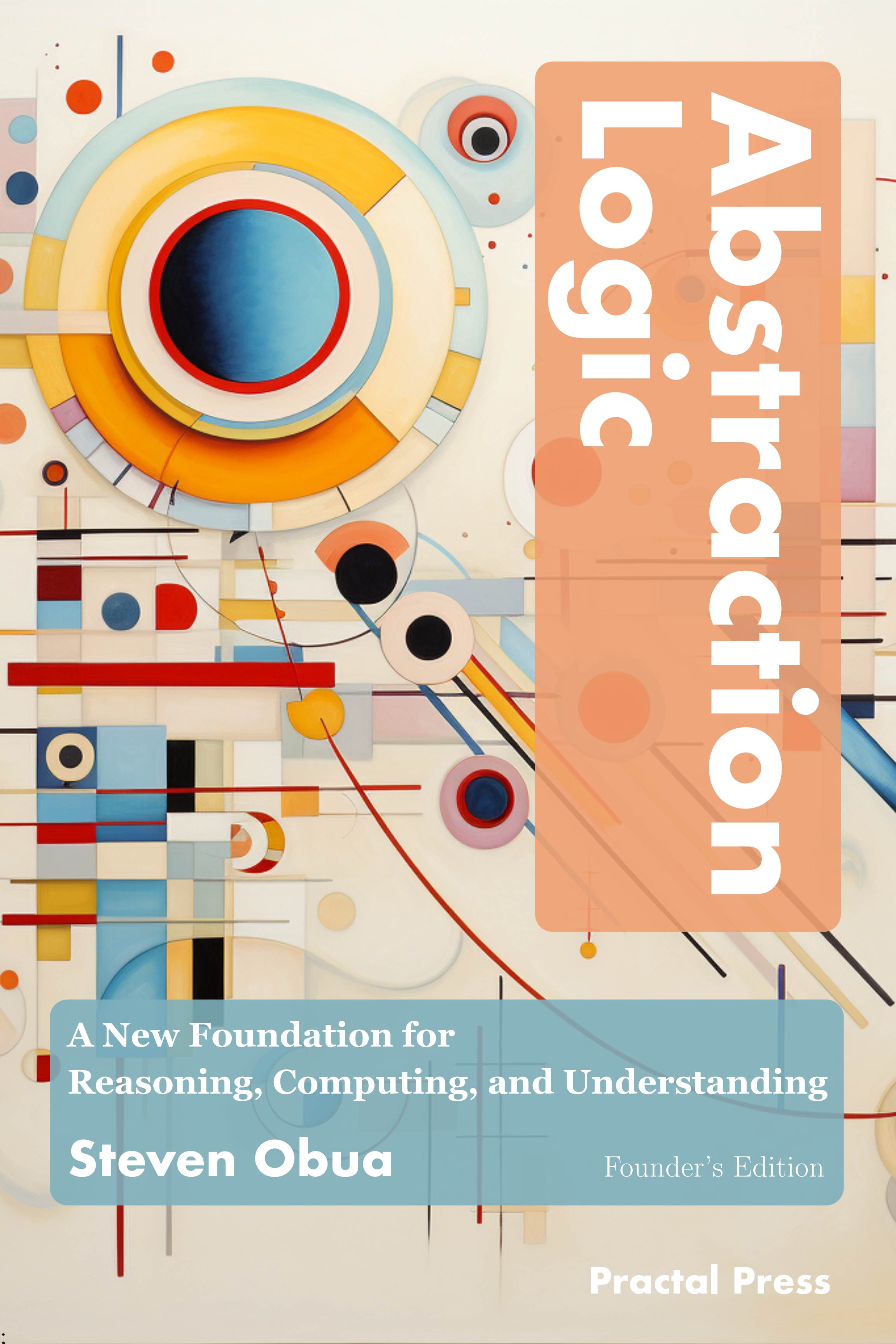
Abstraction Logic

A New Foundation for Reasoning, Computing, and Understanding

Steven Obua

Founder's Edition

Practal Press



Abstraction logic is a new logic combining exceptional simplicity with astonishing generality. It combines the *best features of first-order logic and higher-order logic*, while avoiding their respective drawbacks. It manages to do so because it is based on a simple understanding of the *mathematical universe*, its *operations* and, in particular, its *operators*. *Abstraction algebra* encodes this understanding as a formal language, generalising abstract algebra. It is the right setting for the treatment of *alpha equivalence*. Abstraction logic then turns abstraction algebra into a logic by considering *truth values as a partially ordered substructure* of the mathematical universe. A key property of this logic is that *formulas are merely terms*. Among the presented proof systems are *natural deduction*, which is sound if truth values form a *complete lattice*, and *sequent calculus*, which is sound if truth values form a *complete bi-Heyting algebra*. By constructing the *Rasiowa model*, we prove that *natural deduction is a complete proof system* for abstraction logic.

About the Author

Steven Obua holds a Diplom in mathematics with a focus on computer science from the Technische Universität München (TUM), where he also spent time studying at the University of San Francisco. He earned his PhD in interactive theorem proving from TUM, and as a member of the Isabelle development team during his doctoral studies, he made significant contributions to the Flyspeck project, which successfully verified the proof of the Kepler conjecture.

His postdoctoral research included work on the verification of Microsoft's hypervisor and a brief but influential period at École Polytechnique Fédérale de Lausanne exploring the design of a novel theorem proving system. At the University of Edinburgh, he was the researcher co-investigator for ProofPeer, an EPSRC-funded project pioneering collaborative theorem proving.

Dr. Obua's career spans both academia and industry, where he has worked in content management, insurance, mobile app development, hardware and software formal verification, gaming and computer-aided design. He is currently developing Practal, a new system that aims to empower vision through precision, based on abstraction logic.

For Anita (mum) and Wolfgang Rosenberg

Abstraction Logic

A New Foundation for Reasoning, Computing, and Understanding

Founder's Edition

http://abstractionlogic.com

Steven Obua

https://doi.org/10.5281/zenodo.14790824

Key words and phrases. Logic, Algebra, Computer Algebra, Mathematical Foundations, Mathematical Logic, Axiomatic Reasoning, Natural Deduction, Sequent Calculus, Bi-Heyting Algebras, De Bruijn Indices, Unification, Pattern Matching, Lambda Calculus, Simply-Typed Lambda Calculus, Higher-Order Logic, Predicate Logic, First-Order Logic, Second-Order Logic, Algebraic Specification, Algebraic Logic, Universal Algebra, Set Theory, Type Theory, Logical Frameworks

Abstract. *Abstraction logic* is a new logic combining exceptional simplicity with astonishing generality. It combines the *best features of first-order logic and higher-order logic*, while avoiding their respective drawbacks. It manages to do so because it is based on a simple understanding of the *mathematical universe*, its *operations* and, in particular, its *operators*. *Abstraction algebra* encodes this understanding as a formal language, generalising abstract algebra. It is the right setting for the treatment of *alpha equivalence*. Abstraction logic then turns abstraction algebra into a logic by considering *truth values as a partially ordered substructure* of the mathematical universe. A key property of this logic is that *formulas are merely terms*. Among the presented proof systems are *natural deduction*, which is sound if truth values form a *complete lattice*, and *sequent calculus*, which is sound if truth values form a *complete bi-Heyting algebra*. By constructing the *Rasiowa model*, we prove that *natural deduction is a complete proof system* for abstraction logic.

This is the first book on abstraction logic. It presents abstraction logic in its most recent and comprehensive form, and supersedes all previous publications on abstraction logic.

Contents

Preface

This book explores the topic of *abstraction logic*, a new logic I discovered in the second half of 2021 [**1, 2, 3**]. Since then, I have continued to develop the logic [**4, 5, 6**]. The material in this book supersedes any of my previous publications on abstraction logic, improving on them both in presentation and content.

This Founder's Edition consists of three complete chapters, "The Mathematical Universe", "Abstraction Algebra", and "Abstraction Logic", and represents a milestone in the development of abstraction logic.

Abstraction logic forms the theoretical foundation for the Practal system[1]. At its core, Practal is an interactive theorem proving system. Its ambition extends beyond that; it aims to be a general tool for creation and analysis, empowering your vision by precision.

Should you have any comments, questions, or corrections, please do not hesitate to contact me at obua@practal.com.

Acknowledgements. Many thanks to Norbert Schirmer for his comments, questions, and suggestions. I also want to express my gratitude to Zarathustra Elessar Brady for pointing out that without axiom $\textsc{Extensionality}_\forall$ (see Figure 3.10), the completeness of *Deduction Logic with Equality* seems in danger. It is not, but only because the meaning of abstractions is more malleable than even I thought at the time. Amine Chaieb read an early paper on abstraction logic despite going through COVID-19, and provided insightful comments on hard truth. They lingered in the back of my mind and certainly eventually contributed to the concept of logical order. I would like to thank Mario Carneiro and the other participants of a Lean Zulip chat[2] about Practical Types, which motivated me to search for abstraction logic.

[1]`https://practal.com`, in development

[2]`https://leanprover.zulipchat.com/#narrow/stream/236446-Type-theory/topic/Practical.20Types`

Many thanks to Anna Maginis, whose keen eye for design strongly influenced this work. Without her loving support, this book would not exist. I also want to mention Dimitris Metaxas, her brother. His untimely death spurred the development of this book, which at that time consisted of only the first chapter. Time goes by in a hurry, and we have only limited time to do the things we want to do.

January 1st, 2025

CHAPTER 1

The Mathematical Universe

ABSTRACT. *The* mathematical universe *is introduced, together with* operations *to combine the* mathematical objects *of the mathematical universe. We argue* against *trying to turn operations into mathematical objects, as this is impossible to do for* all *operations due to* Cantor's theorem. *Instead, we propose* operators *to manage operations.*

The **mathematical universe**, usually denoted as $\mathcal{U}$, consists of the **mathematical objects** under consideration. These include numbers, functions, sets, booleans, computer programs, vector spaces, algebras, categories, and so forth. The phrase 'under consideration' suggests that the contents of the mathematical universe may be context dependent. For example, at one point, you might be interested solely in natural numbers, while at another, your focus might be limited to the boolean values *true* and *false*. At yet another time, your attention may be on all subsets of $\mathbb{N}$, the set of all natural numbers. That is why the mathematical universe is also called the *universe of discourse*.

While we certainly allow different versions of mathematical universes, as small or as large as you prefer and can justify, in practice

arity n	format	n-ary operations
0	$\begin{pmatrix} f \end{pmatrix}$	$\begin{pmatrix} 0 \end{pmatrix}$ $\begin{pmatrix} 1 \end{pmatrix}$
1	$\begin{pmatrix} f(0) \\ f(1) \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \end{pmatrix}$ $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$ $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ $\begin{pmatrix} 1 \\ 1 \end{pmatrix}$
2	$\begin{pmatrix} f(0,0) & f(0,1) \\ f(1,0) & f(1,1) \end{pmatrix}$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$ $\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$ $\begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$ $\begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}$ $\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$ $\begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}$ $\begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix}$ $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ $\begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix}$ $\begin{pmatrix} 0 & 0 \\ 1 & 1 \end{pmatrix}$ $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$ $\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$ $\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$ $\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$ $\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$

FIGURE 1.1. All nullary, unary, and binary operations on $\mathbb{2}$

$\mathcal{U}$ will often encompass all mathematical objects you could possibly conceive of, simultaneously. For example, you are generally not solely interested in studying just the natural numbers. Instead, you are interested in the natural numbers $\mathbb{N}$, the real numbers $\mathbb{R}$, functions from $\mathbb{N}$ to $\mathbb{R}$, and many more mathematical objects existing simultaneously and possibly interacting with each other. One large mathematical universe can accommodate all of these at the same time.

The mathematical universe is not allowed to be empty. If it comprises exactly one mathematical object, then $\mathcal{U}$ is called **degenerate**. Therefore, every nondegenerate mathematical universe contains at least two distinct mathematical objects.

We also refer to mathematical objects simply as **values**, and often abbreviate 'mathematical universe' to **universe**.

Operations

Values on their own are not particularly interesting. The magic starts when values are combined, resulting in other values.

Operations provide the means to do so.

DEFINITION 1.1 (Operation). An **n-ary operation on $\mathcal{U}$** is a function that accepts n values from $\mathcal{U}$ as inputs and yields a single value from $\mathcal{U}$ as its output. A nullary operation is simply a value.

EXAMPLE 1.1 (Finite Universes). Let $\mathbb{2}$ be the universe consisting of 0 and 1 only. For each fixed n, there are only a finite number of n-ary operations on $\mathbb{2}$, all of which can be listed. There are two nullary operations, four unary operations, and sixteen binary operations, each depicted in Figure 1.1.

Generally, assume the universe $\mathcal{U}$ is finite and consists of s different values. Each n-ary operation can then be represented as an n-dimensional array, where each dimension is of size s, and each array entry contains the result of applying the operation to a specific combination of n values. Each such array has s^n entries, and the content of each entry can be chosen independently from s values to produce a unique operation. This results in $s^{(s^n)}$ different n-ary operations on $\mathcal{U}$.

Specifically, there are $2^{(2^n)}$ n-ary operations on $\mathbb{2}$. It would be impractical to extend Figure 1.1 much further. For $n = 3$ we already need to list 256 operations. For $n = 4$, this increases to 65,536 operations. For $n = 5$, the number of operations grows to 4,294,967,296.

Example 1.2 (Algebra). A universe $\mathcal{U}$ together with a finite number of operations $o_1, \ldots, o_k$ on $\mathcal{U}$ is also called an **abstract algebra** [**7**, p. 287], a **universal algebra** [**8**, p. 8], or simply just an **algebra**. Such an algebra is denoted as $(\mathcal{U}, o_1, \ldots, o_k)$. The following are some examples of algebras.

(1) Natural numbers $\mathbb{N}$, integers $\mathbb{Z}$, and real numbers $\mathbb{R}$ form, together with the binary operations addition $(+)$ and multiplication $(\cdot)$, the algebras $(\mathbb{N}, +, \cdot)$, $(\mathbb{Z}, +, \cdot)$, and $(\mathbb{R}, +, \cdot)$, respectively.

(2) The universe $\mathbb{2}$ forms together with disjunction $(\vee)$, conjunction $(\wedge)$ and negation $(\neg)$ the algebra $(\mathbb{2}, 0, 1, \vee, \wedge, \neg)$. Interpreting 0 as *false*, and 1 as *true*, disjunction is defined as $\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$, conjunction as $\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$, and negation as $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ (see Figure 1.1). This algebra is called the **two-element Boolean algebra**.

(3) Let X be a non-empty set, and $\mathcal{P}(X)$ the set of all subsets of X. Together with the empty set $(\emptyset)$, union $(\cup)$, intersection $(\cap)$, and complement with respect to X $(-)$, this produces another **Boolean algebra** $(\mathcal{P}(X), \emptyset, X, \cup, \cap, -)$.

Example 1.3 (Calculator). Let's design the universe and operations for a simple calculator. It is so simple it can only handle integers, so initially, our universe is $\mathbb{Z}$. As operations, we want addition ($+$), subtraction ($-$), multiplication ($\times$) and division ($\div$). Defining these is straightforward for addition, subtraction, and multiplication. The challenge, naturally, is division. What should the result of $7 \div 3$ be? Or $1 \div 0$? What about $0 \div 0$? There are numerous ways to address this issue. We choose a particularly simple one. Whenever there is a unique integer z such that $x = z \times y$, the result of $x \div y$ is z. Otherwise, the result of $x \div y$ is *Error*. For this to work, we need to expand our universe to include not only all integers, but also the value *Error*. This presents another problem. Now we also need to make sense of $3 + \mathit{Error}$, $\mathit{Error} \div \mathit{Error}$, and so forth. But this is also simple to resolve. Whenever at least one of the input values is *Error*, then the result is also *Error*. In conclusion, this means that $7 \div 3$, $1 \div 0$, $0 \div 0$, $3 + \mathit{Error}$ and $\mathit{Error} \div \mathit{Error}$ all yield the same result: *Error*.

Cantor's Theorem

Example 1.1 shows that for any nondegenerate finite mathematical universe, there are more operations than mathematical objects, even when considering only unary operations. If the universe consists of s values, then there are s^s unary operations, and $s^s > s$ for $s \geq 2$.

The following variant of **Cantor's theorem** [**9**] asserts that there are strictly fewer mathematical objects than unary operations for *any* nondegenerate universe $\mathcal{U}$, even when $\mathcal{U}$ is infinite.

THEOREM 1.1 (Cantor's Theorem for Unary Operations). *Let $\mathcal{U}$ be a nondegenerate mathematical universe. There is an injective function which maps $\mathcal{U}$ to the space of all unary operations on $\mathcal{U}$, but there is no such surjective function.*

PROOF. Define a function I from $\mathcal{U}$ to the space of all unary operations on $\mathcal{U}$ by assigning to each value u the unary operation $I(u) = f_u$, where $f_u(v) = u$ for all values v. Then $u \neq u'$ implies $f_u \neq f_{u'}$, which means that I is injective.

Assume, on the other hand, that there was a surjective function S from $\mathcal{U}$ to the space of all unary operations on $\mathcal{U}$. Define a new unary operation f such that $f(u)$ yields a value v such that $v \neq S(u)(u)$. This is possible because $\mathcal{U}$ is nondegenerate and therefore contains at least two values. For any value u, $f(u) \neq S(u)(u)$, which implies $f \neq S(u)$. Therefore, f is not in the image of S, which contradicts the assumption that S is surjective. $\square$

Are Operations Mathematical Objects?

Cantor's theorem establishes that there are more operations than mathematical objects. This might seem paradoxical: are operations not *also* considered mathematical objects?

Indeed, it is tempting to try to incorporate all operations into the mathematical universe. However, Cantor's theorem informs us that this is impossible. If we proceeded regardless, paradoxes would emerge.

One potential solution to this dilemma could be as follows. We begin with a mathematical universe, $\mathcal{U}_0$. However, we don't stop there. We generate a new mathematical universe, $\mathcal{U}_1$, by incorporating all operations on $\mathcal{U}_0$ into it, possibly augmented by additional mathematical objects such as tuples and lists of elements of $\mathcal{U}_1$. This process is repeated, forming new universes $\mathcal{U}_{i+1}$ on top of universes $\mathcal{U}_i$. We have created a **tower of mathematical universes** (Figure 1.2).

Did we solve our dilemma? What are our mathematical objects now? A sensible definition is that something is a mathematical object if there is a universe $\mathcal{U}_i$ which contains it. Let us call this ultimate collection of mathematical objects $\mathcal{U}_\infty$.

Of course, we didn't really solve our dilemma: $\mathcal{U}_\infty$ does not contain all operations on $\mathcal{U}_\infty$! There are still many entities left we would like to treat as mathematical objects, but cannot. We are exactly where we started, albeit at a higher vantage point.

Example 1.4 (Simple Operation on $\mathcal{U}_\infty$). Let us assume that each $\mathcal{U}_i$, for $i \in \mathbb{N}$, contains all natural numbers and all finite lists of elements of $\mathcal{U}_i$. Then $\mathcal{U}_\infty$ also contains all natural numbers and all finite lists of elements of $\mathcal{U}_\infty$. But the operation *length* which assigns to each such list its length is not an element of $\mathcal{U}_i$ for any $i \in \mathbb{N}$, and is consequently not an element of $\mathcal{U}_\infty$. $\square$

We must accept that operations on $\mathcal{U}$ are, generally, not mathematical objects residing within $\mathcal{U}$. Once this is understood, it becomes clear that we have not lost anything. We do not need to build a tower into our foundations. Any concrete tower $(\mathcal{U}_i)_{i\in\mathbb{N}}$ of interest can be modelled explicitly as a mathematical object within our mathematical universe $\mathcal{U}$, including $\mathcal{U}_\infty$ and all operations on $\mathcal{U}_\infty$. Usually, we do not need a tower at all.

However, to realise this, we need a substitute for the tower's defining feature. By considering the operations on $\mathcal{U}_i$ as mathematical objects of $\mathcal{U}_{i+1}$, the tower makes it possible to discuss operations at each level of the tower. Without the tower, we need an alternative method to talk about operations. This is the purpose of *operators*.

Operators

Just as operations take values as arguments, *operators* take operations as their arguments.

Definition 1.2 (Operator). An **n-ary operator on $\mathcal{U}$** is a function that accepts n operations on $\mathcal{U}$ as inputs and yields a single value from $\mathcal{U}$ as its output. Each input position i of the operator accepts only operations of a fixed arity m_i. The list $[m_1, \ldots, m_n]$ is referred to as the **shape** of the operator. □

Since a value is also a nullary operation, an n-ary operation is also an n-ary operator, the shape of which is $\underbrace{[0, \ldots, 0]}_{n \text{ times}}$. In particular, a value is an operator of shape $[\,]$. Thus, every mathematical object is also an operation, and every operation is an operator (Figure 1.3). Nevertheless, generally, operations and operators are not mathematical objects.

Example 1.5 (Calculator, continued). Continuing Example 1.3, let us introduce operators $\sum$ and $\prod$. Both have shape $[0, 0, 1]$, which means they take values l and u as their first two input arguments, and a unary operation f as their third (and last) input argument. We define them via

$$\begin{aligned} \textstyle\sum(l, u, f) &= f(l) + f(l+1) + \cdots + f(u-1) + f(u), \\ \textstyle\prod(l, u, f) &= f(l) \times f(l+1) \times \cdots \times f(u-1) \times f(u). \end{aligned}$$

A few special cases are understood. For $l > u$ we set $\sum(l, u, f) = 0$ and $\prod(l, u, f) = 1$. As before, $\sum(\mathit{Error}, u, f) = \sum(l, \mathit{Error}, f) = \prod(\mathit{Error}, u, f) = \prod(l, \mathit{Error}, f) = \mathit{Error}$.

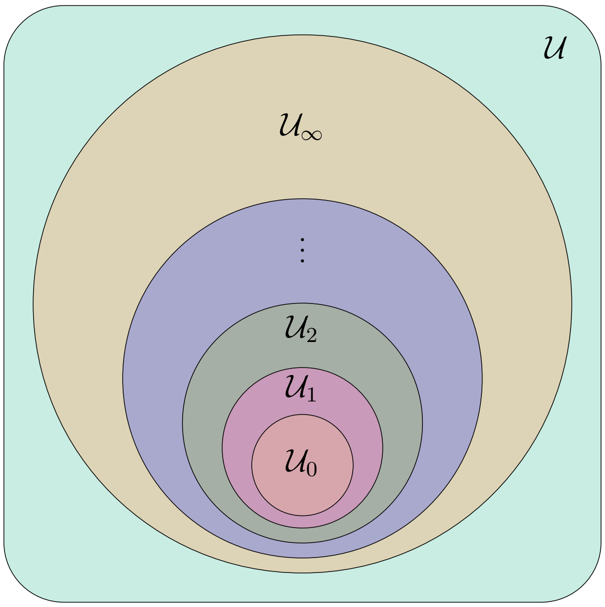


FIGURE 1.2. A tower of universes $\mathcal{U}_i$, forming a universe $\mathcal{U}_\infty$. $\mathcal{U}_\infty$ is still insufficient and does not give us all of $\mathcal{U}$

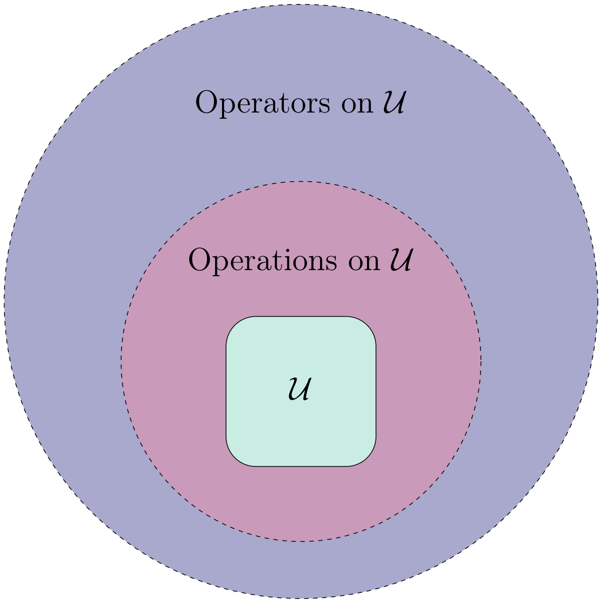


FIGURE 1.3. The mathematical universe $\mathcal{U}$, together with its operations and operators

Example 1.6. Generalising $\sum$ and $\prod$, we define an operator *iter* of shape $[0, 0, 0, 2, 1]$ by

$$\begin{aligned} \mathit{iter}(l, u, x, g, f) &= g(\mathit{iter}(l, u-1, x, g, f), f(u)) \quad \text{for } l \leq u, \\ \mathit{iter}(l, u, x, g, f) &= x \quad \text{for } l > u, \\ \mathit{iter}(l, u, x, g, f) &= \mathit{Error} \quad \text{for } l = \mathit{Error} \text{ or } u = \mathit{Error}. \end{aligned}$$

Then $\sum(l, u, f) = \mathit{iter}(l, u, 0, +, f)$ and $\prod(l, u, f) = \mathit{iter}(l, u, 1, \times, f)$.
□

At first, it might appear overly restrictive that an operator can return only a single value. There are two reasons why this is not the case. First, much like *currying* [**10**, p. 86], we can see that operators can, in a sense[1], return operations and even other operators. For instance, consider the operator *repeat* of shape $[0, 1, 0]$ defined by

$$\mathit{repeat}(n, f, x) = \underbrace{f(f(\ldots f}_{n \text{ times}}(x)\ldots)).$$

This operator could also be interpreted as a function mapping n to an operator of shape $[1, 0]$, as in mapping 2 to the operator *twice*.

Second, if we wish to return more complicated things than a single value, such as multiple values, we can simply extend our mathematical universe to include them. These things could be tuples, sets, or even functions.

We will find that the trinity of *values*, *operations* and *operators* is sufficient to express all of mathematics in a theoretically satisfying way, and perhaps more importantly, in a practically satisfying way as well.

[1]But operators cannot *actually* be partially applied.

CHAPTER 2

Abstraction Algebra

ABSTRACT. *We introduce a simple and elegant formal language for describing and working with mathematical objects, operations, and operators, called abstraction algebra. After presenting its syntax and semantics, we discuss alpha equivalence, a notion that captures the idea that two expressions of the language mean the same under any circumstances. Using de Bruijn indices, we prove that alpha equivalence has an alternative characterisation which is purely syntactic and easily computable. An essential tool used in the proof is substitution, which we also base on de Bruijn indices. The composition of substitutions is studied, as well as the relationship between substitution and evaluation.*

In Example 1.2, we have already mentioned that a universe $\mathcal{U}$ together with a (finite) collection of operations on $\mathcal{U}$ is called an **abstract algebra** [**7**, p. 287]. Given that in our setting we are considering not only values and operations, but also operators, it is natural to introduce the following generalisation of abstract algebras.

Proof. Corollary 2.2 already proves one direction. For the other direction, assume that S and T are alpha equivalent. Then they have the same arity n, and for all $\mathfrak{S}$-algebras $\mathcal{A}$, and all valuations ν into $\mathcal{A}$, $[\![S]\!]_\nu = [\![T]\!]_\nu$. In particular, this is true for the de Bruijn abstraction algebra $\mathcal{A}_{\mathrm{dB}}$ and the canonical valuation κ. Therefore, $[\![S]\!]_\kappa = [\![T]\!]_\kappa$, and

$$\begin{aligned}
& \alpha(S) \\
\text{(Lemma 2.16, Theorem 2.1)} \quad & = \alpha(S)[\uparrow 0, \ldots, \uparrow(n-1)] \\
\text{(Lemma 2.33, Theorem 2.1)} \quad & = [\![S]\!]_\kappa(\uparrow 0, \ldots, \uparrow(n-1)) \\
& = [\![T]\!]_\kappa(\uparrow 0, \ldots, \uparrow(n-1)) \\
\text{(Lemma 2.33, Theorem 2.1)} \quad & = \alpha(T)[\uparrow 0, \ldots, \uparrow(n-1)] \\
\text{(Lemma 2.16, Theorem 2.1)} \quad & = \alpha(T).
\end{aligned}$$

□

Concluding Remarks

This chapter has introduced abstraction algebra both as a formal language and as the name of a certain kind of mathematical structure which generalizes abstract algebras to also include operators instead of just operations.

Hopefully, the first two sections on the syntax and semantics of abstraction algebra have convinced you of its simplicity and elegance. Despite this simplicity, abstraction algebra is a powerful tool. As a consequence, the other sections discussing properties of abstraction algebra, instead of just using it, may appear less simple. This is especially the case if you are unfamiliar with the semantics of programming languages. For this is what abstraction algebra can also be used as: *a programming language.*

It shares this property with the lambda calculus. Issues arising in abstraction algebra have been identified long before in the lambda calculus, in particular with respect to how to perform substitution while avoiding the capture of free variables. To deal with these issues in his AUTOMATH system, de Bruijn introduced de Bruijn indices (which he called 'nameless dummies') over fifty years ago [**11**]. Unlike the lambda calculus, though, which needs to be augmented with types for this purpose (as de Bruijn also did for AUTOMATH), abstraction algebra can be turned into a logic directly, as we will see in the next chapter.

Our representation of terms via de Bruijn indices is similar to that found in implementations of typed higher-order logic, commonly referred to as *locally nameless* [**12**]. A difference is our treatment of dangling indices. While in the locally nameless approach, terms should have no dangling indices and be *locally closed*, we embrace the use of dangling indices, which occur organically in the alpha conversion of templates. For us, de Bruijn indices are not merely a representational issue but are of interest in their own right to settle theoretical issues. In particular, abstraction algebra is the correct setting to treat alpha equivalence.

Abstraction algebra generalizes abstract algebra and at the same time incorporates higher-order features. What is of particular interest is how this combination affects the treatment of standard topics in *term rewriting* [**13**], such as *pattern matching* and *unification*. This is an exciting area for future research.

CHAPTER 3

Abstraction Logic

Abstract. Abstraction logic *is introduced, building on our development of abstraction algebra. Its key property is that* formulas are merely terms*, because* truth values *form a partially ordered substructure within the mathematical universe. If truth values constitute a* complete lattice*, then* natural deduction *is a* sound proof system *for abstraction logic. Furthermore, if they form a* bi-Heyting algebra*,* sequent calculus *is also shown to be sound. By constructing the* Rasiowa model*, we prove that natural deduction is a* complete *proof system for abstraction logic.*

According to Tao [**14**, p. 1], the material implication 'A implies B' can be thought of as 'B is at least as true as A'. This is a very general view of implication, and as it turns out, perfectly suited to turn abstraction algebra into logic, *abstraction logic*.

Truth Values and Logical Order

For the sentence 'B is at least as true as A' to make sense, A and B must be things which can be compared. Let us call these things

$$\frac{\mathsf{or}_n(x_1, \ldots, x_n)}{x_1 \ldots x_n} \qquad \frac{x_i}{\mathsf{or}_n(x_1, \ldots, x_n)}$$

(OrE$_n$) For all $n \geq 0$ (OrI$_n$) For all $n \geq 1$ and $1 \leq i \leq n$

$$\frac{\mathsf{ex}^n(x_1 \ldots x_n.\, A[x_1, \ldots, x_n])}{x_1 \ldots x_n.\; A[x_1, \ldots, x_n]} \qquad \frac{A[x_1, \ldots, x_n]}{\mathsf{ex}^n(x_1 \ldots x_n.\, A[x_1, \ldots, x_n])}$$

(ExE$_n$) For all $n \geq 1$ (ExI$_n$) For all $n \geq 1$

Figure 3.14. Axioms Included in $\mathcal{L}^+$

any theorem that is a sequent into a theorem that is a rule, and back. For example,

$$\frac{S_1 \;\ldots\; S_n}{(x\,y\,z.\,a) \quad (y.\,b)} \quad \leftrightsquigarrow \quad \frac{S_1 \;\ldots\; S_n}{\mathsf{or}_2(\mathsf{ex}^3(x\,y\,z.\,a), \mathsf{ex}^1(y.\,b))}.$$

So for any additional axioms it is unnecessary to state them as general sequents, as they can just be stated as rules and converted to sequents afterwards.

By developing this idea further, a completeness theorem for sequent calculus might be within reach, as a consequence of the completeness of natural deduction.

Concluding Remarks

The insight that sequents, and thus templates, should be at the heart of abstraction logic was a late one. Originally, the focus of abstraction logic was on terms only, and required built-in abstractions for implication and universal quantification to do any logic. But then, both theory and practical considerations suggested that axioms should not just be mere terms, but inference rules. Expanding this to general sequents was obvious and straightforward. Yet, it is not clear if there is any practical advantage in considering sequent calculus instead of just using natural deduction, although the symmetry of sequent calculus is compelling. The deduction theorem holds for both sequent calculus and natural deduction, but completeness has been proven so far only for natural deduction. The deduction theorem does not seem to hold for those proof systems in between natural deduction and sequent calculus, such as sequence deduction. This might be taken as a hint that they are not important.

This is only the beginning of our journey into abstraction logic. The goal has always been to create a new foundation for reasoning, computing and understanding — one that aligns better with the way mathematics is practised than current standards such as first-order logic or type theory. So far things are looking good, but there is much work ahead, and much to explore. Let us conclude this chapter by briefly discussing a few important points.

Mathematical Freedom. Abstraction logic does not limit your mathematical freedom in any way. You can discuss anything under the sun — or rather, anything possibly living in a mathematical universe. Your mathematical objects can mix and mingle as much as they like, but you can also introduce disciplined and conservative ways of constructing new mathematical objects. To manage your mathematical objects, you can introduce any abstraction you like. For example, it is easy to deal with **undefinedness**: mathematical objects such as *Error* can be introduced, different from *false* but logically equivalent. No static type system needs to be pacified for this to work; the mathematical expression of situations in which no error can occur remains unaffected and uncluttered.

Both Logic and Logical Framework. This is because abstraction logic serves as both a logic and a **logical framework**. The logic has a clear and simple semantics. It is at the same time minimal yet so powerful that we can prove a completeness theorem for natural deduction without assuming any concrete abstractions. This might be particularly interesting for fields such as algebraic specification, which have invented notions such as *institutions* to cope with the sheer variety of different useful logics that have resisted a unifying foundation so far. It is possible that abstraction logic is that foundation that has been evasive for so long, making notions such as ‘logical framework’ obsolete. However, it is important to acknowledge that abstraction logic is the spiritual child of one such framework: **Isabelle/Isar** [**26, 27, 28**]. Isabelle is also both a logic and a framework, and therefore there are many parallels between Isabelle’s logic *Pure* [**29**], which is intuitionistic higher-order logic, and abstraction logic. The difference is that Pure is rooted in type theory and thus proof theory, offering only complicated semantic accounts [**30**] of its logic, while the meaning of abstraction logic is clear and simple.

Hilbert-Style Natural Deduction. Abstraction logic unifies not only different kinds of logic but also different *approaches* to logic. It is a Hilbert-style system based on a few fixed inference rules and an arbitrary collection of axioms. But these axioms transcend the usual notion of an axiom, and are in fact also rules or even sequents. This means that natural deduction and also sequent calculus are integral to abstraction logic, unifying aspects of both model-theoretic and proof-theoretic approaches to logic.

Paraconsistency. *Paraconsistent* logics seem at first to be at odds with abstraction logic, as there is only one value *true* representing truth in abstraction logic, the top element of the logical order. There are paraconsistent logics such as Graham Priest's **Logic of Paradox** [**25**] which have multiple logical values designated as being true. Amine Chaieb pointed out early that this does not have to be a problem, as there can be a *hard truth* co-existing with other *soft truth values*. And indeed, this can be modelled in abstraction logic by using a logical order consisting of the ordinary two-valued Boolean algebra $\{true, false\}$, a universe $\{true, both, false\}$, and a logical mapping that assigns *true* to both *true* and *both*.

Philosophical Considerations. The example of paraconsistency shows that supposedly high-brow aspects of logic turn out to be quite simple in abstraction logic. This is not a coincidence. Abstraction logic is based on a simple conceptualisation of the mathematical universe, its operations and operators. It is hard to imagine a simpler foundation that still respects Cantor's theorem. But even to describe this simple foundation, we need to presuppose things like *collections*, to describe the mathematical universe, and *functions*, to describe operations and operators. Philosophically, our approach is Platonic. We can talk about basic concepts such as collections and functions without defining them because they are part of a mathematical reality we just tap into, rather than invent. While we reject paraconsistency and intuitionism from a philosophical point of view, they are perfectly valid and useful mathematical concepts that can be explained naturally within the framework of abstraction logic.

Logic	**One Mathematical Universe**	**General Binders**
First-Order	Yes	No
Second-Order	Yes	No
Higher-Order	No	Yes
Abstraction	Yes	Yes

FIGURE 3.15. Classifying Logics Along Two Dimensions

Standard and Non-Standard. Another example of how abstraction logic crystallises difficult concepts is that of standard versus non-standard models. According to Hintikka [**31**], the distinction between standard and non-standard models was first explicitly formulated by Henkin in 1950, in the context of higher-order logic, and represented a watershed in the foundations of mathematics. In abstraction logic, valuation spaces are a simple concept that elegantly captures the essence of this distinction. They make it possible for natural deduction to be complete yet capable of expressing rules such as natural induction succinctly:

$$\frac{A[0] \quad x.\, A[x] \Rightarrow A[x+1]}{A[x]}.$$

In standard models, the variable A varies over all possible unary operations on the mathematical universe, while in non-standard models which unary operations are covered depends on the valuation space of the model.

First-Order and Higher-Order. Logics are usually classified according to their order, such as first-order, second-order or higher-order. First-order logic is the common standard, while second-order and especially higher-order logic have gained popularity due to their practical expressiveness [**32**] and adjacency to computer languages. First-order logic makes it possible to reason about the entire mathematical universe uniformly, and ZFC set theory takes advantage of that. Higher-order logic cannot do that, but needs to under-approximate the mathematical universe as a tower of typed slices owing to Cantor's theorem (see Figure 1.2). The advantage of higher-order logic is its general mechanism of variable binding based on the lambda calculus. We call this feature **general binders**. First-order logic does not have general binders, but only two specific binding mechanisms, universal quantification and existential quantification. We can classify logics along these two dimensions, instead of their order, and Figure 3.15 shows the result. Interestingly, second-order logic is more like first-order logic than higher-order logic in that respect. Furthermore, the result tells us that abstraction logic combines the advantages of first-order logic and higher-order logic while avoiding their weaknesses by scoring in both dimensions. Note that abstraction logic does not achieve general binders via the lambda calculus, as this leads inevitably to a divided mathematical universe. Instead, abstraction logic embraces that operations and operators are not mathematical objects, but need to be represented through a separate mechanism, as illustrated in Figure 1.3.

This fusion of a uniform mathematical universe with general binders is powerful, and can be confusing initially. When abstraction logic was first presented at the UNILOG 2022 conference in Crete, an audience member wondered how the formula

$$\forall x.\, x$$

could ever mean anything sensible. Lemma 3.21 tells us that its meaning is *false*. The possibility of defining *false* like this has been noted as early as 1951 by Church [**34**], albeit Church let the quantifier range over booleans only. Much later in 2013 Kripke found it remarkable enough to exhibit it in a short note about Fregean first-order logic [**33**]. Unlike Church, and like us, Kripke lets the quantifier range over the entire universe, which he calls a *Fregean domain.*

Relationship to Established Logics. We have given a self-contained presentation of abstraction logic. What remains to be done is to clarify its relationship to established logics. We have seen that axioms and inference rules for predicate logic (Figure 3.11) are easily encoded in abstraction logic (Figure 3.10), but we have not yet gone beyond acknowledging this surface-level similarity. For example, is our completeness result for natural deduction strong enough to imply the established completeness results for known logics when they are encoded in abstraction logic?

Computer Algebra. The algebraic nature of abstraction logic seems particularly well-suited to finally allowing logic to make sizable inroads into computer algebra, which has resisted various such efforts based on conventional logics before.

Automation. Much research has been done on automating theorem proving in first-order logic. Significant work has also been done on automating higher-order logic [**35**]. Automating higher-order logic is more difficult than automating first-order logic, much of which is due to the complications that lambda calculus and type systems bring. It seems that abstraction logic would make a great context in which to study automation, given that neither lambda calculus nor type systems are built into it, yet it integrates both first-order and higher-order aspects.

Artificial Intelligence. Beyond conventional techniques for automation, AI has become a hot topic in machine-assisted proof [**36**]. Of particular interest is **autoformalization** [**37**], which means the process of translating informal mathematical prose into formal logic. Abstraction logic seems particularly well suited for autoformalization, as again, no type systems or towers of universes complicate the picture on the logical side.

AI not only seems to be destined to revolutionise the practical application of logic, but the reverse is also of increasing importance. The emergence of **reasoning models** [**38**] suggests that reasoning and thus logic will have a profound impact on modern AI as well. Of course, informal reasoning is not exactly the same as formal reasoning, but abstraction logic seems to be ideally positioned to bridge that gap.

Bibliography

[1] Steven Obua. *Practical Types.* https://doi.org/10.47757/practical.types.1, July 2021.

[2] Steven Obua. *Abstraction Logic.* https://doi.org/10.47757/abstraction.logic.2, November 2021.

[3] Steven Obua. *Philosophy of Abstraction Logic.* https://doi.org/10.47757/pal.2, December 2021.

[4] Steven Obua. *Automating Abstraction Logic.* https://doi.org/10.47757/aal.1, March 2022.

[5] Steven Obua. *Abstraction Logic: A New Foundation for (Computer) Mathematics.* https://arxiv.org/abs/2207.05610, July 2022.

[6] Steven Obua. *Logic is Algebra.* https://arxiv.org/abs/2304.00358, April 2023.

[7] Helena Rasiowa. *Introduction to Modern Mathematics.* https://doi.org/10.1016/C2013-0-11892-2, North-Holland 1973.

[8] Georg Grätzer. *Universal Algebra.* https://doi.org/10.1007/978-0-387-77487-9, Second Edition, Springer 1979.

[9] Georg Cantor. *Ueber eine elementare Frage der Mannigfaltigkeitslehre.* In *Jahresbericht der Deutschen Mathematiker-Vereinigung*, pp. 75-78. 1890/1891.

[10] Haskell B. Curry. *Some Philosophical Aspects of Combinatory Logic.* In *The Kleene Symposium*, pp. 85-101. North-Holland 1980.

[11] N. G. de Bruijn. *Lambda Calculus Notation with Nameless Dummies, a Tool for Automatic Formula Manipulation, with Application to the Church-Rosser Theorem.* https://doi.org/10.1016/1385-7258(72)90034-0, North-Holland 1972.

[12] Arthur Charguéraud. *The Locally Nameless Representation.* https://doi.org/10.1007/s10817-011-9225-2, Springer 2012.

[13] Franz Baader and Tobias Nipkow. *Term Rewriting and All That.* https://doi.org/10.1017/CBO9781139172752, Cambridge University Press 1998.

[14] Terence Tao. *Compactness and Contradiction.* https://doi.org/10.1090/mbk/081, American Mathematical Society, 2013.

[15] Nikolaos Galatos, Peter Jipsen, Tomasz Kowalski and Hiroakira Ono. *Residuated Lattices: An Algebraic Glimpse at Substructural Logics.* Elsevier 2007.

[16] Jorge Picado and Aleš Pultr. *Frames and Locales: Topology without Points.* Birkhäuser 2012.

[17] Cecylia Rauszer. *Semi-Boolean algebras and their applications to intuitionistic logic with dual operations.* http://eudml.org/doc/214696, 1974.

[18] Gonzalo E. Reyes and Houman Zolfaghari. *Bi-Heyting algebras, toposes and modalities.* https://doi.org/10.1007/BF00357841, 1996.

[19] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order.* Second Edition. Cambridge University Press, 2002.

[20] Katuzi Ono. *Reduction of logics to the primitive logic.* https://doi.org/10.2969/jmsj/01930384, 1967.

[21] Anil Nerode and Richard A. Shore. *Logic for Applications.* Second edition, Springer 1997.

[22] Errata for *Logic for Applications* [**21**], https://pi.math.cornell.edu/~shore/bookcorrections2023.pdf, 2023.

[23] Helena Rasiowa. *An Algebraic Approach to Non-Classical Logics.* North-Holland Publishing Company / Elsevier, May 1974.

[24] Kurt Gödel. *Zum Intuitionistischen Aussagenkalkül.* Anzeiger der Akademie der Wissenschaften in Wien, Vol. 69, pp. 65-66, 1932.

[25] Graham Priest. *Logic of Paradox.* https://doi.org/10.1007/BF00258428, 1979.

[26] Lawrence C. Paulson. *Isabelle: The Next 700 Theorem Provers.* https://doi.org/10.1007/BFb0012891, 1988.

[27] Tobias Nipkow, Markus Wenzel and Lawrence C. Paulson. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic.* https://doi.org/10.1007/3-540-45949-9, 2002.

[28] Markus Wenzel. *Isar — A Generic Interpretative Approach to Readable Formal Proof Documents.* https://doi.org/10.1007/3-540-48256-3_12, 1999.

[29] Lawrence C. Paulson. *The foundation of a generic theorem prover.* https://doi.org/10.1007/BF00248324, 1989.

[30] Chad E. Brown. *A semantics for intuitionistic higher-order logic supporting higher-order abstract syntax.* https://www.ps.uni-saarland.de/iholhoas/msethoas.pdf, 2014.

[31] Jaakko Hintikka. *Standard vs. Nonstandard Distinction: A Watershed in the Foundations of Mathematics.* In *From Dedekind to Gödel*, pp. 21-44. Synthese Library, Vol. 252. Springer 1995.

[32] William M. Farmer. *Simple Type Theory: A Practical Logic for Expressing and Reasoning About Mathematical Ideas.* https://doi.org/10.1007/978-3-031-21112-6, 2023.

[33] Saul A. Kripke. *Fregean Quantification Theory.* https://doi.org/10.1007/s10992-013-9299-x, 2013.

[34] Alonzo Church. *A formulation of the logic of sense and denotation.* https://archive.org/details/structuremethodm0000henl, 2013.

[35] Alexander Bentkamp, Jasmin Blanchette, Visa Nummelin, Sophie Tourret, Petar Vukmirović and Uwe Waldmann. *Mechanical Mathematicians.* `https://doi.org/10.1145/3557998`, 2023.

[36] Terence Tao. *Machine-Assisted Proof.* `https://doi.org/10.1090/noti3041`, 2024.

[37] Yuhuai Wu, Albert Q. Jiang, Wenda Li, Markus N. Rabe, Charles Staats, Mateja Jamnik and Christian Szegedy. *Autoformalization with Large Language Models.* `https://doi.org/10.17863/CAM.94956`, 2022.

[38] DeepSeek-AI. *DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning.* `https://doi.org/10.48550/arXiv.2501.12948`, January 2025.