LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

CHAIR OF THEORETICAL COMPUTER SCIENCE AND THEOREM PROVING

Bishop's Compilation Theorem

Luca Maio

Master's Thesis
Computer Science

Supervisors:
Priv-Doz. Dr. Iosif Petrakis,
Prof. Dr. Jasmin Blanchette

Submission Date: 14th January 2024

Disclaimer

I confirm that this Master's Thesis is my own work and I have documented all sources and material used.

Munich, 14th January 2024 Luca Maio

Acknowledgments

I would first like to thank my supervisor Iosif Petrakis for his immense patience and his constant guidance regarding both the creation of this thesis but also my further scientific career. Next, I would like to thank my future PhD supervisor Jasmin Blanchette who has also shown an exorbitant amount of patience regarding my progress and additionally supported me in the creation of this thesis despite it not being in his field of research. I would also like to thank my family, specifically my father for constantly showing interest in my work and for his support throughout both my Bachelor's and Master's degree and my fiancé, whose unconditional and unwavering love and support I could always count upon and who spared no effort to help me in whichever way she could. Finally I would like my good friend Lukas Bartl for continuously giving me feedback on my writing.

Without any of the aforementioned people this thesis would not have been possible for me to complete.

Abstract

Erret A. Bishop is well known in the community of constructive mathematicians for his work on informal constructive set theory and informal constructive analysis. However among his unpublished work, manuscripts have been found that, along some of his published work, concern themselves with formal foundations of his constructive mathematics. In this thesis we present two systems from this unpublished work. First a simply typed one called system Σ for which we also provide Bishop's concept of compilation, a re-discovery of core concepts of realizability theory. We then show that Σ is sound in regards to this through polishing the proof Bishop gave himself. Second we discuss his draft for a dependently typed system, which is built to reflect all of Bishop Set Theory.

Contents

1 Introduction

Erret Bishop's work on his informal system of constructive mathematics BISH [9, 2] is well known in the community of intuitionists and has been topic of active research and development ever since. In [19] a semi-formal system, the so called Bishop Set Theory is developed, which serves as a intermediate step between Bishop's original theory of sets and a formal system for it. In this thesis however we provide an overview of his less well known, even unpublished work on two formal systems that he developed for his informal system of constructive mathematics.

In this introductory section we will first explain why constructivism matters and why Bishop's approach to constructive mathematics is of interest. Moreover, we will briefly present notable formalization approaches by other mathematicians. In Section 2 we will then present the definitions to system Σ, the theoretical basis of Bishop's first approach to formalization. Following this in Section 3 we will state and present the proof of Bishop's fundamental result of [4], a soundness theorem for the concept of compilation he defined in order to finally achieve the translation from system Σ into ALGOL, an at the time prevalent imperative programming language, showing first ideas that could lead to mechanical proof verification or even proof assistants. In that section we will also see how his concept of compilation has great similarities to (modified) realizability but has key differences in concepts and intuition. Finally in Section 4 we will showcase the previously unreleased attempt of Bishop to expand the formalization to a full dependent type theory in order to represent the complete scope of BST. In that section we will also explore similarities and differences between Bishop's approach and the nowadays well known, but at the time not yet published work of Per Martin-Löf on his intuitionistic type theory known as Martin-Löf Type Theory (MLTT).

1.1 Contributions

Since this work is largely based on unpublished manuscripts of Bishop, here we would like to indicate what we have simply taken from these manuscripts, where we have made modifications and where we have added new things.

First of all, throughout this thesis we have updated notation, which was either deemed less readable by us or simply non-standard for modern times, since Bishop of course could not have participated the 50 years of development in type theory since him coming up with his notation. We have tried to make note of wherever we substantially altered the notation and why.

Next, the definitions of system Σ in Section 2 as well as the rules and axioms given there are taken from [4] without modifications to their meaning. We have newly introduced the distinct variable convention (DVC), see Section 2 into this system to make it less cumbersome, which eliminates the need for Bishop's generalized constants, terms and formulas, which we have still given the way Bishop did for completeness. Similarly in Section 4 we have also not changed the definitions of his general language or other definitions, unless specifically indicated otherwise, and even then only to adapt them to modern times.

Our main contribution comes in Section 3, where the basic proof of Bishop's Compilation Theorem is still taken from [4], but we have extracted one case into Lemma 1, since for this case we found Bishop's proof to be too vacuous. We have also fixed an apparent error in the last case of the proof, where instead of the decidable case distinction we have now contributed, Bishop performed a sort of meta-induction on natural numbers, while this principle is what is to be shown correct in that case, and also not making use of the structure specifically defined earlier for exactly this purpose. Additionally we have created tables that encapsulate the essence of the theorem for easier reference.

Lastly we have made a connection between Bishop's work on compilation and the work on realizability, as well as drawn parallels between these systems of his and some of his more philosophically minded work as to the canonical form of mathematical statements.

1.2 What is Constructive Mathematics?

Since the field of mathematics is very old, dating back to at least the ancient Greeks or even before, it has gone through a lot of evolution. This evolution has come in many forms, be it in contents, where mathematical theories were formed and refined continuously throughout the ages, in the way we do mathematics, where new proof methods are discovered and developed in turn with the needs of the evolving contents, or finally in the foundations of mathematics themselves.

This last change might be viewed as the most drastic one, stemming from the "Grundlagenkriese", which ignited when Bertrand Russel found an inconsistency in Cantor's informal set theory with his famous paradox. Up until this point, nearly all of mathematics of course had been subject to some form of rigour and formality, but its foundations were just thought to be intuited in some way or another, and their consistency taken as intuitive fact. The field of mathematical logic then strived to formalize mathematics from the very core, resulting in modern axiomatic set theories like Zermelo-Fränkel set theory with the axiom of choice (ZFC), which are now widely used by mathematicians. But with this formalization also came another split in the community, between the formalists, championed by Hilbert, and the intuitionists, led by Brouwer. While formalists were very much content with the new axiomatic theories, the intuitionists found issues with parts of it, namely the law of the excluded middle, $A \vee \neg A$ and related propositions, insisting that in order to show proper existence of an element, it is paramount that one produces one such element, not just disprove the non-existence. This idea of the intuitionists is the core of constructive logic and by extensions constructive, sometimes also intuitionistic (in contrast to "normal" or classical), mathematics in general. The main reason for this is that constructive proofs always contain actual information, also sometimes called computational content. Let us see a small example of a classical proof that showcases the information lost by using the law of the excluded middle:

Example 1. *There exist irrational numbers a, b such that a^b is rational.*

Proof. By the law of the excluded middle, $\sqrt{2}^{\sqrt{2}}$ is rational or not rational, i.e. irrational.

- **$\sqrt{2}^{\sqrt{2}}$ is rational:**
  We let $a = b = \sqrt{2}$, thus $a^b = \sqrt{2}^{\sqrt{2}}$, which by assumption is rational.

- $\sqrt{2}^{\sqrt{2}}$ **is irrational:**
  We let $a = \sqrt{2}^{\sqrt{2}}, b = \sqrt{2}$, resulting in $a^b = \sqrt{2}^{\sqrt{2}^{\sqrt{2}}} = \sqrt{2}^2 = 2$, proving our statement.

$\square$

Classically speaking this proof is completely acceptable, but in actuality it does not provide us with any information whatsoever. We now know there "exist" such a and b, but we still don't know any, since we have not actually proven the status of rationality of $\sqrt{2}^{\sqrt{2}}$. The actual proof of $\sqrt{2}^{\sqrt{2}}$ being irrational uses the Gelfond-Schneider theorem [18, Ch. X], the proof of which in turn is much more involved, showing that the deceptively simple proof of this proposition actually obscures the true effort required in order to truly know the result, at least as far as a constructivist point of view is concerned.

Another important point of view to consider in regards to constructive mathematics is that of a computer scientist. Computer scientists are interested in algorithms which can actually compute a concrete result in order to be able to use them in a more concrete fashion, either as further input or as end result. This approach is being applied, both by computer scientists as well as some mathematicians, to help prove mathematical theorems in an algorithmic manner, both using interactive [22, 7, 16] and fully automated [26, 10, 24] methods. For this then the constructive approach is of course very natural, as it always offers the possibility of extracting programs from such proofs, and sometimes it is even required in order to get the needed information, for example from a proof of existence, which in the classical way does not necessarily hold enough information to actually deduce a representative.

1.3 The State of Formalization

Formalization of mathematics has evolved greatly since Bishop wrote the manuscripts we concern ourselves with here. Starting with the intuitionistic type theory of Martin-Löf [17] there has been a lot of work to use dependent type theories as formal systems for mathematics. It is remarkable that Martin-Löf's work was only first published around five years after [4, 3] were most likely written, so specifically with the dependent type theory proposed in [3] in mind, which we discuss further in Section 4, Bishop would have possibly contributed much to constructive foundations of mathematics if it had not been for his early passing. After Martin-Löf's initial proposal resulting in MLTT as we know it today after some refinements, for example regarding the self-referential universe of MLTT, which resulted in paradoxical behaviour and could be solved by introducing the now used infinite hierarchy of universes, there has been the rather recent extension, first with work of Hoffmann and Streicher [12] and most notably through the introduction of the Univalence Axiom by Voevodsky [25], of MLTT to Homotopy Type Theory (HoTT) [23], which even ultimately led to Voevodsky winning the fields medal. Flavours of both MLTT and HoTT served as the basis for modern proof assistants like Agda [22] and Coq [7] but also predominantly classically minded proof assistants like Lean [16] were born from the ideas of dependent type theory and are seeing more and more active use by mathematicians. The first major result to famously be formalized, after checking of certain computer assisted parts of the proof turned out to be too involved to do by hand to gain complete certainty, was the four colour theorem by Appel and Haken [1] and since then proof assistants have become increasingly accepted not only by computer scientist but also by mainstream mathematicians. In essence these tools are the natural continuation of the thoughts Bishop expressed in [4], showing that the idea behind the concept of compilation, which we will explore in detail in Section 3, of using computers to formalize mathematics on the basis of types was at the forefront of its time, coming around the beginning of De Bruijn's Automath project [6] and maybe even preceding it. It is thus of immense interest to see what ideas of Bishop in this regard might lead to intuitions or developments that could possibly influence how we see proof systems and particularly formal foundations of mathematics in the modern day. We discuss this further in Section 3.4.

2 System Σ

In this section we will concern ourselves with the definition of the mathematical system Σ as well as discuss a canonical form for mathematical statements conjectured to exist by Bishop in [5]. The former was first given by Bishop in [5, 4], and inspired (in name and content) by Clifford Spector [21], who concerned himself with a system, essentially starting from slightly extended version of this Σ, iteratively giving a stronger language for proofs of recursive functionals of analysis. Bishop's version studied here is therefore a weaker system and might not be suitable to formalize the entirety of mathematics (something which Bishop attempted to rectify in [3], see Section 4), thus future work might concern itself with extending the ideas laid out later in Section 3 to a system of strength similar to Σ_4 of [21], or even adapting them to a more modern system like HoTT and see if profound differences exist between Bishop's approach and modern theorem proving approaches.

In Section 2.1 we present and motivate the types, terms and formulas of Σ, in Section 2.2 we then provide the axioms and rules of the calculus. Finally Section 2.3 concerns itself with Bishop's aforementioned supposed canonical form of mathematical theorems.

2.1 Types, Terms and Formulas

First a few basic definitions are in order for completeness:

Definition 1 (Bound and Free Variables). *An occurrence of a variable is said to be a bound occurrence in a term t or formula A if it occurs as $x_1, \ldots, x_n$ in a subterm of t of the form $\lambda_{x_1,\ldots,x_n}.t_1$ or as x in a subformula of A of the form $\forall_x A_1(x)$ or $\exists_x A_1(x)$.*

All occurrences that are not bound are said to be free occurrences.

Definition 2 (Distinct Variable Convention (DVC)). *All bound variables of a term t or formula A must have distinct names, as well as names distinct from all occurring free variables.*

In case this does not hold, it can be achieved through substituting fresh names for all bound variables. We will thus assume DVC to hold throughout.

The distinct variable convention is something not employed by Bishop, we however find this to be much more intuitive, especially for a formal approach, since the need to constantly mention how renaming or different variable assignment might impact the truth of some statement makes things overly cumbersome and approaches to signify that all possible variable assignments are considered, like the g-constants, -terms and -formulas Bishop uses, see Section 3.1 below, impair readability needlessly.

As a last remark, we would like to stress the difference between the two types of equalities we use throughout. The first is that of *definitional* equality, denoted by $\equiv$ and the second that of propositional equality, denoted by the standard $=$. This differentiation exists in many type systems, most notably in MLTT and HoTT and serves to distinguish between what can be described as a given rewrite rule, and a proposition that must first be proved and can only then be used with an inference rule to actually rewrite in context. The former of course implies the latter, of course not the other direction does not generally hold, and the former can only be reasoned about in a very limited way within the system itself.

Now we first present the type system of Σ.

$$\frac{}{\mathbb{Z} \text{ is the type of nonnegative integers}}\ (1T, Nat)$$

$$\frac{T_1, \ldots, T_n \text{ types}}{T_1 \times \cdots \times T_n \text{ type}}\ (2T, Prod)$$

$$\frac{T_1, T_2 \text{ types}}{T_1 \to T_2 \text{ type}}\ (3T, Fun)$$

Figure 1: Types of Σ

This type system, seen in Fig. 1, is rather basic, featuring the naturals (called $\mathbb{Z}$ in [4] or `[0]` in [5]), with 0 as the starting constant, as well as the type of all finite product and function types. As will become apparent in Section 3 this product is sometimes even used in a dependent way, even though Bishop does not treat this in a special manner. For the n-product type, Bishop used tuple notation, which we converted to a more standard product notation. Also he either used natural language or in [3] a sort of function $\mathbf{T}$ in order to indicate the typing of a term. We have opted to instead use standard modern $t : T$ notation, meaning term t is of type T.

The terms in the language Σ, see Fig. 2, consist at their base of arbitrary constants and variables, which need to be of a given type. Further, if we have an element of the function type we can create a new term as usual by applying this function to an argument of the appropriate type, note that Bishop used the unconventional notation $(t; t_1)$ for function application, which we modernized. In case we have a term t of the n-product type, there exist terms $\pi_i(t)$, called the i-th projection of t, for these Bishop also used peculiar notation, namely the same as for function application $(t; i)$, which we did not find to be helpful, opting to introduce π_i as a primitive function symbol instead as is often done nowadays. Conversely it is possible to construct a n-tuple from n appropriately typed terms. For the successor of natural numbers Bishop opted to use the notation of t', as is also used in [21]. A term of a function type is constructed with lambda notation, natively taking a n-tuple of variables, using $(6R)$ and $(7R)$ from Section 2.2 we see this can also be curried. In $(7t)$ $[u_1 \mapsto t_1, \ldots, u_n \mapsto t_n]$ will be equal to the functional t_i where $1 \leq i \leq n$ is the least integer such that $u_i \neq 0$ or t_n if $\forall_i u_i = 0$. Bishops original notation $[u_1 : t_1; \ldots u_n : t_n]$ is not problematic in and of itself. Nevertheless, we opted to change it however to avoid confusion with the colon now being used for typing declaration. Finally concerning $(8t)$ we define $\texttt{ind}(t_1, t_2, t_3, t_4)$ as follows:

$$\texttt{ind}(t_1, t_2, t_3, t_4) = \begin{cases} t_3 & \text{if } t_2 = t_1 \\ \texttt{ind}(t_1', t_2, t_4((t_1, t_3)), t_4) & \text{otherwise.} \end{cases}$$

This then gives us the tool to perform induction within our language.

$$\frac{}{0 : \mathbb{Z}}\ (0t, Zero)$$

$$\frac{x : T \text{ constant or variable}}{x : T \text{ term}}\ (1t, Var)$$

$$\frac{t : T_1 \to T_2 \qquad t_1 : T_1}{t(t_1) : T_2}\ (2t, App)$$

$$\frac{t : T_1 \times \cdots \times T_n}{\forall_{1 \leq i \leq n} \pi_i(t) : T_i}\ (3t, Project)$$

$$\frac{\forall_{1 \leq i \leq n} t_i : T_i}{(t_1, \ldots, t_n) : T_1 \times \cdots \times T_n}\ (4t, Pair)$$

$$\frac{t : \mathbb{Z}}{t' : \mathbb{Z}}\ (5t, Succ)$$

$$\frac{t : T \qquad \forall_{1 \leq i \leq n} x_i : T_i \text{ distinct variables}}{\lambda_{x_1, \ldots, x_n}.t : ((T_1 \times \cdots \times T_n) \to T)}\ (6t, Abst)$$

$$\frac{u_1, \ldots, u_n : \mathbb{Z} \qquad t_1, \ldots, t_n : T}{[u_1 \mapsto t_1, \ldots, u_n \mapsto t_n] : T}\ (7t, Functional)$$

$$\frac{t_1, t_2 : \mathbb{Z} \qquad t_3 : T \qquad t_4 : (\mathbb{Z}, T) \to T}{\texttt{ind}(t_1, t_2, t_3, t_4) : T}\ (8t, Ind)$$

Figure 2: Terms of Σ

$$\frac{t_1, t_2 : \mathbb{Z}}{t_1 = t_2, t_1 \neq t_2 \text{ formulas}}\ (1F, Eq)$$

$$\frac{A \text{ formula} \qquad x \text{ variable}}{\exists_x A, \forall_x A \text{ formulas}}\ (2F, Quant)$$

$$\frac{A, B \text{ formulas}}{A \wedge B, A \vee B, A \Rightarrow B \text{ formulas}}\ (3F, Base)$$

Figure 3: Formulas of Σ

Finally we present the definition of the formulas of Σ in Fig. 3. They are built inductively with equality/inequality of two terms of type $\mathbb{Z}$ as our basic prime formulas, or as Bishop calls them "simple formulas". We have the usual connectives $\wedge, \vee$ and $\Rightarrow$ as well as existential and universal quantifiers. (Bishop overloaded $\to$ for implication as well as function types, we chose to make these more visually distinct) When dealing with variables we apply the **generality interpretation**, meaning free variables are regarded to be universally quantified, i.e. $A(x_1, \ldots, x_n)$

means $\forall_{x_1,\ldots x_n} A(x_1, \ldots, x_n)$.

Primitively the language Σ only allows for (in-)equality between terms of type $\mathbb{Z}$, thus for terms of product and function types we define equality pointwise, meaning terms $f, g : T_1 \to T_2$ are viewed to be equal iff $f(\alpha) = g(\alpha)$ for any $\alpha : T_1$, and likewise terms $t, r : T_1 \times \cdots \times T_n$ are viewed to be equal iff $\pi_i(t) = \pi_i(r)$ for $1 \leq i \leq n$. With this inductive approach we can also define formulas $t_1 = t_2$, and analogously for $t_1 \neq t_2$, as follows:

Case $t_1, t_2 : T_1 \times \cdots \times T_n$:

$$t_1 = t_2 \Leftrightarrow \pi_1(t_1) = \pi_1(t_2) \wedge \cdots \wedge \pi_n(t_1) = \pi_n(t_2)$$

Case $t_1, t_2 : T_1 \to T_2$:

$$t_1 = t_2 \Leftrightarrow \forall_x (t_1(x) = t_2(x))$$

2.2 Inference Rules

In [4] Bishop gives the rules and axioms of Σ in the form of natural language (e.g. "If X then Y"). We have adapted this to the modern notation of inference rules. Also Bishop makes an effort of distinguishing between axioms and rules, saying the view of axioms as a rule without premises is not customary. We will in general not make this distinction, only sometimes referring to it for the sake of consistency. Thus going forward, "inference rules" equally address what Bishop calls axioms as well as rules.

The first set of inference rules in Fig. 4 form the, of course intuitionistic, propositional calculus with Gentzen style introduction and elimination rules for $\wedge, \vee$ and $\Rightarrow$, (un-)currying and the principle of explosion, *ex falso quodlibet.*

With Fig. 4 the system is then extended to the predicate calculus. Note here that x is a variable and t a term, where both have the same but arbitrary type. Originally Bishop posed the additional restriction for $(4A)$ and $(5A)$ that t be free for x, a condition we have made more explicit in general by requiring the DVC to hold at all times as to not have to deal with variable renaming explicitly, as well as these kinds of conditions.

An inference rule deserving of special mention is the constructive axiom of choice:

$$\frac{}{\forall_x \exists_y A(x, y) \Rightarrow \exists_z \forall_x A(x, z(x))}\ (6A, AC)$$

where if $x : T_1$ and $y : T_2$ we require $z : T_1 \to T_2$. This rule is of interest both since its constructive truth for arbitrary types T_1 is not immediate, as Bishop also discusses in [4], and also since in [5] Bishop conjectures that the form $\exists_y \forall_x A(x, y)$ is the canonical form for all mathematical statements. We will therefore continue the discussion of $(6A)$ in Section 2.3.

The inference rules in Fig. 6 concern themselves mostly with formalizing the intuitive meaning to the appropriate terms, specifically $(8A)$, $(9A)$ and $(10A)$, defining the behaviour of projection, application and the functional respectively. In rule $(7A)$ a property similar to the transport of MLTT is defined, where t_1 and t_2 are of the same arbitrary type and equality is as defined above, so $t_1 = t_2$ need not necessarily be a simple formula. Rule $(10R)$ provides the principle of induction for natural numbers and $(11R)$ gives the means to instantiate a proven existentially quantified formula

$$\frac{}{A \Rightarrow A}\ (1A, Base)$$

$$\frac{A}{B \Rightarrow A}\ (1R, True)$$

$$\frac{A \qquad A \Rightarrow B}{B}\ (2R, MP)$$

$$\frac{A \Rightarrow B \qquad B \Rightarrow C}{A \Rightarrow C}\ (3R, Trans)$$

$$\frac{}{A \wedge B \Rightarrow A \qquad B \wedge A \Rightarrow A \qquad A \Rightarrow A \vee B \qquad B \Rightarrow A \vee B}\ (2A, \vee + \ \wedge -)$$

$$\frac{A \Rightarrow C \qquad B \Rightarrow C}{A \vee B \Rightarrow C}\ (4R, \vee -)$$

$$\frac{C \Rightarrow A \qquad C \Rightarrow B}{C \Rightarrow A \wedge B}\ (5R, \wedge +)$$

$$\frac{A \Rightarrow B \Rightarrow C}{A \wedge B \Rightarrow C}\ (6R, Uncurry)$$

$$\frac{A \wedge B \Rightarrow C}{A \Rightarrow B \Rightarrow C}\ (7R, Curry)$$

$$\frac{}{0 = 0' \Rightarrow A}\ (3A, EfQ)$$

Figure 4: Axioms and rules of Σ forming the propositional calculus

$$\frac{A \Rightarrow B(x) \qquad x \text{ not free in A}}{A \Rightarrow \forall_x B(x)}\ (8R, \forall +)$$

$$\frac{B(x) \Rightarrow A \qquad x \text{ not free in A}}{\exists_x B(x) \Rightarrow A}\ (9R, \exists +)$$

$$\frac{}{\forall_x A(x) \Rightarrow A(t)}\ (4A, \forall Base)$$

$$\frac{}{A(t) \Rightarrow \exists_x A(x)}\ (5A, \exists Base)$$

Figure 5: Axioms and rules of Σ forming the predicate calculus

with the constant c_0, which represents the functional constructed "according to the proof P of $\exists_x A(x)$"[4], which we may in general also denote as $\alpha(P)$ later.

The final set of inference rules in 7 enables the behaviour of equality as expected, where all variables are of type $\mathbb{Z}$ and further equality is again defined inductively as

$$\frac{}{(t_1 = t_2 \wedge A(t_1)) \Rightarrow A(t_2)}\ (7A, Transport)$$

$$\frac{m, n : \mathbb{Z} \qquad 1 \leq m \leq n}{\pi_m((t_1, \dots, t_n)) = t_m}\ (8A, Proj)$$

$$\frac{}{(\lambda_{x_1, \dots x_n}.t(x_1, \dots, x_n))(t_1, \dots, t_n) = t(t_1, \dots, t_n)}\ (9A, App)$$

$$\frac{i, n : \mathbb{Z} \qquad 1 \leq i \leq n}{u_1 = 0 \wedge \dots \wedge u_{i-1} = 0 \wedge u_i \neq 0 \Rightarrow [u_1 \mapsto t_1, \dots, u_n \mapsto t_n] = t_i}\ (10A, Functional)$$

$$\frac{x : \mathbb{Z} \qquad A(0) \qquad A(x) \Rightarrow A(x')}{A(x)}\ (10R, Ind)$$

$$\frac{\exists_x A(x) \qquad x_1, \dots, x_n \text{ all free variables of } A}{A(c_0(x_1, \dots, x_n))}\ (11R, \exists Proof)$$

Figure 6: Further axioms and rules of Σ

$$\frac{}{x = x \qquad (x = y \wedge z = y) \Rightarrow x = z}\ (12A, Trans)$$

$$\frac{}{x = y \Rightarrow t(x) = t(y)}\ (13A, Ext)$$

$$\frac{}{x' = 0 \Rightarrow 0' = 0}\ (14A, False)$$

$$\frac{}{x' = y' \Rightarrow x = y \qquad x = y \Rightarrow x' = y'}\ (15A, +Ext)$$

$$\frac{}{x = y \vee x \neq y \qquad (x = y \wedge x \neq y) \Rightarrow 0' = 0}\ (16A, False)$$

Figure 7: Axioms for $\mathbb{Z}$ in Σ

given above.

2.3 Canonical Form of Mathematical Statements

In this section we will discuss Bishop's argument as to the previously alluded to canonical form for all (constructive) mathematical statements that he posits in [5], its relation to the numerical meaning of implication and further how it might relate to the constructive axiom of choice, $(6A)$ in Σ.

First, Bishop distinguishes between *complete* and *incomplete* mathematical statements, the differentiating factor being whether or not the statement contains all definitions needed for it. One can easily see that most statements are therefore deemed incomplete, making it sensible to talk about the canonical form for these in particular. Bishop conjectures that such a statement can be said to depend on a finite amount of pre-existing assumptions, which therefore should be constructible following some finite rules. Thus, Bishop posits, these parameters are existentially quantified, we shall call them by y for ease of use, even though y may have more than one component. The statement can then be said to be equivalent to $\exists_y P(y)$, where this $P(y)$ is then complete in Bishop's sense. Such complete statements, Bishop argues, should semantically assert "that a given constructively defined function f, from a given constructively defined set S to the integers, vanishes identically. In other words, it asserts $\forall_x A(x)$, where A is the decidable predicate $f(x) = 0$ and x ranges over S"[5, p. 57]. This interpretation of constructivism is especially valuable in our opinion when dealing with machine computations, as the integers are the primitive components of the language of computation.

Joining these definitions we obtain this canonical form for incomplete mathematical statements:

$$\exists_y \forall_x A(x, y) \tag{1}$$

where $x : S$ and $y : T$.

Granting the mere existence of a canonical form for all mathematical statements, and it even being this one in particular, may at first seem to be a big and unwarranted commitment to make. Let us thus, departing for the moment from the focus of Bishop as to the numerical meaning of implication, briefly discuss the relationship between the constructive axiom of choice and (1), which will hopefully make it more believable.

Let's first see as to why, at least within the framework that concerns us here, where everything is finally grounded in natural numbers, we may even want to regard $(6A)$ to intuitively be true in a constructive sense, in the manner of Bishop [4, p. 9-10]. Given $\forall_x \exists_y A(x, y)$, where $x : T_1, y : T_2$, we can for all functionals $f : T_1$, since we have a proof of $\exists_y A(f, y)$, construct some functional $\tilde{f} : T_2$, such that $A(f, \tilde{f})$. To now define a functional $h : T_1 \to T_2$, the first instinct would be $h(f) := \tilde{f}$, and this can be made to work, depending on the kind of types one permits as T_1 and T_2. Specifically we need to be able to convert the elements of the types into a fixed representation, for example binary for natural numbers, so that we can always guarantee if $f_1 = f_2$ then $h(f_1) = h(f_2)$. For higher types this fixed representation, into which all representatives of the type can be converted, may not necessarily exist, but Bishop conjectures, that "However, it is intuitively very plausible that for sets of the types being considered every constructively definable operation is in fact a function" [4, p. 10]. In general since the time of writing of Bishop's manuscripts, the specific version of choice posed in $(6A)$ can be shown provable for example in MLTT (not implying the law of the excluded middle, which versions in even constructive set theory do), which can be viewed as further evidence of the constructive truth of $(6A)$ in this context.

Now using the constructive axiom of choice, we can see that formulas of the form of (1) can be achieved as the consequence of the former from any formula of the form $\forall_x \exists_y A(x, y)$. This form now, is always easily achieved, since the inverse operation to $(6A)$ is undoubtedly true. Now once all quantifiers in possibly mixed order have been ordered to first have all universal and only then all existential quantifiers, $(6A)$ provides us with the sought after form, providing further justification to the existence of this canonical form.

Now regarding the numerical meaning of implication, Bishop [5] transforms the starting canonical form of the implication $P \Rightarrow Q$,

$$\exists_y \forall_x A(x, y) \Rightarrow \exists_v \forall_u B(u, v) \tag{2}$$

into a statement the following:

$$\forall_y \exists_v \forall_u (\forall_x A(x, y) \Rightarrow B(u, v)). \tag{3}$$

This critically rests on the claim made, that

$$C \Rightarrow \exists_r D \tag{4}$$

shall imply the statement

$$\exists_r (C \Rightarrow D). \tag{5}$$

Bishop justifies this implication, saying that while the procedure of construction of this r may indeed assume the truth of C, it is still possible to give a universally valid definition of r, by assigning some convenient constant to it if C is not known to be true at the time, "with the property that in case [C] does hold, then [r] will have the value originally prescribed" [5, pp. 58] (Bishop discusses this directly with subformulas of (2) and v, we decided however to keep it more general).

Continuing from (3), fixing values for y, v and u, Bishop goes on to postulate, that in a constructive context, experience indicates that a proof of a statement

$$\forall_x A(x, y) \Rightarrow B(u, v) \tag{6}$$

typically only relies on finitely many $x_1, \ldots, x_n$ for which $A(x_i, y)$ holds in order to prove $B(u, v)$. This assumption further transforms the formula into

$$A(x_1, y) \wedge \cdots \wedge A(x_n, y) \Rightarrow B(u, v), \tag{7}$$

from where, since it is a finitary statement of only decidable propositions it must hold, that there exists $1 \leq k \leq n$ such that

$$A(x_k, y) \Rightarrow B(u, v). \tag{8}$$

This then means it would hold, that

$$\exists_x (A(x, y) \Rightarrow B(u, v)), \tag{9}$$

essentially replacing the universal by an existential quantifier.

This finally means (3) can be transformed into

$$\forall_y \exists_v \forall_u \exists_x (A(x, y) \Rightarrow B(u, v)), \tag{10}$$

and if we then apply ($6A$), we finally receive what Bishop views as the candidate for the canonical numerical version of $P \Rightarrow Q$,

$$\exists_{\bar{v}} \exists_{\bar{x}} \forall_y \forall_u (A(\bar{x}(y, u), y) \Rightarrow B(u, \bar{v}(y))). \tag{11}$$

As future work it remains to actually define and use a system with (11) as primitive definition of implication, called *numerical implication* by Bishop, possibly even in the context of the dependent type theory we will discuss in Section 4.

3 Bishop's Compilation Theorem

In this section, building on the system Σ, as defined in the section above, defining a notion of compilation, which very closely relates to realizability [14, 13] and modified realizability [14], constituting a mixture of both in some ways, and then giving a soundness theorem, called compilation theorem by Bishop, for this notion, which was contained in [4] but we wrote some parts out in more detail, gave some extra supporting proofs explicitly and fixed the proof of one of the cases.

3.1 Definitions

In [4], Bishop defines the concepts of generalized, or g-, constants, formulas and terms. For the sake of completeness we include these below. However since we opted to use DVC, use of these is not necessary, since leaving all free variables in their current form, may no longer lead to conflicts that may change the meaning of the original term or formula.

Definition 3 (generalized constant). *A generalized constant of a specific type is a representation of a specific functional of the given type.*

Definition 4 (generalized formula). *We define a generalized formula, or g-formula, A to be derived from a (non-generalized) term A_0 of Σ by substituting generalized constants of the appropriate types for certain of the free variables of A_0.*

Definition 5 (generalized term). *We define a generalized term, or g-term, t to be derived from a (non-generalized) term t_0 of Σ by substituting generalized constants of the appropriate types for certain of the free variables of t_0.*

While generally only terms have types, for the notion of compilation which we will introduce shortly, it is natural and helpful to define the notion of the type of a (g-)formula, which will then be the type of the term that compiles this formula.

Definition 6 (Type of (g-)formulas). *To each (g-)formula A we associate the inductively defined type $T(A)$. We now inductively define $T(A)$:*

1. *In case A does not contain any $\exists$ or $\vee$ (i.e. is existence-free), specifically if it is a simple formula, $T(A) = \mathbb{Z}$. For all other cases A is assumed to contain at least one of $\exists$ or $\vee$ (i.e. is existential).*

2. *In case $A \equiv A_1 \wedge A_2$, then $T(A) \equiv (T(A_1), T(A_2))$.*

3. *In case $A \equiv A_1 \vee A_2$, then $T(A) \equiv (\mathbb{Z}, T(A_1), T(A_2))$.*

4. *In case $A \equiv A_1 \Rightarrow A_2$, then $T(A) \equiv T(A_1) \rightarrow T(A_2)$.*

5. *In case $A \equiv \exists_x A_1(x)$, then $T(A) \equiv (T(x), T(A_1(x)))$, where $T(x)$ is the type of x.*

6. *In case* $A \equiv \forall_x A_1(x)$*, then* $T(A) \equiv T(x) \to T(A_1(x))$*.*

Next we finally present Bishop's definition of compilation. Note that Bishop originally did not include a case for simple formulas, which we proceeded to add in a style that resembles realizability. The relationship between compilation and realizability will be subject of further discussion in the following Section 3.3.

Definition 7 (Compilation)**.** *We inductively define what it means for a generalized term* t *of type* $T(A)$ *to compile a constructive proof* P *of a generalized formula* A*, written* $t\ \boldsymbol{c}\ P^A$*:*

1. *In case* A *has free variables, then* $t\ \boldsymbol{c}\ P^A$ *iff for each assignment* a *of a functional of the requisite type to each of the free variables of* A*,* $t_a\ \boldsymbol{c}\ {P_a}^{A_a}$*. For the remaining cases we may assume that* A *has no free variables.*

2. *In case* A *is a simple formula, i.e.* $A \equiv t_1 = t_2$ *or* $A \equiv t_1 \neq t_2$*, or* A *is existence-free,* $t\ \boldsymbol{c}\ P^A$ *iff* $t : \mathbb{Z}$ *and* P *constructively proves the equality/inequality, i.e.* A *is true.*

3. *In case* $A \equiv A_1 \wedge A_2$*, then* $t\ \boldsymbol{c}\ P^A$ *iff* $\pi_i(t)\ \boldsymbol{c}\ {P_i}^{A_i}$*, where* P_i *is the corresponding proof of* A_i *which can be inferred from* P *for* $i \in \{1,2\}$*.*

4. *In case* $A \equiv A_1 \vee A_2$*, then* $t\ \boldsymbol{c}\ P^A$ *iff* $\pi_{i+1}(t)\ \boldsymbol{c}\ {P_i}^{A_i}$*, where* $i = 1$ *iff* $\pi_1(t) = 0$ *and* $i = 2$ *iff* $\pi_1(t) \neq 0$ *and* P_i *is the corresponding proof of* A_i *which can be inferred from* P*.*

5. *In case* $A \equiv A_1 \Rightarrow A_2$*, then* $t\ \boldsymbol{c}\ P^A$ *iff whenever* $t_1\ \boldsymbol{c}\ {P_1}^{A_1}$*, it also holds that* $t(t_1)\ \boldsymbol{c}\ {P_2}^{A_2}$*.*

6. *In case* $A \equiv \exists_x A_1(x)$*, then* $t\ \boldsymbol{c}\ P^A$ *iff* $\pi_1(t) = \alpha(P)$ *(for all values of free variables of* $\pi_1(t)$*, if any) and* $\pi_2(t)\ \boldsymbol{c}\ P_1(\alpha(P))^{A_1(\alpha(P))}$*.*

7. *In case* $A \equiv \forall_x A_1(x)$*, then* $t\ \boldsymbol{c}\ P^A$ *iff* $t(x)\ \boldsymbol{c}\ P_1(x)^{A_1(x)}$*.*

3.2 Compilation Theorem

Bishop's fundamental result in [4] is the following version of a soundness theorem relating the above notion of compilation with his notion of an informal constructive proofs.

Theorem 1 (Compilation Theorem)**.** *There exists an algorithm* $\boldsymbol{AL}$*, realizable in any general string-manipulation programming language, such that*

$$\vdash_{\Sigma}^{P^A} A \Rightarrow \boldsymbol{AL}(P^A)\boldsymbol{c}P^A$$

where $\boldsymbol{AL}(P^A)$ *is a constant free term of* Σ*.*

Algorithm	Axiom/Rule	Term
AL1	1A	$t \equiv \lambda_x.x$
AL2	1R	$t(\beta) \equiv \lambda_x.\beta$
AL3	2R	$t(\beta, \gamma) \equiv \gamma(\beta)$
AL3	3R	$t(\beta, \gamma) \equiv \lambda_x.\gamma(\beta(x))$
AL1	2A$\wedge$	$t \equiv \lambda_x.\pi_1(x)$
AL1	2A$\vee$	$t \equiv \lambda_x.(0, x, y)$
AL3	4R	$t(\beta, \gamma) \equiv \lambda_x.[\pi_1(x) \mapsto \gamma(\pi_3(x)), 1 \mapsto \beta(\pi_2(x))]$
AL3	5R	$t(\beta, \gamma) \equiv \lambda_x.(\beta(x), \gamma(x))$
AL2	6R	$t(\beta) \equiv \lambda_x.(\beta(\pi_1(x)))(\pi_2(x))$
AL2	7R	$t(\beta) \equiv \lambda_{x,y}.\beta((x, y))$
AL1	3A	$t \equiv \lambda_x.y$
AL2	8R	$t(\beta) \equiv \lambda_{y,x}.\beta(y)$
AL2	9R	$t(\beta) \equiv \lambda_y.((\lambda_x.\beta)(\pi_1(y)))(\pi_2(y))$
AL1	4A	$t \equiv \lambda_y.y(w)$
AL1	5A	$t \equiv \lambda_y.(w, y)$
AL1	6A	$t \equiv \lambda_z.(\lambda_x.\pi_1(z(x)), \lambda_x.\pi_2(z(x)))$
AL1	7A	$t \equiv \lambda_z.\pi_2(z)$
AL3	10R	$t(\beta, \gamma) \equiv \texttt{ind}(0, x, \beta, \lambda_2.\gamma(\pi_1(w)))$
AL2	11R	$t(\beta) \equiv \pi_2(\beta)$

Table 1: AL

Proof. Like Bishop we actually present three algorithms **AL1**, **AL2** and **AL3** which in turn define the algorithm **AL**. The constructed terms are listed as an overview in Table 2, Table 3 and Table 4 respectively and joined for the sake of completeness Table 1. where we decided to omit the premises and conclusions to aid readability. These can be cross-referenced from the respective algorithm below. It remains to justify for each of the listed terms that t **c** P^A indeed holds, for which we will consider each algorithm, and within that each rule, in turn. When doing this, we will justify before each of them respectively, that the algorithms actually satisfy our needs, using Bishop's arguments.

3.2.1 AL1

Axiom	Statement	Term
1A	$A \equiv A' \Rightarrow A'$	$t \equiv \lambda_x.x$
2A$\wedge$	$A \equiv A' \wedge A'' \Rightarrow A'$	$t \equiv \lambda_x.\pi_1(x)$
2A$\vee$	$A \equiv A' \Rightarrow A' \vee A''$	$t \equiv \lambda_x.(0, x, y)$
3A	$A \equiv 0 = 0' \Rightarrow A'$	$t \equiv \lambda_x.y$
4A	$A \equiv \forall_x A'(x) \Rightarrow A'(w)$	$t \equiv \lambda_y.y(w)$
5A	$A \equiv A'(w) \Rightarrow \exists_x A'(x)$	$t \equiv \lambda_y.(w, y)$
6A	$A \equiv \forall_x \exists_y A'(x, y) \Rightarrow \exists_z \forall_x A'(x, z(x))$	$t \equiv \lambda_z.(\lambda_x.\pi_1(z(x)), \lambda_x.\pi_2(z(x)))$
7A	$A \equiv (t_1 = t_2 \wedge A'(t_1)) \Rightarrow A'(t_2)$	$t \equiv \lambda_z.\pi_2(z)$

Table 2: AL1

First for **AL1** we are only dealing with axioms, thus not needing any outside hypotheses.

If now $\mathbf{AL1}(P^A) = t$, this t might still contain constants c_i occurring in A. In this case, since these constants must be introduced by $(11R)$, which is handled by **AL2**, thus, **AL** in its entirety must already have given a term t_i, for which $t_i \ \mathbf{c}\ P_i^{\exists_{x_i} A_i(x_i)}$ holds and which thus is equal to c_i but constant free. Thus the whole constant free term is obtained by replacing the c_i by t_i in each occurrence.

Now that we have seen **AL1** to function as desired, we will define it and prove $t \ \mathbf{c}\ P^A$ in turn for each axiom.

(**1A**): It suffices to show that given $\bar{t} \ \mathbf{c}\ P'^{A'}$ holds, it follows that $t(\bar{t}) \ \mathbf{c}\ P''^{A'}$ also holds. We obtain this from $t(\bar{t}) \equiv \bar{t}$ and the fact that P' and P'' must in fact be the same in the sense given above.

(**2A**$\wedge$): It suffices to show that given $s \ \mathbf{c}\ L^{A' \wedge A''}$ holds, it follows that $t(s) \ \mathbf{c}\ P'^{A'}$ also holds. By definition of t,

$$t(s) \equiv \pi_1(s),$$

which in turn by definition behaves as desired.

The term and argument follow analogously for $A \equiv A' \wedge A'' \Rightarrow A''$.

(**2A**$\vee$): It suffices to show that given $\bar{t} \ \mathbf{c}\ P'^{A'}$ holds, it follows that $t(\bar{t}) \ \mathbf{c}\ S^{A' \vee A''}$ also holds. By definition of t,

$$t(\bar{t}) \equiv (0, \bar{t}, y),$$

where y is any variable of type $T(A'')$, and since $\pi_1((0, \bar{t}, y)) \equiv 0$ means that S is actually a proof of A', thus it remains to show $\pi_2(t(\bar{t})) \ \mathbf{c}\ P''^{A'}$, which holds, since $\pi_2((0, \bar{t}, y)) \equiv \bar{t}$ and P'' and P' must be the same.

The term and argument follow analogously for $A \equiv A'' \Rightarrow A' \vee A''$.

(**3A**): It suffices to show that given $u \ \mathbf{c}\ R^{0=0'}$ holds, it follows that $t(u) \ \mathbf{c}\ P'^{A'}$ also holds. Bishop proves this in a vacuous way, postulating that there are no proofs of $0 = 0'$.

(**4A**): It suffices to show that given $\bar{t} \ \mathbf{c}\ P'^{\forall_x A'(x)}$ holds, it follows that $t(\bar{t}) \ \mathbf{c}\ P''^{A'(w)}$ also holds. By definition of t,

$$t(\bar{t}) \equiv \bar{t}(w).$$

By the definition of compilation from $\bar{t} \ \mathbf{c}\ P'^{\forall_x A'(x)}$ it follows that $\bar{t}(f) \ \mathbf{c}\ P'^{A'(f)}$ holds for all functionals f, thus also for w.

(**5A**): It suffices to show that given $\bar{t} \ \mathbf{c}\ P'^{A'(w)}$ holds, it follows that $t(\bar{t}) \ \mathbf{c}\ P''^{\exists_x A'(x)}$ also holds. By definition to show $t(\bar{t}) \ \mathbf{c}\ P''^{\exists_x A'(x)}$, we need to show $\pi_1(t(\bar{t})) = \alpha(P'')$ as well as $\pi_2(t(\bar{t})) \ \mathbf{c}\ P'''(\alpha(P''))^{A'(\alpha(P''))}$. Also by definition of t,

$$t(\bar{t}) \equiv (w, \bar{t})$$

and thus $\pi_1((w, \bar{t})) = w$, which fulfils the first condition, since $\alpha(P'') = w$. For the second condition we have $\pi_2((w, \bar{t})) = \bar{t}$, which by assumption $\bar{t} \ \mathbf{c}\ P'^{A'(w)}$ compiles a proof of $A'(\alpha(P''))$, which must in turn be the same proof as P'.

(**6A**): While well-typedness has been straightforward so far, for this term it might not immediately be apparent, thus we will first argue this and only then prove the main property.

Looking at Definition 6, we see that for the first element we need an element of the same type as z from the existential. This then is a function from the type of x to the type of y from the existential in the premise. Thus we create a function, abstracting over x, thus having the correct domain type, applying x to our argument function and recovering the generated functional of the type of y from the given existential. The second element of the dependent pair needs to have the same type as $\forall_x A'(x, z(x))$, which again is a function type by Definition 6, again with the same domain but we need something of the same type as $A'(x, z(x))$, which since $z(x)$ is of the same type as y, is the same type as $A'(x, y)$, which we recover as the second projection of our argument function applied on x.

Now to show that $t \mathbf{c} P^{\forall_x \exists_y A'(x,y) \Rightarrow \exists_z \forall_x A'(x,z(x))}$ holds, it suffices to show that given $\bar{t} \mathbf{c} P'^{\forall_x \exists_y A'(x,y)}$ holds, then $t(\bar{t}) \mathbf{c} P''^{\exists_z \forall_x A'(x,z(x))}$ also holds. By definition to show $t(\bar{t}) \mathbf{c} P''^{\exists_z \forall_x A'(x,z(x))}$, we need to show that $\pi_1(t(\bar{t})) = \alpha(P'')$ and $\pi_2(t(\bar{t})) \mathbf{c} P'''^{\forall_x A'(x,z(x))}$. By definition $\pi_1(t(\bar{t})) = \lambda_x.\pi_1(\bar{t}(x))$. Since equality of functions is defined pointwise we show that for each functional f of the same type as x, $\pi_1(\bar{t}(f)) = \alpha(P'')(f)$. Since $\bar{t} \mathbf{c} P'^{\forall_x \exists_y A'(x,y)}$, by definition we have $\pi_1(\bar{t}(f)) \equiv \alpha(P'(f))$, where $P'(f)$ proves $\exists_y A'(f, y)$ and since $\alpha(P'')(f) = \alpha(P'(f))$ we have shown the first condition to hold.

For the second condition, by definition of t, $\pi_2(t(\bar{t})) = \lambda_x.\pi_2(\bar{t}(x))$. Again by definition of compilation, from $\bar{t} \mathbf{c} P'^{\forall_x \exists_y A'(x,y)}$, we know that for any functional f of the same type as x $\pi_2(\bar{t}(f)) \mathbf{c} P''''(\alpha(P'(f)))^{A'(f,\alpha(P'(f)))}$ holds. To show the second condition it now just remains to show $\pi_2(\bar{t}(f)) \mathbf{c} S^{A'(f,\alpha(P'')(f))}$ also holds, which must be the case, since S and $P''''(\alpha(P'(f)))$ must be the same proof and $\alpha(P'')(f) = \alpha(P'(f))$.

(**7A**): It suffices to show that given $\bar{t} \mathbf{c} P'^{t_1=t_2 \wedge A'(t_1)}$ holds, it follows that $t(\bar{t}) \mathbf{c} P''^{A'(t_2)}$ also holds. By definition $\bar{t} \mathbf{c} P'^{t_1=t_2 \wedge A'(t_1)}$ means $\pi_1(\bar{t}) \mathbf{c} S^{t_1=t_2}$ and $\pi_2(\bar{t}) \mathbf{c} S'^{A'(t_1)}$ and also by definition of t,

$$t(\bar{t}) \equiv \pi_2(\bar{t}).$$

Thus what is left to be shown is $\pi_2(\bar{t}) \mathbf{c} P''^{A'(t_2)}$, which holds due to $t_1 = t_2$ and Lemma 1.

3.2.2 AL2

Next for **AL2** we consider all rules with exactly one premise. Thus we will for all cases assume that for the listed formula B in Table 3 there is some term $u : T(B)$ such that $u \mathbf{c} Q^B$. In the terms we define for the conclusion we use β as the parameter that will be replaced by u. In cases (8R) and (9R), where u depends on some x, we denote this through square brackets $u[x]$, thus for some functional $f : T(x)$, $u[f]$ denotes the term $u[x]$, where all occurrences of x have been replaced by f. Note that the DVC can always be fulfilled by renaming bound variables that conflict with f or choosing an encoding like De Bruijn indices [8] which does not use variable names, thus we will not further discuss this.

Rule	Premise	Conclusion	Term
1R	$B \equiv B$	$A \equiv B' \Rightarrow B$	$t(\beta) \equiv \lambda_x.\beta$
6R	$B \equiv D \Rightarrow (E \Rightarrow F)$	$A \equiv D \wedge E \Rightarrow F$	$t(\beta) \equiv \lambda_x.(\beta(\pi_1(x)))(\pi_2(x))$
7R	$B \equiv D \wedge E \Rightarrow F$	$A \equiv D \Rightarrow (E \Rightarrow F)$	$t(\beta) \equiv \lambda_{x,y}.\beta((x,y))$
8R	$B \equiv E \Rightarrow F(x)$	$A \equiv E \Rightarrow \forall_x F(x)$	$t(\beta) \equiv \lambda_{y,x}.\beta(y)$
9R	$B \equiv E(x) \Rightarrow F$	$A \equiv \exists_x E(x) \Rightarrow F$	$t(\beta) \equiv \lambda_y.((\lambda_x.\beta)(\pi_1(y)))(\pi_2(y))$
11R	$B \equiv \exists_x B'(x)$	$A \equiv B'(c_0(x_1,\ldots,x_n))$	$t(\beta) \equiv \pi_2(\beta)$

Table 3: AL2

If now $\mathbf{AL2}(P^A) = t(\beta)$, by induction hypothesis a term u such that u **c** Q^B is already defined. Thus, the term $t(u)$ by assumption compiles P^A.

Now that we have seen **AL2** to function as desired, we will define it and prove t **c** P^A in turn for each single premise rule.

(**1R**): To show $t(u)$ **c** P^A holds, it suffices to show that given $\bar{u}$ **c** $Q'^{B'}$ holds, it follows that $(t(u))(\bar{u})$ **c** Q''^B also holds. By definition of t,

$$(t(u))(\bar{u}) \equiv u$$

and since u **c** Q^B also u **c** Q''^B since Q and Q'' must be the same proof.

(**6R**): To show $t(u)$ **c** P^A holds, it suffices to show that given w **c** $R^{D\wedge E}$ holds, it follows that $(t(u))(w)$ **c** S^F also holds. Now by definition of t,

$$(t(u))(w) \equiv (u(\pi_1(w)))(\pi_2(w))$$

and $\pi_1(w)$ **c** R'^D as well as $\pi_2(w)$ **c** R''^E hold. From this we get $u(\pi_1(w))$ **c** $S'^{E\Rightarrow F}$ and thus $(u(\pi_1(w)))(\pi_2(w))$ **c** S''^F showing $(t(u))(w)$ **c** S^F since S and S'' must be the same proof.

(**7R**): To show $t(u)$ **c** P^A holds, it suffices to show that given w_1 **c** R'^D holds, it follows that $(t(u))(w_1)$ **c** $S'^{E\Rightarrow F}$ also holds. For this in turn it suffices to show that given w_2 **c** R''^E holds, it follows that $((t(u))(w_1))(w_2)$ **c** S^F also holds. Now by definition of t,

$$((t(u))(w_1))(w_2) \equiv (u((w_1, w_2)))$$

for which using the assumption u **c** Q^B, we get that $u((w_1, w_2))$ **c** S''^F holds, showing that $((t(u))(w_1))(w_2)$ **c** S^F, since S and S'' must be the same proof.

(**8R**): To show $t(u)$ **c** P^A holds, it suffices to show that given w **c** R^E holds, it follows that $(t(u[x]))(w)$ **c** $S^{\forall_x F(x)}$ also holds. By definition of compilation it now suffices to show that for each functional $f : T(x)$, $(t(u[f]))(w)$ **c** $S'^{F(f)}$ holds. By definition of t,

$$(t(u[f]))(w) \equiv u[f](w).$$

By assumption $u[x]$ **c** $Q^{E\Rightarrow F(x)}$ and thus $u[f]$ **c** $Q'^{E\Rightarrow F(f)}$. From this follows by definition that $u[f](w)$ **c** $S''^{F(f)}$ showing $(t(u[f]))(w)$ **c** $S'^{F(f)}$, since S' and S'' must be the same proof.

(**9R**): To show $t(u)$ **c** P^A holds, it suffices to show that given w **c** $R^{\exists_x E(x)}$ holds, it follows that $(t(u[x]))(w)$ **c** S^F also holds. By definition of t,

$$(t(u[x]))(w) \equiv u[\pi_1(w)](\pi_2(w)).$$

By the definition of compilation, we know $\pi_1(w)$ must be equal to $\alpha(R)$ and from the assumption u **c** Q^B, it then follows that both $u[\pi_1(w)]$ **c** $Q'^{E(\pi_1(w))\Rightarrow F}$ and $u[\alpha(R)]$ **c** $Q'^{E(\alpha(R))\Rightarrow F}$ hold respectively due to the equality. Also from the definition of compilation, we have $\pi_2(w)$ **c** $R'^{E(\alpha(R))}$. From all this follows that $u[\pi_1(w)](\pi_2(w))$ **c** S'^F, showing $(t(u[x]))(w)$ **c** S^A holds, since S' and S must be the same proof.

(**11R**): To show $t(u)$ **c** P^A holds, by definition we need to show $\pi_2(u)$ **c** P^A holds. By the assumption u **c** Q^B and the definition of compilation, $\pi_2(u)$ **c** $Q'^{B'(\alpha(Q))}$ holds. But since by definition $c_0(x_1, \ldots, x_n)$ is exactly this $\alpha(Q)$, the desired property holds by assumption.

3.2.3 AL3

Rule	Premise 1	Premise 2	Conclusion	Term
2R	$B \equiv B$	$C \equiv B \Rightarrow A$	$A \equiv A$	$t(\beta,\gamma) \equiv \gamma(\beta)$
3R	$B \equiv B' \Rightarrow D$	$C \equiv D \Rightarrow C'$	$A \equiv B' \Rightarrow C'$	$t(\beta,\gamma) \equiv \lambda_x.\gamma(\beta(x))$
4R	$B \equiv B' \Rightarrow D$	$C \equiv C' \Rightarrow D$	$A \equiv B' \vee C' \Rightarrow D'$	$t(\beta,\gamma) \equiv$ $\lambda_x.[\pi_1(x) \mapsto \gamma(\pi_3(x)), 1 \mapsto \beta(\pi_2(x))]$
5R	$B \equiv D \Rightarrow B'$	$C \equiv D \Rightarrow C'$	$A \equiv D \Rightarrow B' \wedge C'$	$t(\beta,\gamma) \equiv \lambda_x.(\beta(x), \gamma(x))$
10R	$B \equiv A(0)$	$C \equiv A(x) \Rightarrow A(x')$	$A \equiv A(x)$	$t(\beta,\gamma) \equiv \texttt{ind}(0, x, \beta, \lambda_z.\gamma(\pi_2(z)))$

Table 4: AL3

Lastly for **AL3** we consider all remaining rules, which are the ones with exactly two premises. Thus, in addition to the term $u : T(B)$ we already used in **AL2**, we now analogously assume that in all cases for the listed formula C in Table 4 there is some term $v : T(C)$, such that v **c** R^C. Also analogously to β as in **AL2**, we additionally use γ as the parameter that will be replaced by v.

If now **AL3**$(P^A) = t(\beta, \gamma)$, by induction hypothesis both terms u and v exist, such that u **c** Q^B holds, and also v **c** R^C holds as well. Thus, the term $t(u, v)$ by assumption compiles P^A.

Now that we have seen **AL3** to function as desired, we will define it and prove t **c** P^A in turn for each two premise rule.

(**2R**): To show $t(u, v)$ **c** P^A, we directly use the definition of t, by which

$$t(u, v) \equiv v(u),$$

thus it needs to hold $v(u)$ **c** P^A. By assumption v **c** $R^{B\Rightarrow A}$ and with the definition of compilation, we thus know in combination with the other assumption u **c** Q^B that $v(u)$ **c** P'^A, which shows $t(u, v)$ **c** P^A, since P' and P must be the same.

(**3R**): To show $t(u,v)$ **c** P^A, it suffices to show that given $\bar{u}$ **c** $Q'^{B'}$ holds, it follows that $(t(u,v))(\bar{u})$ **c** $R'^{C'}$ also holds. By definition of t,

$$(t(u,v))(\bar{u}) \equiv v(u(\bar{u})).$$

By assumption u **c** $Q^{B' \Rightarrow D}$ and with the definition of compilation, we get that $u(\bar{u})$ **c** S^D holds. In turn, since also by assumption v **c** $R^{D \Rightarrow C'}$, we analogously know that $v(u(\bar{u}))$ **c** $R''^{C'}$ holds, which yields $(t(u,v))(\bar{u})$ **c** $R'^{C'}$, since R'' and R' must be the same proof.

(**4R**): To show $t(u,v)$ **c** P^A, it suffices to show that given s **c** $L^{B' \vee C'}$ holds, it follows that $(t(u,v))(s)$ **c** S^D also holds. By definition of t,

$$(t(u,v))(s) \equiv [\pi_1(s) \mapsto v(\pi_3(s)), 1 \mapsto u(\pi_2(s))].$$

We perform a case distinction on $\pi_1(s)$, where in the first case we consider it to be equal to 0 and in the second case to not be equal to 0. This case distinction is constructive since $s : T(B' \vee C') \equiv (\mathbb{Z}, T(B'), T(C'))$ and therefore $\pi_1(s) : \mathbb{Z}$ and equality for natural numbers is decidable, thus we do not use the rule of the excluded middle.

Now if $\pi_1(s) = 0$, by (10A) it follows that $[\pi_1(s) \mapsto v(\pi_3(s)), 1 \mapsto u(\pi_2(s))] = u(\pi_2(s))$. By the definition of compilation and using the assumption that $\pi_1(s) = 0$ we get that $\pi_2(s)$ **c** $R'^{B'}$ holds, and then using the assumption u **c** $R^{B' \Rightarrow D}$, we know again from the definition of compilation that $u(\pi_2(s))$ **c** S'^D holds, which yields $(t(u,v))(s)$ **c** S^D, since S' and S must be the same proof.

For the other case, we now assume $\pi_1(s) \neq 0$. This time by (10A) it follows that $[\pi_1(s) \mapsto v(\pi_3(s)), 1 \mapsto u(\pi_2(s))] = v(\pi_3(s))$. By the definition of compilation and using the assumption that $\pi_1(s) \neq 0$ we get that $\pi_3(s)$ **c** $Q'^{C'}$ holds, and then using the assumption v **c** $Q^{C' \Rightarrow D}$, we know again from the definition of compilation that $v(\pi_3(s))$ **c** S''^D holds, which yields $(t(u,v))(s)$ **c** S^D, since S'' and S must be the same proof.

(**5R**): To show $t(u,v)$ **c** P^A, it suffices to show that given w **c** S'^D holds, it follows that $(t(u,v))(w)$ **c** $P'^{B' \wedge C'}$ also holds. By definition of t,

$$(t(u,v))(w) \equiv (u(w), v(w)).$$

To now show $(u(w), v(w))$ **c** $P'^{B' \wedge C'}$, it suffices to show that $u(w)$ **c** $Q'^{B'}$ and that $v(w)$ **c** $R'^{C'}$ both hold. By assumption u **c** $Q^{D \Rightarrow B'}$ from which, using the definition of compilation, we get $u(w)$ **c** $Q''^{B'}$, which yields $u(w)$ **c** $Q'^{B'}$ since Q'' and Q' must be the same proof. Analogously by assumption v **c** $Q^{D \Rightarrow C'}$, from which, again using the definition of compilation, we get $v(w)$ **c** $R''^{C'}$, which yields $v(w)$ **c** $R'^{C'}$ since R'' and R' must be the same proof.

(**10R**): To show $t(u,v)$ **c** P^A, we use the fact that equality on natural numbers is decidable, case splitting on the second argument of `ind`, namely $x : \mathbb{Z}$, as to whether $x = 0$ or $x \neq 0$. By definition of t,

$$t(u,v)[x] \equiv \texttt{ind}(0, x, u, \lambda_z.v[\pi_1(z)](\pi_2(z))).$$

If now $x = 0$, by the computation rule of `ind`, since $0 = 0$, we get

$$\texttt{ind}(0, 0, u, \lambda_z.v(\pi_2(z))) \equiv u.$$

Since by assumption u **c** $B^{A(0)}$, the first case is proven.

In the other case, we fix some $n : \mathbb{Z}$ with $n \neq 0$ and $x = n$. Thus by the computation rule of `ind` we get

$$\begin{aligned}
&\texttt{ind}(0, n, u, \lambda_z.v[\pi_1(z)](\pi_2(z))) \\
&\quad\equiv \texttt{ind}(0', n, (\lambda z.v[\pi_1(z)](\pi_2(z)))(0, u), \lambda_z.v[\pi_1(z)](\pi_2(z))) \\
&\quad\equiv \texttt{ind}(0', n, v[0](u), \lambda_z.v[\pi_1(z)](\pi_2(z))) \\
&\quad\equiv \texttt{ind}(0'', n, v[0'](v[0](u)), \lambda_z.v[\pi_1(z)](\pi_2(z))) \\
&\quad\ldots \\
&\quad\equiv \texttt{ind}(n, n, v[n-1](v[n-2](\ldots v[0'](v[0](u))\ldots)), \lambda_z.v[\pi_1(z)](\pi_2(z))) \\
&\quad\equiv v[n-1](v[n-2](\ldots v[0'](v[0](u))\ldots))
\end{aligned}$$

where the fourth line of course doesn't happen if $n = 0'$ and $n-1$ and $n-2$ are shorthand notation for 0 with $'$ used $n-1$ or $n-2$ times respectively. Thus, what remains to be shown is $v[n](v[n-1](\ldots v[0'](v[0](u))\ldots))$ **c** $P'^{A(n)}$. Since by assumption for any x, $v[x]$ **c** $R[x]^{A(x)\Rightarrow A(x')}$ holds, and also by assumption u **c** $Q^{A(0)}$, we have $v[0](u)$ **c** ${Q_1}^{A(0')}$, $v[0'](v[0](u))$ **c** ${Q_2}^{A(0'')}$, ..., $v[n](v[n-1](\ldots v[0'](v[0](u))\ldots))$ **c** ${Q_n}^{A(n)}$, where Q_n and P must then be the same proof and thus the second case is also proven.

This concludes the definition of the algorithms. While most of this proof follows Bishop [4], notations have been updated significantly, details have been made more precise and crucially we have added a proof of $(7A)$, which Bishop just postulated to hold since $t_1 = t_2$, which we actually prove in Lemma 1, as well as fixed a minor mistake in the proof for $(10R)$, where Bishop used induction, which however is the property we are trying to prove sound in the first place and also does not make use of the definition and properties of the `ind` structure at all, while our proof by decidable case distinction does. □

At this point [4, pp. 24], Bishop also goes into more detail regarding the concept of sameness of proofs, which he heavily uses in the proof of the compilation theorem. He first acknowledges, that up until this point, proofs were viewed throughout as not formally rigid objects, which is why he now bases the sameness of proofs again on the structure of the formula the proof proves.

Definition 8 (Sameness of Proofs [4]). *Let P and P' both be proofs of the same formula A. We define the notion that P and P' are the same proof inductively on the structure of A:*

1. *If A has free variables $x_1, \ldots, x_n$, then P and P' are the same, if the corresponding proofs of $A(f_1, \ldots, f_n)$ are the same for all functionals $f_1, \ldots, f_n$ of requisite types. For all remaining cases, A is assumed to have no free variables.*

2. *If $A \equiv A_1 \wedge A_2$, then P and P' are the same, if the corresponding proofs of A_i are the same (for $i = 1$ and $i = 2$).*

3. *If $A \equiv A_1 \vee A_2$, then P and P' are the same, if the same A_i is being proved, and if the proofs of that A_i are the same.*

4. *If $A \equiv A_1 \to A_2$, then P and P' are the same, if for an arbitrary proof of A_1, the corresponding proofs of A_2 are the same.*

5. *If $A \equiv \forall_x A_1(x)$, then P and P' are the same, if the corresponding proofs for $A_1(x)$ are the same.*

6. *If $A \equiv \exists_x A_1(x)$, then P and P' are the same, if $\alpha(P) = \alpha(P')$ and if the corresponding proofs of $A_1(\alpha(P))$ are the same.*

3.2.4 Supporting Lemmas

Bishop again does not provide a case for the simple formulas. This time however this is most likely due to the informality of the proofs. If they were made to be more precise, defining a notion of sameness for proofs of simple formulas might be feasible beyond simply requiring that they both prove the formula, since in this less formal setting we do not always know *how* the simple formula was proven and if there even are different ways at all of proving them.

As we mentioned above, we still have to prove the main property for $(7A)$, which we shall do now:

Lemma 1. *If terms $t_1, t_2 : T_1$ are equal and $t : T(A(t_1))$, then t **c** $P^{A(t_1)}$ implies t **c** $P'^{A(t_2)}$.*

Proof. Assume t **c** $P^{A(t_1)}$. Note that due to the equality of t_1 and t_2 it also holds that $T(A(t_1)) \equiv T(A(t_2))$. We show t **c** $P'^{A(t_2)}$ by induction over the structure of $A(t_i)$.

1. In case $A(t_i)$ is a prime formula, meaning either $A(t_i) \equiv t_i = t_3$ or $A(t_i) \equiv t_i \neq t_3$, where $t_3 : T_3 \equiv \mathbb{Z}$. Since equality is symmetric by (12A), we disregard the symmetric cases without loss of generality. We now need to show that, given t **c** $P^{t_1[=]t_3}$, t **c** $P'^{t_2[=]t_3}$ holds, where $[=]$ signifies it could be either $=$ or $\neq$, since it doesn't matter for the argument. By definition any term of type $\mathbb{Z}$ is sufficient if the (in-)equality holds, by transitivity, i.e. (12A), it does and since $t : T(A(t_1)) \equiv \mathbb{Z}$ must hold in this case, we are done.

2. In case $A(t_i) \equiv A_1(t_i) \wedge A_2(t_i)$, we need to show that given t **c** $P^{A_1(t_1) \wedge A_2(t_1)}$, t **c** $P'^{A_1(t_2) \wedge A_2(t_2)}$ holds. Thus, by definition we need to show that $\pi_i(t)$ **c** $P_i'^{A_i(t_2)}$ holds, which we get from the induction hypothesis and the fact that by definition of compilation $\pi_i(t)$ **c** $P_i^{A_i(t_1)}$.

3. In case $A(t_i) \equiv A_1(t_i) \vee A_2(t_i)$, we need to show that given t **c** $P^{A_1(t_1) \vee A_2(t_1)}$, t **c** $P'^{A_1(t_2) \vee A_2(t_2)}$ holds. Thus, by definition we need to show that $\pi_2(t)$ **c** $P_1'^{A_1(t_2)}$ holds if $\pi_1(t) = 0$ and $\pi_3(t)$ **c** $P_2'^{A_2(t_2)}$ holds if $\pi_1(t) = 1$. In the first case by definition from t **c** $P^{A_1(t_1) \vee A_2(t_1)}$ we get $\pi_2(t)$ **c** $P_1^{A_1(t_1)}$ and are done using the induction hypothesis, and similarly in the second case we use the induction hypothesis with $\pi_3(t)$ **c** $P_2^{A_2(t_1)}$.

4. In case $A(t_i) \equiv A_1(t_i) \Rightarrow A_2(t_i)$, we need to show that given t **c** $P^{A_1(t_1) \Rightarrow A_2(t_1)}$, t **c** $P'^{A_1(t_2) \Rightarrow A_2(t_2)}$ holds. Thus, by definition we need to show that whenever $\bar{t}$ **c** ${P'_1}^{A_1(t_2)}$ holds, then $t(\bar{t})$ **c** ${P'_2}^{A_2(t_2)}$ also holds. By definition of compilation, from t **c** $P^{A_1(t_1) \Rightarrow A_2(t_1)}$ we obtain that whenever $\tilde{t}$ **c** ${P_1}^{A_1(t_1)}$ holds, $t(\tilde{t})$ **c** ${P_2}^{A_2(t_1)}$ must also hold. The induction hypothesis at first only yields $\bar{t}$ **c** ${P_1}^{A_1(t_1)} \Rightarrow \bar{t}$ **c** ${P_1}^{A_1(t_2)}$, however due to the symmetry of equality, from this we also get $\bar{t}$ **c** ${P_1}^{A_1(t_2)} \Rightarrow \bar{t}$ **c** ${P_1}^{A_1(t_1)}$, thus finishing the proof with the previous assumptions.

5. In case $A(t_i) \equiv \exists_x A_1(t_i, x)$, we need to show that given $\pi_1(t) = \alpha(P^{A(t_1)})$ and $\pi_2(t)$ **c** $P_1(P^{A(t_1)})^{A_1(P^{A(t_1)})}$, $\pi_1(t) = \alpha(P^{A(t_2)})$ and $\pi_2(t)$ **c** $P_1(P^{A(t_2)})^{A_1(P^{A(t_2)})}$. Since $t_1 = t_2$, the proofs $P^{A(t_1)}$ and $P^{A(t_2)}$ are the same, regarding the definition above, thus it also follows, that $\alpha(P^{A(t_1)}) = \alpha(P^{A(t_2)})$, from which by transitivity we of course get $\pi_1(t) = \alpha(P^{A(t_2)})$ as desired. Finally from the induction hypothesis and the sameness of the proofs just discussed, we then get that $\pi_2(t)$ **c** $P_1(P^{A(t_2)})^{A_1(P^{A(t_2)})}$ must hold, since $\pi_2(t)$ **c** $P_1(P^{A(t_1)})^{A_1(P^{A(t_1)})}$ holds.

6. In case $A(t_i) \equiv \forall_x A_1(t_i, x)$, we need to show that given t **c** $P^{\forall_x A_1(t_1,x)}$ holds, then t **c** $P'^{\forall_x A_1(t_2,x)}$ also holds. Thus, by definition of compilation, we need to show that $t(x)$ **c** $P'(x)^{A_1(t_2,x)}$ holds. This immediately follows from the induction hypothesis and the fact that $t(x)$ **c** $P(x)^{A_1(t_1,x)}$ holds.

□

Another property we have used throughout is the fact that compilation respects equality. We prove this by induction in the following:

Lemma 2 (g-Term congruence)**.** *If terms $t_1, t_2 : T$ are equal and t_1 **c** P^A, then also t_2 **c** P^A, where P is a proof of A*

Proof. We prove this by induction on the definition of t_1 **c** P^A.

1. Since t_1 **c** P^A, for each assignment of a functional of the requisite type to each of the free variables of A, t_{1a} **c** ${P_a}^{A_a}$. Since t_2 equals t_1, they have the same free occurrences of variables, thus t_{1a} also equals t_{2a} and from the induction hypothesis we get t_{2a} **c** ${P_a}^{A_a}$.

2. Since t_1 **c** P^A, $t_1 : \mathbb{Z}$ and P constructively proves A, by the equality of t_1 and t_2, it follows that $t_2 : \mathbb{Z}$ and thus t_2 **c** P^A.

3. Since t_1 **c** P^A, $\pi_i(t_1)$ **c** ${P_i}^{A_i}$ holds. Since t_1 and t_2 are equal, it must hold that t_2 **c** $P'^{A'_1 \wedge A'_2}$ where $A'_1 \equiv A_1$ and $A'_2 \equiv A_2$. Since P_i is a proof for A_i it must also be a proof A'_i, since they are equal. Thus from the induction hypothesis we get $\pi_i(t_2)$ **c** ${P_i}^{A_i}$ and finally t_2 **c** P^A.

4. Since t_1 **c** P^A, $\pi_{i+1}(t_1)$ **c** ${P_i}^{A_i}$, where $i = 1$ iff $\pi_1(t_1) = 0$ and $i = 2$ iff $\pi_1(t_1) \neq 0$, note that this is decidable by Axiom ($16A$) and LEM is not required. Since t_1 and t_2 are equal, $\pi_i(t_1) \equiv \pi_i(t_2)$ for $i \in \{1, 2, 3\}$, thus analogous to the previous case from equality of t_1 and t_2 we get $\pi_{i+1}(t_2)$ **c** ${P_i}^{A_i}$ and from the induction hypothesis we further get t_2 **c** P^A.

5. Since t_1 **c** P^A, whenever t_{11} **c** ${P_1}^{A_1}$, we also have $t_1(t_{11})$ **c** ${P_2}^{A_2}$. We want to show t_2 **c** P^A, meaning whenever we have t_{21} **c** ${P_1}^{A_1}$, we also have $t_2(t_{11})$ **c** ${P_2}^{A_2}$. Assume t_{21} **c** ${P_1}^{A_1}$. From the induction hypothesis we get that if t_{11} and t_{21} are equal, we have $t_1(t_{21})$ **c** ${P_2}^{A_2}$. Now since t_1 and t_2 are equal, with Axiom $(7A)$ it follows that $t_2(t_{21})$ **c** ${P_2}^{A_2}$ as desired.

6. Since t_1 **c** P^A, $\pi_1(t_1) = \alpha(P)$ and $\pi_2(t_1)$ **c** $A_1(\alpha(P))^{P_1(\alpha(P))}$. We now want to show that $\pi_1(t_2) = \alpha(P)$ and $\pi_2(t_2)$ **c** $A_1(\alpha(P))^{P_1(\alpha(P))}$. The first property follows immediately with Axiom $(7A)$ and the equality of t_1 and t_2. The second one we recover from using the same axiom with the induction hypothesis.

7. Since t_1 **c** P^A, $t_1(x)$ **c** $P_1(x)^{A_1(x)}$. We now need to show $t_2(x)$ **c** $P_1(x)^{A_1(x)}$. From the equality of t_1 and t_2 we again get the equality of $t_1(x)$ and $t_2(x)$, thus we are done by induction hypothesis.

□

With this result we give the details of Bishop's remarks, since he simply postulated that this was true and could be shown by induction.

After giving the compilation theorem, and the notes described above, Bishop then goes on to describe how one might implement this set of constant free terms he calls Σ_0 into ALGOL, specifically he notes that compilation into ALGOL68 would be "trivial" due to the type system (called modes in ALGOL68) being sufficiently expressive already, but at the time ALGOL60 was still more widespread and thus he devoted some pages to the workarounds necessary for the less expressive type system. Since modern functional programming languages like Haskell or even dependently typed ones like Lean have even richer type systems than ALGOL68, we will not further delve into this part of the manuscript, leaving it as possible future work to maybe implement the proposed compilation into any programming language, maybe even ALGOL60 following Bishop's instructions.

3.3 Compilation vs. Realizability

Realizability [13] as well as modified realizability [15], are concepts that are in principle very close to Bishop's notion of compilation in that they all aim to relate provability of a formula to some other language.

"Normal" realizability, or Kleene realizability, was first given and proven sound (along some other properties) by Kleene [13] and below we present his definition with slightly updated notation. Also note that of course this definition is not in the system Σ, not in a type theory context at all in fact, but in a system of (partial) recursive functions and formulas in the usual sense, but one could of course without much effort adapt the definition to conform to system Σ.

Definition 9 ((Kleene) Realizability [13]). *A natural number is said to realize a formula, or to be a realizer of the formula, denoted as n **r** A iff*

*1. If $A(y_1, \ldots, y_m)$ is a formula containing exactly the distinct free variables $y_1, \ldots, y_m$ in order of first free occurrence, and if e **r** $\forall_{y_1,\ldots,y_m} A(y_1, \ldots, y_m)$,* *then also e $\boldsymbol{r}$ $A(y_1, \dots, y_m)$. For all remaining cases we assume the formulas to not contain any free variables.*

2. *An elementary formula F is realized by 0 if it is true, i.e. 0 $\boldsymbol{r}$ F.*

3. *If a $\boldsymbol{r}$ A and b $\boldsymbol{r}$ B, then $2^a \cdot 3^b$ $\boldsymbol{r}$ $A \wedge B$.*

4. *If a $\boldsymbol{r}$ A, $2^0 \cdot 3^a$ $\boldsymbol{r}$ $A \vee B$, also if b $\boldsymbol{r}$ B, then $2^1 \cdot 3^b$ $\boldsymbol{r}$ $A \vee B$.*

5. *If e is the Gödel number of a partial recursive function ϕ, for which it holds that whenever a $\boldsymbol{r}$ A, also $\phi(a)$ $\boldsymbol{r}$ B, then e $\boldsymbol{r}$ $A \Rightarrow B$.*

6. *If a $\boldsymbol{r}$ $A(x)$, where x is the only free variable of A, then $2^x \cdot 3^a$ $\boldsymbol{r}$ $\exists_x A(x)$.*

7. *If e is the Gödel number of a general recursive function ϕ, for which it holds that for every x, $\phi(x)$ $\boldsymbol{r}$ $\forall_x A(x)$.*

A formula is said to be realizable iff some natural number realizes it.

For ease of use, later iterations of this definitions admitted any natural number n to realize an elementary formula iff it is true instead of only 0 doing so. In the spirit of this convention we also constructed the case for basic formulas in the sense of Bishop for his concept of compilation, which initially lacked a base case.

Later Kreisel [15] adapted this concept of Kleene, from using natural numbers as realizers, to instead use tuples of typed variables, the types depending on the logical structure of the specific formula. This concept called modified realizability also now uses a version of Heyting arithmetic, in the definition used by [14] specifically $E - HA^\omega$, as the language of formulas. To give this definition note that Kohlenbach defines the following notation:

$$\underline{\mathrm{y}}\underline{\mathrm{x}} := y_1\underline{\mathrm{x}}, \dots, y_n\underline{\mathrm{x}}$$

where $\underline{\mathrm{y}} = y_1, \dots, y_n$, $\underline{\mathrm{x}} = x_1, \dots, x_k$ are tuples of functionals of suitable types and $y_i\underline{\mathrm{x}} := y_i x_1 \dots x_n$, essentially mass-applying the elements of a tuple of functionals each to the same tuple of arguments.

Definition 10 (Modified Realizability [14][15])**.** *For each formula A of $\mathcal{L}(E-HA^\omega)$ we define a formula $\underline{x}$ $\boldsymbol{mr}$ A also of $\mathcal{L}(E - HA^\omega)$, to be read as '$\underline{x}$ modified realizes A', where $\underline{x}$ is a - possibly empty - tuple of variables which do not occur free in A. The definition is inductive over the logical structure of A as follows:*

1. *$\underline{x}$ $\boldsymbol{mr}$ $A := A$ with the empty tuple $\underline{x}$, if A is a prime formula.*

2. *$\underline{x}, \underline{y}$ $\boldsymbol{mr}$ $A \wedge B := \underline{x}$ $\boldsymbol{mr}$ $A \wedge \underline{y}$ $\boldsymbol{mr}$ B.*

3. *$z^0, \underline{x}, \underline{y}$ $\boldsymbol{mr}$ $(A \vee B) := [(z =_0 0 \Rightarrow \underline{x}$ $\boldsymbol{mr}$ $A) \wedge (z \neq_0 0 \Rightarrow \underline{y}$ $\boldsymbol{mr}$ $B)]$.*

4. *$\underline{y}$ $\boldsymbol{mr}$ $(A \Rightarrow B) := \forall_{\underline{x}}(\underline{x}$ $\boldsymbol{mr}$ $A \Rightarrow \underline{y}\underline{x}$ $\boldsymbol{mr}$ $B)$.*

5. *$\underline{x}$ $\boldsymbol{mr}$ $(\forall_{y^\rho} A(y)) := \forall_{y^\rho}(\underline{x}y$ $\boldsymbol{mr}$ $A(y))$.*

6. *$z^\rho, \underline{x}$ $\boldsymbol{mr}$ $(\exists_{y^\rho} A(y)) := \underline{x}$ $\boldsymbol{mr}$ $A(z)$.*

Instead of collecting potentially exponantiated, Gödel numbers and their products as one big realizer as Kleene does, Kreisel now collects a tuple of typed witnesses of the type theoretic truth of the formulas, which is already a lot closer to our modern understanding and use of type theory and the intuition we get from type theory.

It turns out that compilation as defined by Bishop, barring the missing base case for prime, or basic, formulas, is very much identical to modified realizability. The main difference is Bishop's focus on compiling the proof, instead of realizing a formula. In essence this very vague concept is however never really used in the definition, since all case analysis is done on the logical structure of the formulas. However despite this apparent irrelevance of the proofs at first glance, philosophically and conceptually it for one depends the relation to the Curry-Howard correspondence, by having the terms compile the crucial derivation part of a formula, instead of realizing the very abstract and formalistic notion of a formula. Starting from such an intuition which might give new interpretation to previously known results, fields have undergone major change, for example MLTT has been significantly expanded upon starting with the idea of viewing more formalistic concepts with a homotopic interpretation in mind, finally resulting in HoTT. Of course the intuition is not the sole factor in such work, therefore in the next section we would like to expand on work which is still left open.

3.4 Future Work

As a consequence of the inclusion of strong inequality in system Σ, it felt natural to think about constructing a proof system in which one may not only have derivations, possibly in the form of Gentzen-style derivation trees, but also constructive refutations of facts talking about constructive or strong negative properties. These concepts then exist in a sort of category theoretic duality, where a refutation may be thought of as a co-derivation, and thus having a co-derivation tree, which like all dual concepts, think co-products or co-limits for example, would then be inverted, following a top-down direction instead of the typical bottom-up one of Gentzen-style derivations. Further and of course more concrete work will be done by Petrakis [20].

Bishop makes note specifically of proofs in both formal and informal notions. Developing further on how possibly informal mathematical proofs might be reasoned about with a formal concept like compilation may prove insightful, to understand more about Bishop's motivations.

4 Bishop's Dependent Type Theory

In this section we will now take a brief look at Bishop's further ventures in regards to formal mathematics, namely [5], where he develops a dependent type theory, which in essence is an extension of the system Σ always with the modus of "typical", meaning informal, Bishop Set Theory in mind. This system however was very much left "in development" by Bishop, meaning it seems to not be completely finished.

First we will present the term system as well as the axioms and rules of this "General Language" of Bishop, which we will further call "Bishop Type Theory". After we briefly will talk about how BST can be actually modelled in Bishop Type Theory according to Bishop and then lastly discuss some specific curiosities of this system.

4.1 Terms and Types

In this section we will give the term system of Bishop Type Theory, where every term also has a type, specifically a possibly dependent type. The typeformers are mostly the same as for System Σ, but critically of course every type also has itself a type again. We have adapted the changes to notation made for System Σ also to Bishop Type Theory, specifically the modern type declarations using colons and more standard application notation were changed in the exact same way as described previously.

As System Σ, Bishop Type Theory is grounded in natural numbers $\mathcal{N}$, but now we also have a sort of "upper bound" for all types, $\mathcal{C}$, which represents the class of all classes, signifying the correspondence between classes in the BST-sense and types in Bishop Type Theory. The constants $0, \mathcal{N}$ and $\mathcal{C}$ are called basic constants, the only other constants are constants c that arise as the result of the proof of some formula $\exists_x A$.

$$\frac{}{0 : \mathcal{N} \qquad \mathcal{N} : \mathcal{C} \qquad \mathcal{C} : \mathcal{C} \qquad \text{terms}}\ (1t)$$

$$\frac{x : t \text{ variable}}{x : t \text{ term}}\ (2t)$$

$$\frac{t : \mathcal{N}}{t' : \mathcal{N}}\ (3t)$$

Due to the age of the paper, the nowadays well-known problematic self-referential $\mathcal{C} : \mathcal{C}$ makes this system by default prone to paradoxes [11], further work might constitute actually showing a specific paradox resulting from this for this specific system, but at least this specific problem can most likely be solved the same way it was for MLTT, by introducing an infinite hierarchy of $\mathcal{C}_i$'s but introducing this and dealing with possible consequences would also be future work.

New to the term structure in contrast to System Σ, is a representation of the power set, more accurately the power class but we will stick to the more common name, $\mathcal{P}$. In order to "switch" the type of a member of a power set to being a "normal" class, we have the construct $|t_1|$, we have the option to view members of this class, which by construction must be a subclass of the one that was used to form the power set, as members of the original class, we may use $t_2 \downarrow$ and in reverse, we can use $t_2 \uparrow t_1$ if we know that t_3 is in fact an element of t_1 viewed on its own. For $(4t, 4)$ note, that $\vdash$ means, that the formula is provable.

Note, that for all rules (nt, x) we mean this to be the x-th part of the n-th rule, thus type declarations from previous parts apply and names are defined across all parts of the same rule.

$$\frac{t : \mathcal{C}}{\mathcal{P}(t) : \mathcal{C}} \ (4t, 1)$$

$$\frac{t_1 : \mathcal{P}(t)}{|t_1| : \mathcal{C}} \ (4t, 2)$$

$$\frac{t_2 : |t_1|}{t_2 \downarrow : t} \ (4t, 3)$$

$$\frac{t_3 : t \qquad \vdash t_3 \in t_1}{t_3 \uparrow t_1 : |t_1|} \ (4t, 4)$$

Where System Σ natively had functions, implying that these objects respect equality, for Bishop Type Theory the primitive in this regard is an assignment operation, which does not by default respect equality. This notion of an assignment operation stems from BST, where it also constitutes the base of functions. Additionally we also have what Bishop called "guided operators", which we would now view as dependent functions, or more accurately in this setting dependent assignment routines. They need a sort of type family as a prerequisite, meaning an (non-dependent) assignment routine, which maps into $\mathcal{C}$. These also occur in informal BST, so they are natural to include and again don't necessarily respect equality.

In terms of notation, while Bishop used $\mathtt{Op}(t_1, t_2)$ for the assignment routines and $\mathtt{Op}(t : t_1)$ for the dependent version, we found this to be not very informative or readable, possibly stemming from this works incomplete state and thus propose the Notation seen below, which is more in line with the notation Bishop himself used in informal BST.

$$\frac{t_1, t_2 : \mathcal{C}}{t_1 \rightsquigarrow t_2 : \mathcal{C}} \ (5t)$$

$$\frac{t_1 : t \rightsquigarrow \mathcal{C}}{\curlywedge_{t} t_1 : \mathcal{C}} \ (6t)$$

As already discussed for System Σ for functions, assignment routines are λ-terms and can be applied to arguments of the correct type as usual. For dependent assignment routines this holds analogously with the main distinction, that the result type of the dependent application is of course dependent on the input. The notation $\mathbf{T}(t)$ hereby denotes the type of some term t.

$$\frac{t_3 : t_1 \rightsquigarrow t_2 \qquad t_4 : t_1}{t_3(t_4) : t_2} \ (7t)$$

$$\frac{t_1 : t \rightsquigarrow \mathcal{C} \qquad t_2 : t \qquad t_3 : \bigwedge_{t} t_1}{t_3(t_2) : t_1(t_2)}\ (8t)$$

$$\frac{\text{x is final in } \mathtt{dep}(\mathrm{x}, \mathrm{t}) \qquad x : t_1 \qquad t : t_2}{\lambda x.t : t_1 \rightsquigarrow t_2}\ (9t)$$

$$\frac{t_1 : t \rightsquigarrow \mathcal{C} \qquad x : t \qquad t_2 : t_1(x) \qquad \text{x is final in } \mathtt{dep}(\mathrm{x}, \mathrm{t}_2)}{\lambda x.t_2 : \bigwedge_{t} t_1}\ (10t)$$

At this point for completeness we will give a formal definition of the parents of terms and formulas, which we will need below. We will also signify where variables are bound, or as we'll discuss below in Bishop's words "dummied".

Definition 11 (Parents of term or formula [3])**.** *Every term or formula has a number of parents, defined through case distinction on the rules of their creations.*

***(1at)**: The basic constants have no parent. The term c has every formula occurring in the proof of $\exists_x A$ as a parent, including the formula $\exists_x A$ itself.*

***(2at)**: The (only) parent of a variable x is its type.*

***(3at)**: The parent of t' is t.*

***(4at)**: The parent of $\mathcal{P}(t)$ is t, of $|t_1|$ is t_1 and of $t_2\downarrow$ is t_2. The parents of $t_3 \uparrow t_1$ are t_1 and t_3.*

***(5at)**: The parents of $t_1 \rightsquigarrow t_2$ are t_1 and t_2.*

***(6at)**: The parents of $\bigwedge_{t} t_1$ are t and t_1.*

***(7at)**: The parents of $t_3(t_4)$ are t_3 and t_4.*

***(8at)**: The parents of $t_3(t_2)$ are t_2 and t_3*

***(9at)**: The parents of $\lambda_x.t$ are x and t, and x is bound.*

***(10at)**: The parents of $\lambda_x.t_2$ are x and t_2, and x is bound.*

***(11at)**: The parents of $[u|t_0, \ldots, t_n]$ are $u, t_0, \ldots, t_n$.*

***(12at)**: The parents of $t_1 \cup \cdots \cup t_n$ are $t_1, \ldots, t_n$. The parents of $in_i(\bar{t_i})$ are $\bar{t_i}$ and $t_1 \cup \cdots \cup t_n$. The parent of $t_0!$ and $\mathtt{out}_i(t_0)$ is t_0.*

***(13at)**: The parents of $\{x|A\}$ are x and A, and x is bound.*

***(14at)**: The parents of $\mathtt{G}(x_1, \ldots, x_n)$ are $x_1, \ldots, x_n$, and $x_1, \ldots, x_n$ are bound.*

***(15at)**: The parents of $\mathtt{ind}(t_1, t_2, t_3, t_4)$ are t_1, t_2, t_3 and t_4.*

***(1aF)**: The parents of $t_1 = t_2$ and $t_1 \neq t_2$ are t_1 and t_2.*

***(2aF)**: The parents of $t_1 \in t_3$ are t_1 and t_3.*

*(**3aF**): The parents of $A \wedge B, A \vee B$ and $A \Rightarrow B$ are A and B.*

*(**4aF**): The parents of $\forall_x A$ and $\exists_x A$ are x and A, and x is bound.*

This definition is taken directly from [3], of course with our new notation. Each rule (nat) or (naF) references the corresponding rule (nt) or (nF) below.

One concept that Bishop introduces in $(10t)$ is that of $\mathtt{dep}(x, t)$, as well as some variable being final in this. Bishop defines this in the following way:

Definition 12 (Dependency)**.** *Dependency is defined inductively as follows:
Every variable x depends upon itself.
If x is not bound for a term or formula and a parent of this depends on x, the term or formula depends on x.*

We denote the dependency of y on x as the relation $x \leq y$.

Let $t_1, \dots, t_m$ be terms and $A_1, \dots, A_n$ be formulas. Then $\mathtt{dep}(t_1, \dots, t_m, A_1, \dots, A_n)$ is defined as the set of all variables on which one of these terms or formulas depend. An element x of this set S is called final iff $y \in S \Rightarrow x \leq y \Rightarrow y \equiv x$. Similarly a Subset $S_1 \subseteq S$ is called a final segment of S iff $y \in S \Rightarrow x \in S_1 \Rightarrow x \leq y \Rightarrow y \in S_1$

Instead of "bound" Bishop uses the notion "dummied" in this definition, we found "bound" to be the more appropriate concept, since their definitions coincide and boundedness is an established well known concept already. Also with the definition we might now say that intuitively speaking x being final in $\mathtt{dep}(x, t)$ means, there are no other variables which depend on x in t, which are not themselves bound somewhere in t before. The only way by which this may be violated is, if some variable distinct from x has a type which then depends on x.
Let's see a few small examples for this definition of dependency and $\mathtt{dep}$:

Example 2. *Let $t \equiv \lambda x.y : T_1 \rightsquigarrow T_2$, where $y : T_2$ and $x : T_1$. Then the parents of t are for one x, which is also bound for this term, and y, which is not bound. Then $\mathtt{dep}(x, t) \equiv \{x, y\}$. For one by including x directly in $\mathtt{dep}$, and it always depends on itself, we will always have it in our dependency set but it would **not** be included through t, since as a parent of t it is bound. On the other hand y is a parent of t which is not bound for this term and thus since it depends on itself needs to be included. If T_1 and T_2 are more complex types, which also include variables, these might also show up in $\mathtt{dep}$, since for all variables their only parent is their type.
We can now see, that x is indeed final in $\mathtt{dep}(x, y)$, since y does not depend on it and for x it indeed holds that $x \equiv x$, thus t is a well formed assignment routine according to (9t).*

Example 3. *Let $t \equiv \lambda x.y : \mathcal{P}(\mathcal{N}) \rightsquigarrow x$, where $x : \mathcal{P}(\mathcal{N})$ and $y : x$. Again $\mathtt{dep}(x, y) \equiv \{x, y\}$, but this time for one x would also be in the dependence set without being explicitly listed as an argument and more critically x is **not** final in this, since y does depend on x here, since its parent is x itself and x depends on itself. Thus this is not a well formed assignment routine according to (9t). The main problem here is, that we have a more or less random free variable, which since it depends on x might prove to be trouble.*

$$\frac{u : \mathcal{N} \qquad t_0, \dots, t_n : t}{[u|t_0, \dots, t_n] : t} \ (11t)$$

This construct represents the u-th element of the n terms, where it always represents t_n if $u > n$.

$$\frac{t_1, \dots, t_n : \mathcal{C}}{t_1 \cup \dots \cup t_n : \mathcal{C}} \ (12t, 1)$$

$$\frac{\bar{t_i} : t_i}{\texttt{in}_i \bar{t_i} : t_1 \cup \dots \cup t_n} \ (12t, 2)$$

$$\frac{t_0 : t_1 \cup \dots \cup t_n}{t_0! : \mathcal{N} \qquad \texttt{out}_i(t_0) : t_i, 1 \leq i \leq n} \ (12t, 3)$$

The language also features all finite (disjoint) unions of classes with the typical notation. For the injection of an element of the i-th part of the union, we have adapted Bishop's original notation of $[t_i', i, t_1 \cup \dots \cup t_n]$ to a more modern one that takes inspiration from `inl` and `inr` used for the constructors of the sum type in MLTT, thus $\texttt{in}_i \bar{t_i}$ is an element of type $t_1 \cup \dots \cup t_n$ for arbitrary $0 \leq i \leq n$. Lastly the operator ! will return the index j, such that t_0 was originally injected from an element of type t_j. Thus $\texttt{out}_i(t_0)$ only represents the corresponding object, iff $i = t_0!$. It requires further work to see whether it might be feasible to simply replace i with $t_0!$, thus only allowing "correct" extractions, or if this somehow limits expressiveness. Note that Bishop originally used the notation $(t_0; i)$ instead of $\texttt{out}_i(t_0)$, this however is for one, as in the original notation for System Σ, ambiguous with the notation he used for application (which we also changed as noted in Section 2) and also not very informative.

$$\frac{x : t \qquad A \text{ is formula} \qquad \text{x is final in } \texttt{dep}(\text{x}, \text{A})}{\{x|A\} : \mathcal{P}(t)} \ (13t)$$

Here we introduce the of course ever important structure which enables defining a set through some property A. Again we need to restrict x to variables such that no free variable in A depends on it, since this structure additionally binds x.

$$\frac{x_1, \dots, x_n \text{ final segment of } \texttt{dep}(x_1, \dots, x_n) \qquad \forall_{i \leq j}\ x_i \text{ does not depend on } x_j}{\texttt{G}(x_1, \dots, x_n) : \mathcal{C}} \ (14t, 1)$$

$$\frac{x_1, \dots, x_n \Rightarrow t_1, \dots, t_n \text{ is a specialization}}{\langle t_1, \dots, t_n \rangle : \texttt{G}(x_1, \dots, x_n)} \ (14t, 2)$$

$$\frac{t : \texttt{G}(x_1, \dots, x_n)}{\forall_{1 \leq i \leq n} \pi_i(t) \quad \text{Types depend on specialization } x_1, \dots, x_n \Rightarrow \pi_1(t), \dots, \pi_n(t)} \ (14t, 3)$$

The last new concept is that of $G(x_1, \dots, x_n)$, which Bishop calls a layered product. This name most likely stems from the fact, that each variable, or "layer", may only depend on the layers before it, but not the ones after. Therefore this layered product can be thought of as the generalized version of a Σ-type from modern dependent type theories such as MLTT. We have only slightly simplified the notation for this, opting to exclude the additional "$: \texttt{G}(t_1, \dots, t_n)$" Bishop originally put after the list of terms inside the n-tuple. In order to fully understand these layered products as Bishop envisioned them, we need to define and then understand what a specialization is in the first place. The following is how Bishop defines these specializations and the accompanying concept of an image of a proof, term or formula.

Definition 13 (Specialization, Image [3])**.** *Let $z_1, \dots, z_k$ be a final sequence of variables that is a final segment of $\textbf{\textit{dep}}(z_1, \dots, z_k)$ where z_i does no depend on z_j for $i < j$. A specialization of a finite sequence of variables $z_1, \dots, z_k$ assigns to each z_i a term v_i denoted by $z_1, \dots, z_n \Rightarrow v_1, \dots, v_n$, such that the types of v_1 and z_1 are the same and for all other v_i it holds, that*

$$v_i : \textbf{\textit{im}}(\boldsymbol{T}(z_i) | z_1, \dots, z_{i-1} \Rightarrow v_1, \dots, v_{i-1})$$

for $2 \leq i \leq k$, where $\boldsymbol{T}(z_i)$ stands for the type of z_i and $\textbf{\textit{im}}(t | z_1, \dots, z_k \Rightarrow v_1, \dots v_k)$ stands for a certain term, the image of the term t with respect to the given specialization.

Let now $z_1, \dots, z_k \Rightarrow v_1, \dots v_k$ be a fixed specialization and let $\textbf{\textit{im}}(t)$ stand for the image of a term t with respect to that fixed specialization. For $\textbf{\textit{im}}(t)$ to be defined, it is required, that for each variable u belonging to $\textbf{\textit{dep}}(z_1, \dots, z_k, t)$, either $u = z_i$ for some i, or else $\textbf{\textit{im}}(\boldsymbol{T}(u))$ is defined and $\textbf{\textit{im}}(\boldsymbol{T}(u)) = \boldsymbol{T}(u)$. Analogously this holds for a formula A instead of t.
In case t is a basic constant, $\textbf{\textit{im}}(t) \equiv t$.
In case the constant c is declared to represent the element constructed by the proof P of the formula $\exists_x B$, then $\textbf{\textit{im}}(x)$ is a constant declared to represent the element constructed by the proof $\textbf{\textit{im}}(P)$ of the formula $\textbf{\textit{im}}(\exists_x B)$.
In case x is a variable, then $\textbf{\textit{im}}(x) \equiv v_i$ in case $x \equiv z_i$ and $\textbf{\textit{im}}(x) \equiv x$ otherwise.
For a term t that is neither a variable nor a constant or for a formula A, let $t_1, \dots, t_m, A_1, \dots, A_n$ be the parents of t (respectively A), and let $x_1, \dots, x_p$ be bound. We may assume, after reordering if necessary, that x_i does not depend on x_j for $i < j$. Let $\bar{x_1}, \dots, \bar{x_p}$ be newly declared variables, with $\bar{x_1} : \textbf{\textit{im}}(\boldsymbol{T}(x_1))$ and

$$\bar{x_i} : \textbf{\textit{im}}(\boldsymbol{T}(x_i) | \bar{z_1}, \dots, \bar{z_k}, x_1, \dots, x_{i-1} \Rightarrow \bar{v_1}, \dots, \bar{v_k}, \bar{x_1}, \dots, \bar{x_{i-1}})$$

for $2 \leq i \leq p$, where $\bar{z_i}$ stands for z_i and $\bar{v_i}$ for v_i, except that they are left out of the list id $x_j \leq z_i$, meaning z_i depends on x_j, for some j $(1 \leq j \leq p)$. Let $\bar{t_i}$ (respectively $\bar{A_i}$) be the image of t_i (respectively A_i) with respect to the specialization

$$z_1, \dots, z_k, x_1, \dots, x_p \Rightarrow v_1, \dots, v_k, \bar{x_1}, \dots, \bar{x_p}.$$

Then $\textbf{\textit{im}}(t)$ (respectively $\textbf{\textit{im}}(A)$) is obtained from t (respectively A) by replacing each of its parents t_i or A_i by its image $\bar{t_i}$ or $\bar{A_i}$.
Finally, if P is a proof of a theorem A, then proof $\textbf{\textit{im}}(P)$ of $\textbf{\textit{im}}(A)$ is obtained as follows. If A is an axiom, $\textbf{\textit{im}}(A)$ will have a canonical proof. If A is the conclusion of some rule of inference, whose premises are $A_1, \dots, A_n$, then there exist canonical specializations $\sigma_1, \dots, \sigma_n$, uniquely determined by the specialization $z_1, \dots, z_k \Rightarrow v_1, \dots, v_k$ and the number of the rule of inference, and a canonical deduction D of $\textbf{\textit{im}}(A)$ from $\textbf{\textit{im}}(A_1 | \sigma_1), \dots, \textbf{\textit{im}}(A_n | \sigma_n)$. The proof $\textbf{\textit{im}}(P)$ then consists in proving $\textbf{\textit{im}}(A_i | \sigma_i)$ by the proof $\textbf{\textit{im}}(P_i | \sigma_i)$ (where P_i is the proof of A_i) for $1 \leq i \leq n$, and then deducing $\textbf{\textit{im}}(A)$ by D.

While this definition might seem overwhelming at first, in essence it boils down to the following steps to calculate the types for the terms of a concrete specialization:

1. Take the first variable, its type is also the type of your first term. Then for all following terms that are left, repeat steps 2 to 6 until all terms are typed.
2. Identify the type of the corresponding variable, meaning the n-th variable if you want the type of the n-th term.
3. Identify all parents of that type.
4. For all these parents, continue recursively with enumerating their parents until you end up with only constants and variables.
5. Leave all basic constants the same, replace all variables that are part of the specialization with their corresponding specialization term and for existential constants input the constant the proof calculates, leave all other variables untouched.
6. Reassemble the type with these replaced parents in the same order you previously deconstructed it.

Step 4, 5 and 6 apply regardless of if the deconstructed object is a term or formula. Calculating the image of a proof simply means splitting the proof into its premises and canonical specializations. Notation wise, where we opted to use "|" as a separator within the im-statements, Bishop originally used ":", which of course would conflict with our decision to use modern typing declarations using ":".

$$\frac{x, t_1, t_2 : \mathcal{N} \qquad y, t_3 : t \qquad t_4 : \mathtt{G}(x, y) \rightsquigarrow t \qquad x \text{ and } y \text{ are independent}}{\mathtt{ind}(t_1, t_2, t_3, t_4) : t} \quad (15t)$$

Lastly we introduce a term that we already recognize from System Σ. The intended definition for ind then is as follows:

$$\mathtt{ind}(t_1, t_2, t_3, t_4) = \begin{cases} t_3 & \text{if } t_2 = t_1 \\ \mathtt{ind}(t_1', t_2, t_4(\langle t_1, t_3 \rangle), t_4) & \text{otherwise.} \end{cases}$$

We see that the definition we want to have is very similar to the one in System Σ, the notable difference being, that it is dependently typed with the new layered product instead of a "standard" pair.

$$\frac{t_1, t_2 : \mathbb{Z}}{t_1 = t_2, t_1 \neq t_2 \text{ formulas}} \quad (1F)$$

$$\frac{t_1 : t_2 \qquad t_3 : \mathcal{P}(t_2)}{t_1 \in t_3 \text{ formula}} \quad (2F)$$

$$\frac{A, B \text{ formulas}}{A \wedge B, A \vee B, A \Rightarrow B \text{ formulas}} \quad (3F)$$

$$\frac{A \text{ formula} \qquad x \text{ final in } \texttt{dep}(x, A)}{\exists_x A, \forall_x A \text{ formulas}} \ (4F)$$

Formulas in Bishop Type Theory are the same as in System Σ, with two small exceptions, for one the quantified formulas also require, that x be final in $\texttt{dep}(x, A)$ and there is an additional formula $t_1 \in t_3$ which of course will semantically be used to signify membership of a set/class.

4.2 Inference Rules

The rules and axioms are originally given in the same natural language based way as those of System Σ and we have again adapted this to a modern notation of inference rules.
Now, as was also the case for System Σ, Bishop begins with the inference rules (we will again as in Section 2 use this to mean both axioms and rules) constituting the constructive propositional calculus. For the most part, as seen in Fig. 8, these are also the same as the inference rules used in System Σ, but there are two key differences.

$$\frac{}{A \Rightarrow A} \ (1A)$$

$$\frac{B}{A \Rightarrow B} \ (1R)$$

$$\frac{A \qquad A \Rightarrow B \qquad \{x_1, \dots, x_n\} = (\texttt{dep}(A) \setminus \texttt{dep}(B))}{(\exists_{x_1,\dots,x_n} 0 = 0) \Rightarrow B} \ (2R)$$

$$\frac{A \Rightarrow B \qquad B \Rightarrow C \qquad \{x_1, \dots, x_n\} = (\texttt{dep}(B) \setminus \texttt{dep}(A, C))}{(\exists_{x_1,\dots,x_n} 0 = 0) \Rightarrow (A \Rightarrow C)} \ (3R)$$

$$\frac{}{A \wedge B \Rightarrow A \qquad B \wedge A \Rightarrow A \qquad A \Rightarrow A \vee B \qquad B \Rightarrow A \vee B} \ (2A)$$

$$\frac{A \Rightarrow C \qquad B \Rightarrow C}{A \vee B \Rightarrow C} \ (4R)$$

$$\frac{C \Rightarrow A \qquad C \Rightarrow B}{C \Rightarrow A \wedge B} \ (5R)$$

$$\frac{A \Rightarrow B \Rightarrow C}{A \wedge B \Rightarrow C} \ (6R)$$

$$\frac{A \wedge B \Rightarrow C}{A \Rightarrow B \Rightarrow C} \ (7R)$$

$$\frac{}{0 = 0' \Rightarrow A} \ (3A)$$

Figure 8: Axioms and rules of Bishop Type Theory, forming the propositional calculus

As is probably immediately apparent to anyone versed in systems of logic, two rules which are foundational to really any such system, namely $(2R)$, modus ponens, and $(3R)$, transitivity of implication, have been modified with a very specific pre-condition, which is completely new and to our knowledge has not appeared anywhere else in this form before or since. For some unknown reason, Bishop postulates, that in order for these two rules to be usable, first the existence of all "forgotten dependencies" must be shown. The formula $0 = 0$ itself is of course vacuously true, so the existential pre-conditions imposed here amount to a proof of the inhabitedness of the respective types of $x_1, \ldots, x_n$. This inhabitedness corresponds to truth of the formula represented by that type in regards to the Curry-Howard correspondence, but since Bishop does not give justification for or even in any way comments on these new restrictions, it is unclear what his intentions may have been.

In $(8R)$ and $(9R)$, A does not depend on x. $A(t)$ is short for $\mathtt{im}(A|x \Rightarrow t)$:

$$\frac{(\exists_x 0 = 0) \Rightarrow A \Rightarrow B \qquad \text{x is final in } \mathtt{dep}(x, B)}{A \Rightarrow \forall_x B} \; (8R)$$

$$\frac{(\exists_x 0 = 0) \Rightarrow B \Rightarrow A \qquad \text{x is final in } \mathtt{dep}(x, B)}{(\exists_x B) \Rightarrow A} \; (9R)$$

$$\frac{}{K \Rightarrow (\forall_x A(x)) \Rightarrow A(t) \qquad \text{x is final in } \mathtt{dep}(x, A)} \; (4A)$$

$$\frac{}{K \Rightarrow A(t) \Rightarrow \exists_x A(x) \qquad \text{x is final in } \mathtt{dep}(x, A)} \; (5A)$$

where $K \equiv 0 = 0$ if $A(x)$ actually depends on x, and $K \equiv \exists_x(0 = 0)$ otherwise. Again analogously to System Σ, these four inference rules finish the definition of the constructive predicate calculus. However again, Bishop chose to include the existential pre-conditions albeit this time for the two axioms only if A actually depends on x. So for some reason, Bishop was concerned with possibly allowing quantification over an empty type. Again, he does not make further notice of anything regarding this.

Axiom $(6A)$ is $(\mathtt{AC})$, the type theoretic axiom of choice, which we have already discussed in great detail in Section 2. For this, y may not depend on x and $z : \mathbf{T}(x) \Rightarrow \mathbf{T}(y)$. It further also holds in case there is some t, such that $t : \mathbf{T}(x) \Rightarrow \mathcal{C}$ not depending on x, $y : t(x)$ and $z : \underset{\mathbf{T}(x)}{\curlywedge} t$, meaning in a dependent setting:

$$\frac{}{\forall_x \exists_y A(x, y) \Rightarrow \exists_z \forall_x A(x, z(x))} \; (\mathtt{AC})$$

Both $(10R)$ and $(11R)$ are exactly the same as for System Σ:

$$\frac{A(0) \qquad A(x) \Rightarrow A(x')}{A(x)} \; (10R)$$

$$\frac{\exists_x A(x)}{A(c) \qquad c \text{ constant from proof of existence}} \; (11R)$$

The next rule now encapsulates the previously mentioned interpretation of the new term $\{x|A(x)\}$:

$$\frac{}{t_1 \in \{x|A(x)\} \Leftrightarrow A(t_1)}\ (7A)$$

Here $B \Leftrightarrow C$ as usual is simply shorthand notation for $(B \Rightarrow C) \wedge (C \Rightarrow B)$.

For $(8-10A)$ all terms are of type $\mathcal{N}$, note that these rules are only a subset of the axioms for $\mathbb{Z}$ used in System Σ.

$$\frac{}{t_1 = t_2 \vee t_1 \neq t_2}\ (8A)$$

$$\frac{}{t_1 = t_2 \wedge (t_1 \neq t_2) \Rightarrow 0 = 0'}\ (9A)$$

$$\frac{}{t_1 = t_2 \Rightarrow t_1' = t_2' \qquad t \neq t'}\ (10A)$$

Bishop at this point defines a notation that two classes are actually "equivalent" in some sense:

Definition 14. *Let $t : \mathcal{C}, t_1, t_2 : t, x : \mathcal{P}(t)$. Then $t_1 \approx t_2$ is defined by:*

$$\forall_x (t_1 \in x \Leftrightarrow t_2 \in x)$$

$(11-12A)$ are in context of $(4t)$, specifically t_1, t_2 and t_3, and again now define the previously described meaning:

$$\frac{}{t_2 \downarrow\in t_1}\ (11A)$$

$$\frac{}{t_2 \downarrow\uparrow t_1 \approx t_2 \qquad (t_3 \uparrow t_1) \downarrow\approx t_3}\ (12A)$$

The inference rule $(13A)$ is in context of $(9t)$, with x and $\bar{t}$ having the same type, and now defines application of assignment operations:

$$\frac{}{(\lambda x.t(x))(\bar{t}) \approx t(\bar{t})}\ (13A)$$

Next, $(14A)$ is the dependent version of $(13A)$ and in context of (10t), again x and $\bar{t}$ have the same type.

$$\frac{}{(\lambda x.t_2(x))(\bar{t}) \approx t_2(\bar{t})}\ (14A)$$

The inference rule $(15A)$ is in context of $(11t)$, defining the semantics of the choice object we defined there. We use the notation 0^i as shorthand for $0^{'\cdots'}$ with i primes:

$$\frac{}{u = 0^i \Rightarrow [u|t_0, \ldots, t_n] \approx t_j \qquad j \equiv \mathtt{min}(i, n)}\ (15A, 1)$$

$$\frac{}{u \neq 0^0 \wedge \cdots \wedge u \neq 0^n \Rightarrow [u|t_0, \ldots, t_n] \approx t_n}\ (15A, 2)$$

Now, $(16A)$ is in context of $(12t)$ and gives the semantics of the operators defined there as previously described:

$$\frac{}{\texttt{out}_i(\bar{t_i})! = 0^i \qquad \texttt{in}_i(\texttt{out}_i(\bar{t_i})) \approx \bar{t_i} \qquad \bar{t}! = 0^i \Rightarrow \texttt{out}_i(\texttt{in}_i(\bar{t})) \approx \bar{t}}\ (16A)$$

The rules of inference $(17 - 18A)$ are in context of $(14t)$ where there is a formula $A(x_1, \ldots, x_n)$ such that $A(t_1, \ldots, t_n)$ and $A(\bar{t_1}, \ldots, \bar{t_n})$ are defined. They give the semantics of the layered product, which we discussed in length above:

$$\frac{}{\langle \pi_1(t), \ldots, \pi_n(t) \rangle \approx t}\ (17A)$$

$$\frac{}{\langle t_1, \ldots, t_n \rangle \approx \langle \bar{t_1}, \ldots, \bar{t_n} \rangle \Rightarrow (A(t_1, \ldots, t_n) \Leftrightarrow A(\bar{t_1}, \ldots, \bar{t_n}))}\ (18A)$$

For $(19A)$ $t_1, t_2 : \mathcal{N}$, giving an intuitively desired property of the above defined equivalence operator:

$$\frac{}{t_1 = t_2 \Rightarrow t_1 \approx t_2}\ (19A)$$

Finally $(20A)$ is in context of $(16t)$, defining the semantics of `ind` as given above:

$$\frac{}{\begin{array}{c} t_1 = t_2 \Rightarrow \texttt{ind}(t_1, t_2, t_3, t_4) \approx t_3 \\ t_2 = t_1^i \Rightarrow (\texttt{ind}(t_1, t_2, t_3, t_4) \approx \texttt{ind}(t_1', t_2, t_4(\langle t_1, t_3 \rangle), t_4)) \end{array}}\ (20A)$$

This concludes the definition of Bishop Type Theory as laid out in [3]. At the end of the rules of inference, Bishop writes, that "It should be stressed that certain seemingly artificial restrictions of the language are motivated by the desire not to press the search for constructive meaning too far. This does not mean that we are primarily concerned with avoiding contradictions. A contradiction would be just an indication that we were indulging in meaningless formalism" [3, p. 16]. This might be directed towards the peculiar pre-conditions we discussed above, but still does not really give any further insight as to their meaning.

In the rest of the manuscript, Bishop then works on formalizing some mathematical concepts, like Borell sets in this language of his. Discussing this would however go beyond the scope of this thesis, and requires further research, especially with regard to the oddities of the existential pre-conditions and their effect on the expressiveness of Bishop Type Theory as a whole. Thus it is left as future work for now. Additional future work involves checking if the paradoxes involving $\mathcal{C} : \mathcal{C}$ actually apply to this setting, as in the case of Girard's Paradox [11].

5 Conclusion

In this thesis we have explored the previously unpublished works of Bishop in the realm of formal foundations for mathematics, beginning with the concept of compilation, for which we first explored system Σ and its possible relation to what Bishop viewed as the canonical form for all mathematical statements and have then seen how Bishop ended up showing this system to be compilable into a constant free subset of terms and eventually into a programming language like ALGOL68. To this end we have given Bishop's proof, for which we have made certain cases more precise and fixed one case. We have also provided a better visual representation of the compilation algorithm by using tables to show the correspondence between rules and compilation terms. This concept of compilation developed by Bishop we have further compared to (modified) realizability and found that while the definitions are very similar, core concepts regarding proofs might prove valuable for gaining a better understanding and further developing proof systems in a constructive framework. Finally we have discussed the early stages of a dependent type theory based on Bishop Set Theory, which would have preceded that of Martin-Löf and features interesting peculiarities which might warrant further examination. Throughout we have always adapted the notation used to modern times and introduced new notation where deemed worthwhile to aid readability.

References

[1] K. Appel and W. Haken. ‘Every planar map is four colorable. Part I: Discharging’. In: *Illinois Journal of Mathematics* 21.3 (1977), pp. 429 –490. DOI: 10.1215/ijm/1256049011.

[2] E. Bishop. *Foundations of Constructive Analysis*. McGraw-Hill, 1967.

[3] E. Bishop. ‘A General Language’. unpublished manuscript. 1969?

[4] E. Bishop. ‘How to Compile Mathematics into Algol’. unpublished manuscript. 1969?

[5] E. Bishop. ‘Mathematics as a Numerical Language’. In: *Intuitionism and Proof Theory: Proceedings of the Summer Conference at Buffalo N.Y. 1968*. Ed. by A. Kino, J. Myhill and R.E. Vesley. Vol. 60. Studies in Logic and the Foundations of Mathematics. Elsevier, 1970, pp. 53–71. DOI: 10.1016/S0049-237X(08)70740-7.

[6] N. G. de Bruijn. ‘AUTOMATH, a Language for Mathematics’. In: *Automation of Reasoning: 2: Classical Papers on Computational Logic 1967–1970*. Ed. by Jörg H. Siekmann and Graham Wrightson. Berlin, Heidelberg: Springer Berlin Heidelberg, 1983, pp. 159–200. ISBN: 978-3-642-81955-1. DOI: 10.1007/978-3-642-81955-1_11.

[7] *Coq Website*. https://coq.inria.fr/. Accessed: 13.01.2024.

[8] N. G. de Bruijn. ‘Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem’. In: *Indagationes Mathematicae (Proceedings)* 75.5 (1972), pp. 381–392. ISSN: 1385-7258. DOI: 10.1016/1385-7258(72)90034-0.

[9] D. Bridges E. Bishop. *Constructive Analysis*. Vol. 279. Grundlehren der mathematischen Wissenschaften. Springer Berlin, Heidelberg, 1985. DOI: 10.1007/978-3-642-61667-9.

[10] *E-Prover Website*. http://wwwlehre.dhbw-stuttgart.de/~sschulz/E/E.html. Accessed: 13.01.2024.

[11] J.Y. Girard. ‘Interprétation fonctionnelle et élimination des coupures de l’arithmétique d’ordre supérieur’. Thèse d’État. Université Paris VII, 1972.

[12] M. Hofmann and T. Streicher. ‘The groupoid interpretation of type theory’. In: *Twenty Five Years of Constructive Type Theory*. Oxford University Press, Oct. 1998. ISBN: 9780198501275. DOI: 10.1093/oso/9780198501275.003.0008.

[13] S. C. Kleene. ‘On the interpretation of intuitionistic number theory’. In: *Journal of Symbolic Logic* 10.4 (1945), 109–124. DOI: 10.2307/2269016.

[14] U. Kohlenbach. *Applied Proof Theory: Proof Interpretations and their Use in Mathematics*. Springer Berlin, Heidelberg, 2008. DOI: 10.1007/978-3-540-77533-1.

[15] G. Kreisel. ‘On weak completeness of intuitionistic predicate logic’. In: *Journal of Symbolic Logic* 27.2 (1962), 139–158. DOI: 10.2307/2964110.

[16] *Lean Website*. https://lean-lang.org/. Accessed: 13.01.2024.

[17] P. Martin-Löf. 'An intuitionistic theory of types'. In: *Twenty Five Years of Constructive Type Theory*. Oxford University Press, Oct. 1998. ISBN: 9780198501275. DOI: 10.1093/oso/9780198501275.003.0010.

[18] I. Niven. *Irrational Numbers*. Ed. by T. Rado. Vol. 11. Carus Mathematical Monographs. The Mathematical Association Of America, 1985. DOI: 10.5948/UPO9781614440260.

[19] I. Petrakis. *Families of Sets in Bishop Set Theory*. Habilitationsschrift. Ludwigs-Maximilians-Universität München, 2020.

[20] I. Petrakis. 'A strong constructive logic'. in preparation. 2024.

[21] C. Spector. 'Provably recursive functionals of analysis: a consistency proof of analysis by an extension of principles formulated in current intuitionistic mathematics'. In: *Proceedings of Symposia in Pure Mathematics*. Ed. by J. C. E. Dekker. Vol. 5. 1962, pp. 1 –27. DOI: 10.1090/pspum/005/0154801.

[22] *The Agda Wiki*. https://wiki.portal.chalmers.se/agda/pmwiki.php. Accessed: 13.01.2024.

[23] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study: https://homotopytypetheory.org/book, 2013.

[24] *Vampire Website*. https://vprover.github.io/. Accessed: 13.01.2024.

[25] V. Voevodsky. 'Univalent Foundations Project'. In: *a modified version of an NSF grant application* (2010), 1–12. URL: http://www.math.ias.edu/vladimir/files/univalent_foundations_project.pdf.

[26] *Zipperposition Website*. http://sneeuwballen.github.io/zipperposition/. Accessed: 13.01.2024.