejects them. The trick is to rename all the too large indices with the chosen bound. The valuations are checked on this renamed expression. If the renamed expression is not a tautology, then the given expression is not a tautology. In the experiment of Section 7 this trick works and eliminates expressions with too large indices which need not to be checked further.

[1] In my experience with canonical expressions of size 100 the variables should not exceed 30 which is rare enough to affect no classical theorem.

7 Results

7.1 Ratio cheap vs classical

I run my Haskell program on a sample of 20 000 randomly generated canonical expressions of size 100 and I found 759 classical tautologies, among which 733 were cheap expressions, hence guaranteed to be intuitionistic theorems. Therefore the ratio of cheap theorems over classical theorems is 96.6%. Are the 26 classical non cheap theorems still intuitionistic? The experience cannot tell. I presume that there are likely more than 733 intuitionistic theorems and therefore more than 96.6% of classical theorems that are intuitionistic.

7.2 Simple intuitionistic theorems vs non simple antilogies

In [7], Genitrini, Kozik and Zaionc take the ratio of the number of simple intuitionistic theorems over the number of non simple antilogies as the quantity that goes to 1 and bounds the ratio of the number of intuitionistic theorems over the number of classical theorems. Among 10 000 random canonical expressions of size 100, I found 238 simple intuitionistic theorems and 685 non simple antilogies, for a ratio closed to 36%, a ratio largely smaller than the above one.

7.3 Simple intuitionistic theorems

Besides, another number of interest is the ratio R_n of simple intuitionistic theorems over all canonical expressions of size n. In the next array, this is compared with the formula $\frac{log(n)}{n}$.

n	$\frac{log(n)}{n}$	R_n
25	$0,128755033$	0.2214
50	$0,07824046$	0.1248
100	$0,046051702$	0.0506
500	$0,012429216$	0.0119
1000	$0,006907755$	0.006

Genitrini, Kozik and Zaionc [7] gave $\frac{e\ log(n)}{n}$ in Lemma 2, for the same quantity, but after viewing my results the formula has been corrected [6]. Notice that this does not affect their other results.

Acknowledgements

I thank Valeria De Paiva for an interesting interaction and the incentive to address this problem, Jean-Luc Rémy for very old discussions on binary tree generation and Antoine Genitrini for recent discussions, on Zaionc paradox.

8 Conclusion

Algorithms for random generation presented in *The Art of Computer Programming* [8, 9] allow implementing Monte-Carlo methods that confirm experimentally Zaionc paradox and show that the convergence[2] of the set of intuitionistic theorems toward this of classical theorems is faster than expected from the asymptotic approximations proposed by the theory [7]. Indeed, whereas

[2] As the size of the expressions grows.

I compare the set of cheap intuitionistic theorems (Section 5.6) with this of classical theorems, Genitrini, Kozik and Zaionc compare the set of simple intuitionistic theorems (see Section 5.1) with the set of non simple antilogies (Section 6.1). This is a too rough approximation. This suggests to complete the analytic development to justify this faster convergence.

Notice that Tarau and de Paiva [14] looked at a phenomenon similar to Zaionc paradox for linear logic. Therefore, it should be interesting to extend my approach to this case. Likewise, it would be interesting to investigate experimentally other models of expressions, for both traditional logic and linear logic. Currently I am exploring expressions made of a binary operator, like $\wedge$, $\vee$ or $\rightarrow$, and a unary operator like $\neg$. This requires a random generator of Motzkin trees, aka unary-binary trees, which is something new.

It seems that this result on the distribution of propositions has to do with the amazing efficiency of SAT-solvers [2, 1]. The fact that most of the classical propositions can be solved as "cheap" intuitionistic propositions may explain why SAT-solvers are so efficient and the connection should be further investigated. Likely, the remaining true classical propositions contribute to the hardness of SAT for the worst case analysis.

References

[1] Armin Biere, Marijn J.H. Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS press, 2009.

[2] Curtis Bright, Jürgen Gerhard, Ilias S. Kotsireas, and Vijay Ganesh. Effective problem solving using SAT solvers. In Jürgen Gerhard and Ilias S. Kotsireas, editors, *Maple in Mathematics Education and Research - Third Maple Conference, MC 2019, Waterloo, Ontario, Canada, October 15-17, 2019, Proceedings*, volume 1125 of *Communications in Computer and Information Science*, pages 205–219. Springer, 2019. URL: https://doi.org/10.1007/978-3-030-41258-6_15, doi:10.1007/978-3-030-41258-6_15.

[3] Stephen A. Cook. The complexity of theorem-proving procedures. In Michael A. Harrison, Ranan B. Banerji, and Jeffrey D. Ullman, editors, *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*, pages 151–158. ACM, 1971. URL: https://doi.org/10.1145/800157.805047, doi:10.1145/800157.805047.

[4] Philippe Flajolet and Robert Sedgewick. *Analytic Combinatorics*. Cambridge University Press, 2008.

[5] Hervé Fournier, Danièle Gardy, Antoine Genitrini, and Marek Zaionc. Classical and intuitionistic logic are asymptotically identical. In Jacques Duparc and Thomas A. Henzinger, editors, *CSL*, volume 4646 of *Lecture Notes in Computer Science*, pages 177–193. Springer, 2007.

[6] Antoine Genitrini. Erratum for the paper *Intuitionistic vs Classical Tautologies, Quantitative Comparison*, 2021.

[7] Antoine Genitrini, Jakub Kozik, and Marek Zaionc. Intuitionistic vs. classical tautologies, quantitative comparison. In Marino Miculan, Ivan Scagnetto, and Furio Honsell, editors, *Types for Proofs and Programs, International Conference, TYPES 2007, Cividale del Friuli, Italy, May 2-5, 2007, Revised Selected Papers*, volume 4941 of *Lecture Notes in Computer Science*, pages 100–109. Springer, 2007. URL: https://doi.org/10.1007/978-3-540-68103-8_7, doi:10.1007/978-3-540-68103-8_7.

[8] Donald E. Knuth. *The Art of Computer Programming, Volume 4, Fascicle 3: Generating All Combinations and Partitions*. Addison-Wesley Publishing Company, 2005.

[9] Donald E. Knuth. *The Art of Computer Programming, Volume 4, Fascicle 4: Generating All Trees, History of Combinatorial Generation*. Addison-Wesley Publishing Company, 2006.

[10] Jean-Luc Rémy. Un procédé itératif de dénombrement d'arbres binaires et son application à leur génération aléatoire. *RAIRO Theor. Informatics Appl.*, 19(2):179–195, 1985. URL:

https://doi.org/10.1051/ita/1985190201791, doi:10.1051/ita/1985190201791.

[11] N. J. A. Sloane. The on-line encyclopedia of integer sequences. Published electronically at https://oeis.org/, 2021.

[12] A. J. Stam. Generation of a random partition of a finite set by an urn model. *J. Comb. Theory, Ser. A*, 35(2):231–240, 1983. URL: https://doi.org/10.1016/0097-3165(83)90009-2, doi:10.1016/0097-3165(83)90009-2.

[13] Paul Tarau. A hiking trip through the orders of magnitude: Deriving efficient generators for closed simply-typed lambda terms and normal forms. In Manuel V. Hermenegildo and Pedro López-García, editors, *Logic-Based Program Synthesis and Transformation - 26th International Symposium, LOPSTR 2016, Edinburgh, UK, September 6-8, 2016, Revised Selected Papers*, volume 10184 of *Lecture Notes in Computer Science*, pages 240–255. Springer, 2016. URL: https://doi.org/10.1007/978-3-319-63139-4_14, doi:10.1007/978-3-319-63139-4_14.

[14] Paul Tarau and Valeria de Paiva. Deriving theorems in implicational linear logic, declaratively. In Francesco Ricca, Alessandra Russo, Sergio Greco, Nicola Leone, Alexander Artikis, Gerhard Friedrich, Paul Fodor, Angelika Kimmig, Francesca A. Lisi, Marco Maratea, Alessandra Mileo, and Fabrizio Riguzzi, editors, *Proceedings 36th International Conference on Logic Programming (Technical Communications), ICLP Technical Communications 2020, (Technical Communications) UNICAL, Rende (CS), Italy, 18-24th September 2020*, volume 325 of *EPTCS*, pages 110–123, 2020. URL: https://doi.org/10.4204/EPTCS.325.18, doi:10.4204/EPTCS.325.18.