Studies on computability and complexity: A new structure for the lambda calculus and defining useful information

Marlou M. Gijzen

Japan Advanced Institute for Science and Technology

Doctoral Dissertation

Studies on computability and complexity: A new structure for the lambda calculus and defining useful information

Marlou M. Gijzen

Supervisor: Prof. Hajime Ishihara

Graduate School of Advanced Science and Technology
Japan Advanced Institute of Science and Technology
Doctor of Science (Information Science)
March 2023

Abstract

Core topics in computability and complexity were revisited, in order to generate new insights. In computability theory, we considered structures of the lambda calculus. We wanted to obtain a better understanding of the properties of structures of the lambda calculus, by studying structures more general than the already established ones.

For this purpose, we introduced a new notion for combinatory algebras, called reflexivity. Reflexivity can be characterized as the algebraic counterpart of the Meyer-Scott axiom for combinatory and lambda models. With reflexivity, we defined strongly reflexive combinatory algebras. Strongly reflexive combinatory algebras can interpret the lambda calculus, but are more general than lambda algebras.

Strongly reflexive combinatory algebras relate to the known structures in the following way: they are exactly the retracts of combinatory models, and a strongly reflexive combinatory algebra that satisfies stability is a lambda algebra.

In algorithmic complexity, we gave an overview of results related to logical depth. Logical depth is a notion that uses Kolmogorov complexity to capture the amount of useful information in strings. The definition of logical depth uses a significance level. We proved that similarly to sophistication, a related definition, logical depth is unstable with respect to the significance level: the value of logical depth changes a lot with only small changes to the significance level. For sophistication it was suggested to interpret the notion as a function in the significance level, because of this instability.

We showed that interpreting logical depth as a function in the significance level entails that it is not always possible to straightforwardly compare the logical depth of two different strings. This makes the usability of the current definition of logical depth uncertain. We hereby argued that it is necessary to reconsider the basis for the definition, and clarify the assumptions and requirements behind it.

Keywords: Lambda calculus, combinatory algebra, reflexivity, Kolmogorov complexity, logical depth

Acknowledgements

The help of numerous people was crucial for the process of creating this dissertation. First and foremost, I would like to thank professor Hajime Ishihara for welcoming me into his laboratory and for his supervision. Because of his continuous guidance and input, the research was able to take a turn for the better. I also want to thank professor Tatsuji Kawai for all his help, and for the work we did together.

I am also extremely grateful towards professor Akitoshi Kawamura. He has taught me a lot, and he was always patient with me. I have greatly improved my writing thanks to him.

Professor Nao Hirokawa, professor Takako Nemoto and other professors at JAIST have always provided me with useful discussions and therefore also receive my gratitude. I also thank the students of the Ishihara laboratory for making my time at JAIST so enjoyable.

This endeavour would not have been possible without the Ministry of Education, Culture, Sports, Science and Technology. They provided me with the funds to do my research and studies.

I also want to thank Mr. and Mrs. Mizuno of the Café Hickory in Kanazawa, for welcoming me during lonely times, and for providing me with snacks while I was working.

Lastly, I want to thank my friends and family, especially Akihiro, who supported me through the hardest times. Doing a PhD on the other side of the world is not easy in times of a global pandemic. Without them, I would not have been able to finish.

Contents

Chapter 1

Introduction

Computability and complexity are underlying to major studies in theoretical computer science. The field computability theory involves topics related to the notion of a computable function. There are several alternatives to describe computable functions, such as Turing machines, the lambda calculus, combinatory logic, Post systems and μ-recursive functions. All these turned out to be provably equivalent. Algorithmic complexity, or Kolmogorov complexity, uses Turing machines as the model for computation. The field encompasses the study of the complexity of objects by considering the amount of information in them. Strings which contain the most information are called random.

In this dissertation, we revisit core notions in computability and complexity, that are already accepted or taken for granted. This is done in order to gain new insights and a deeper understanding of fundamental topics.

1.1 Computability

1.1.1 Background

In the first part of the dissertation, we focus on a topic related to the correspondence between combinatory logic and the lambda calculus. The combinatory algebra is a structure for the system of combinatory logic. For computations in combinatory logic only two specific functions, called *combinators*, are used. These combinators are often referred to as `k` and `s`. With `k` and `s`, it is possible to define a lambda abstraction, as in the system of the lambda calculus, for combinatory algebras. This lambda abstraction is denoted with λ^*. However, this abstraction does not respect the β-equality of the lambda calculus.

1.1.2 Previous situation

There are several known structures for the lambda calculus, based on combinatory algebras, for which the defined lambda abstraction does respect the β-equality. In the following, we will outline those.

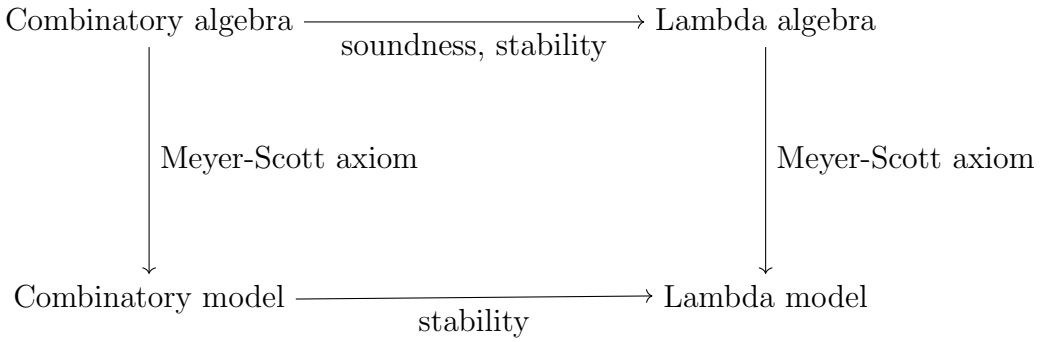
There is the *lambda algebra*, which is a combinatory algebra A such that two conditions hold with respect to the lambda abstraction λ^*: *soundness* and *stability*.

When we add a specific axiom called the *Meyer-Scott axiom* to a lambda algebra, we obtain a *lambda model*, first defined by Scott [37]. Satisfying the Meyer-Scott axiom is equivalent to being weakly extensional with respect to the lambda abstraction λ^*.

Another structure is the *combinatory model*: it can be defined as a combinatory algebra satisfying the Meyer-Scott axiom.

Combinatory models satisfy the soundness condition as mentioned above for lambda algebras. However, in general, combinatory models do not satisfy stability. Lambda models can be characterized as combinatory models which satisfy stability.

We have thus identified three structures for the lambda calculus: lambda algebras, lambda models and combinatory models. Note that the combinatory algebra cannot interpret the lambda calculus in general.



In recent years, not much research on this topic has been done. The lambda model and the combinatory model were defined in the beginning of the 1980s. In 2002, Selinger showed that in a lambda algebra, the lambda abstraction is interpreted as a polynomial. Because of the Meyer-Scott axiom, in combinatory models and in lambda models, the abstraction is interpreted as a function.

1.1.3 Motivation and significance

We revisit the topic of structures for the lambda calculus, with the goal of finding a more general structure than the three mentioned before. By doing this, we hope to obtain a better understanding of the properties that are absolutely necessary for interpreting the lambda calculus.

The observation that underlies this search for a more basic structure is the following. We already have three known structures that can interpret the lambda calculus, as shown in the diagram. What is missing from these, however, is a structure that does not satisfy stability, like combinatory models, but interprets lambda abstractions as a polynomial, like lambda algebras.

In order to reach the goal of gaining a better understanding of the requirements needed for such a structure, we searched in a bottom-up manner. We started with a combinatory algebra, and gradually added properties that seemed to be necessary for interpreting the lambda calculus. The most important property that was introduced, is *reflexivity*: an algebraic analogue of the Meyer-Scott axiom. We found this to be a fundamental property for interpreting the lambda calculus, however, it was not sufficient. At the end, we indeed found a structure that is the algebraic counterpart

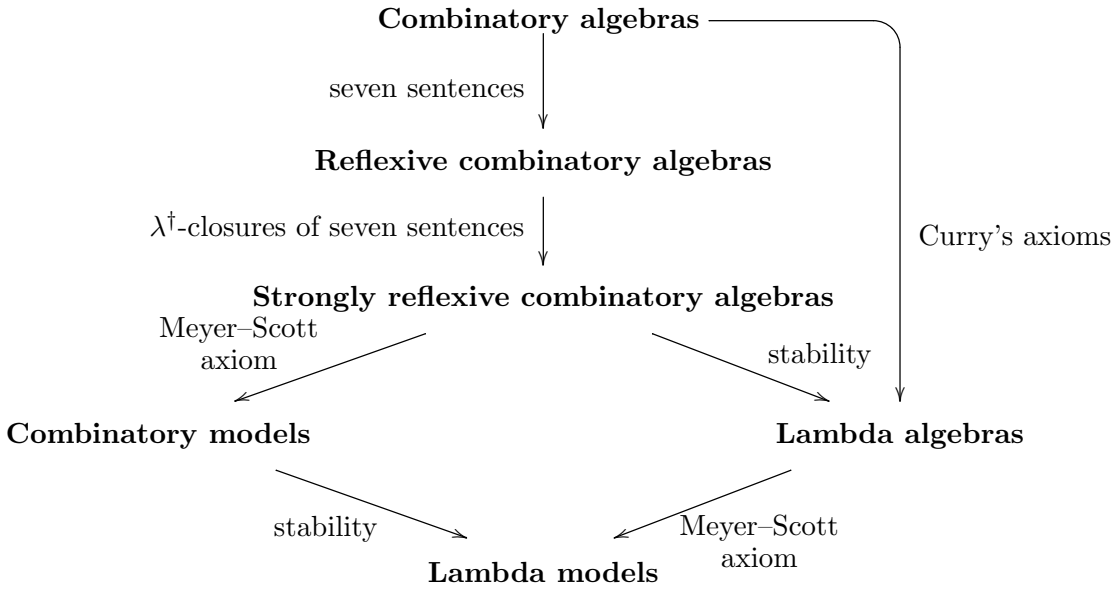


Figure 1.1: Relationship between various structures

of a combinatory model, where the abstractions are interpreted as polynomials, without satisfying stability. This structure is called a *strongly reflexive combinatory algebra*. We also formulated a finite axiomatization for these structures. Figure 1.1 gives the relationships between the structures.

Axiomatizations for lambda algebras were given by Curry [16] and Selinger [38]. However, these were found using a top-down manner, and it is not easy to explain why those axioms are necessary for interpreting the lambda calculus. Because of working in a bottom-up manner, the origins of the axioms for the strongly reflexive combinatory algebra are clear, and it can thus be explained why these are necessary for interpreting the lambda calculus. Moreover, these axioms are similar to those introduced by Selinger, and can hence be used to explain his axiomatization as well.

1.2 Complexity

1.2.1 Background

The notion of *Kolmogorov complexity* was independently introduced by Chaitin, Solomonoff and Kolmogorov. We say that the Kolmogorov complexity of a binary string is the length of the shortest program that computes the string. It has been used as a definition for the amount of information of binary strings. We can also use it to define randomness: random strings are those, for which the Kolmogorov complexity, and thus the shortest program that computes it, has roughly the same length as the string itself. A random string is incompressible: there is no shorter way to describe it than just giving the string itself. This is because random strings are patternless.

For any string x, there is always a program that computes it. A canonical program would have the instructions "print x". The length of such a program is the length of x plus a constant. It is therefore that the Kolmogorov complexity of a string is never larger than the length of the string itself plus a constant. Random strings thus have the largest Kolmogorov complexity, relative to the length of the string. This means that random strings contain the most information.

This seems counter-intuitive, since random strings do not seem like they have lots of information. That is why people have started wondering, whether there is also a way to capture the amount of *useful* information in strings. In order to understand the intuition behind the several approaches to defining useful information, we will give an example.

Consider the following three sounds: a constant tone at a specific frequency, random noise, and a recording of Hiromi Uehara playing a piece on the piano. All three sounds can be described with a binary string. The string that describes the constant tone has a really low Kolmogorov complexity (low information content). It can be compressed a lot, since it is only one simple pattern. The string that describes the noise has a really high Kolmogorov complexity; it cannot be compressed at all. The string that describes a recording of the piece on the piano can be compressed a bit, since the music will contain patterns. The compression is however not so simple as the one for the constant single pitch.

Objects that contain patterns, things that seem to have some form of organization, appear all around us. Music, but also this dissertation, our DNA, etc. And exactly these objects are those that intuitively contain more useful information than random strings.

1.2.2 Previous situation

There are several different definitions around, that were defined with the aim to capture this notion of useful information. Here we outline three of these, that all take a different approach.

The first definition is called *effective measure complexity* and it is the least well-known definition of the three. It is defined by Chaitin in [12] and it uses the amount mutual information between substrings in order to capture the amount of useful information.

The definition of effective measure complexity takes a string and a natural number s, and returns a natural number. How much useful information is contained in the string is then described by the function in s of the effective measure complexity for a fixed string.

A more well-known definition is the *sophistication* of Koppel and Atlan [24, 25, 3]. The definition uses two-part descriptions: the assumption is that a program for a string consists of a part that specifies the noise and a part that specifices the structure. It is defined for a string x and a natural number s, the significance level. This significance level was introduced because we cannot be certain about which two-part description is the "right" one for the string. The bigger the significance level, the smaller the chance that the value of sophistication that we have is the "true" value that represents the amount of structure in the string.

Later it was shown that sophistication is unstable with respect to the significance level: the value of sophistication changes a lot for only small changes in the significance level [2]. From this it was concluded that sophistication should be interpreted as a function in the significance level instead, similarly as for the effective measure complexity.

The final definition that we discuss is *logical depth* of Bennett [9]. It uses computation time as a measure of the amount of useful information. Logical depth is defined for a string and again a natural number s, the significance level. This significance level was introduced for similar reasons as for sophistication. Other than that it has to be small, it is not made precise which significance level to pick for the physical complexity of a string. In a somewhat recent publication, evidence was given that probably logical depth is also unstable with respect to the significance level [6].

1.2.3 Motivation and significance

In this work, we focus on the most-used definition logical depth. There are quite some results that use this definition, however, they are scattered and sometimes incorrect. We give an overview of the most important results, also relating several variants of the definition.

We also confirm that logical depth is indeed unstable with respect to the significance level. This could mean that logical depth should also be interpreted as a function in the significance level, as for sophistication. We also explore a problem that arises when considering logical depth as a function. Namely, we look into how we should compare the logical depth of two different strings. Even without considering functions, this was already a bit unclear because of the significance level. Suppose we are comparing the logical depth of two strings, while interpreting logical depth as a function in the significance level. The most straightforward way to decide which string has more useful information, would be to say that the string that has a higher logical depth than the other for each significance level, has higher logical depth (and thus more useful information).

We show that such straightforward decisions are not always possible. Namely, we show that for every length, there are two different strings x and y of that length, and two significance levels s_1 and s_2, such that for s_1, the string x has a higher logical depth than y, and for s_2 the other string y has a higher logical depth.

1.3 Structure

In Chapter 2, we give the several models of computation that are used in this dissertation, together with their history. It starts with the systems of combinatory logic and the lambda calculus. After defining Turing machines, the notion of Kolmogorov complexity is given.

In Chapter 3 the background that is needed for the next chapter is given. It contains previously known results. We outline combinatory algebras and several structures for the lambda calculus. We also introduce polynomial algebras, and give the axiomatizations of Curry and Selinger. Chapter 4 is based on joint work

with Tatsuji Kawai and Hajime Ishihara [19]. We first introduce a structure called a *combinatory pre-model*. From this structure, we structurally add more requirements. One of these requirements is *reflexivity*. Finally, we arrive at a new structure called a *strongly reflexive combinatory pre-model*. We show that strongly reflexive combinatory pre-models are sound with respect to the lambda calculus, and we outline how the structure is related to other structures of the lambda calculus.

Part two of the dissertation starts with Chapter 5. We first discuss the necessary preliminaries for this part of the dissertation. In Chapter 6, we give the definition of logical depth, together with several related definitions. We also outline relations between the different definitions. Afterwards we highlight problems with the definition of logical depth. This is based on joint work with Akitoshi Kawamura.

Finally, Chapter 7 discusses the results in the dissertation.

Chapter 2

Historical background and models of computation

The beginning of the 1900's was a time when many were concerned with formal systems. Russel had just formulated his paradox which gives a contradiction in naive set theory. In naive set theory, every definable collection is a set. But when we consider the set of all sets that are not members of themselves, $R = \{x \mid x \notin x\}$, then we arrive at a contradiction both when $R \in R$ and $R \notin R$. This is known as *Russel's paradox*.

It was also a time when *Gödel's incompleteness theorems* were not yet published, which appeared in 1931. A formal system is *complete* if for every formula of the system either the formula or the negation is provable in the system. A formal system is *inconsistent* if there is a formula, such that both the formula and the negation are provable in the system. Gödel's incompleteness theorems only concern formal systems that can carry out some elementery arithmetic. The first incompleteness theorem states roughly that any such consistent formal system will always be incomplete. The second theorem states that such a formal system can never prove its own consistency. And thus, not knowing of these obstacles, several researchers were working on mathematical foundations, hoping to create a useful formal system and also use it to show its own consistency. Some abandoned the sets and worked on foundations based on functions instead. Combinatory logic and lambda calculus were results of such efforts.

2.1 Combinatory logic

Schönfinkel was the initiator of combinatory logic as it is know today. He was a member of a research group led by Hilbert [10]. Hilbert is mostly known for his program with which he wanted to solve the problem posed by Russell. Hilbert planned on formulating a foundation of mathematics and proving its own consistency, while only using *finetary methods*. That is, he wanted to restrict mathematics to only using objects that are "intuitively present as immediate experience prior to all thought"[20], without using infinite totalities [42]. Eventually Gödel's theorems put an end to this.

At the time, Schönfinkel's main goal was to reduce the number of primitive

notions in logic. Specifically, he showed how to eliminate bound variables from expressions [36].

Schönfinkel based himself on the observation that using functions in a single argument is enough, as we can successively apply these functions in order to obtain the same result as with using multi-variable functions. This process is currently known as "currying" even though it was initiated by Frege, whom Schönfinkel referred to in his paper. For example, the process of adding two numbers a and b, can be done by a function or operation $\oplus$ that only has one argument. If we apply $\oplus$ to a, we get another function $\oplus a$. When we apply this to b, and thus perform the operation $(\oplus a)b$, we obtain the result $a + b$.

Schönfinkel also introduced several *combinators*: specific functions that use function application on a single argument only. He then continued to show that we can create all formulas of predicate logic using the combinators now known as $\mathtt{k}$, $\mathtt{s}$ and a NAND-operator, without using any bound variable [36].

Curry set out on pursuing a similar goal of obtaining a more simplistic foundational system. After discovering Schönfinkel's work he continued working on a formal system using the combinators $\mathtt{k}$ and $\mathtt{s}$. He also added a rule that is known as extensionality (see Definition 3.1.10 later).

The formal systems of Schönfinkel and Curry were not so successful (similarly for the lambda calculus, as we will see). However, combinatory logic is now very well known as a model of computation.

Before we introduce the theory of Combinatory logic, we outline how variables are used. There are countably many variables. For each $i \in \mathbb{N}$, let $\mathtt{x}_i$ be a variable. For each $i, j \in \mathbb{N}$ with $i \neq j$, the variables $\mathtt{x}_i$ and $\mathtt{x}_j$ are distinct. We will often use $\mathtt{x}$, $\mathtt{y}$ or $\mathtt{z}$ to denote any variable.

Definition 2.1.1. *Combinatory logic* is a theory with the following alphabet.

1. Two constants $\mathtt{k}$ and $\mathtt{s}$, also known as *combinators*.
2. The variable $\mathtt{x}_i$ for each $i \in \mathbb{N}$.
3. The symbol $=$ for equality, the symbol $\cdot$ for application and $(\ ,\ )$ as parenthesis.

The set of terms T_{CL} of combinatory logic is defined inductively as follows.

1. The variable $\mathtt{x}_i \in \mathsf{T}_{\mathsf{CL}}$ for each $i \in \mathbb{N}$.
2. The combinators $\mathtt{k}$, $\mathtt{s} \in \mathsf{T}_{\mathsf{CL}}$.
3. For all terms s, $t \in \mathsf{T}_{\mathsf{CL}}$, also their application $(s \cdot t) \in \mathsf{T}_{\mathsf{CL}}$.

Notation 2.1.2. We will often omit the symbol $\cdot$ for application and use the convention of association to the left. We thus write rst instead of $((r \cdot s) \cdot t)$

Definition 2.1.3. The *formulae* of combinatory logic are $s = t$ for all s, $t \in \mathsf{T}_{\mathsf{CL}}$.

The *axioms and rules* for combinatory logic are the following. For all q, r, s, $t \in \mathsf{T}_{\mathsf{CL}}$:

k) $((\mathtt{k} \cdot r) \cdot s) = r$

s) $(((\mathtt{s} \cdot r) \cdot s) \cdot t) = (r \cdot t) \cdot (s \cdot t)$

refl) $r = r$

symm) $r = s \Rightarrow s = r$

trans) $r = s \wedge s = t \Rightarrow r = t$

cong) $q = r \wedge s = t \Rightarrow (q \cdot s) = (r \cdot t)$

Definition 2.1.4. For $s, t \in \mathsf{T}_{\mathsf{CL}}$ we write $\mathsf{CL} \vdash s = t$ when we can derive $s = t$ in combinatory logic using the rules and axioms above.

We give an example of this below.

Example 2.1.5. Let $s, t \in \mathsf{T}_{\mathsf{CL}}$. Then $\mathsf{CT} \models \mathtt{s}(\mathtt{k}(\mathtt{skk}))st = st$, since

$$\begin{aligned} \mathtt{s}(\mathtt{k}(\mathtt{skk}))st &= \mathtt{k}(\mathtt{skk})t(st) \\ &= \mathtt{skk}(st) \\ &= \mathtt{k}(st)(\mathtt{k}(st)) \\ &= st. \end{aligned}$$

It is also possible to define natural numbers in combinator logic. We can do computations by rewriting terms, only using $\mathtt{k}$ and $\mathtt{s}$. Although, written down, these computations are not so easy for us to follow. The lambda calculus is another model of computation, somewhat similar to combinatory logic, but more intuitive in this manner.

2.2 Lambda calculus

Church was also a researcher who wanted to make a formal system based on the notion of a function, without using bound variables [10]. Underlying to this system was the (untyped) "pure lambda-calculus". Eventually Church's system was proven inconsistent by two of his students, Rosser and Kleene [22]. However, the lambda-calculus turned out to be very useful.

Definition 2.2.1. The *lambda calculus* is a theory with the following *alphabet*.

1. The symbol '.' and the symbol λ.
2. The variable $\mathtt{x}_i$ for each $i \in \mathbb{N}$
3. The symbol $=$ for equality, the symbol $\cdot$ for application and (,) as parenthesis.

The set of *terms* T_Λ of the lambda calculus is defined inductively as follows.

1. The variable $\mathtt{x}_i \in \mathsf{T}_\Lambda$ for each $i \in \mathbb{N}$.
2. For all $t \in \mathsf{T}_\Lambda$ and any variable $\mathtt{x}$, the *lambda abstraction* $\lambda\mathtt{x}.t \in \mathsf{T}_\Lambda$.
3. For all terms $s, t \in \mathsf{T}_\Lambda$, also their application $(s \cdot t) \in \mathsf{T}_\Lambda$.

Before we can define the formulae and the axioms and rules of the lambda calculus, we need to give two additional definitions. First, we define the set of free variables for any term.

Definition 2.2.2. For any term $t \in \mathsf{T}_\Lambda$, the set $FV(t)$ of all *free variables* in t is defined inductively as follows.

1. $FV(\mathtt{x}) = \{\mathtt{x}\}$ for any variable $\mathtt{x}$.
2. $FV(\lambda\mathtt{x}.s) = FV(s) \setminus \{\mathtt{x}\}$ for any variable $\mathtt{x}$ and any $s \in \mathsf{T}_\Lambda$.
3. $FV(t_1 \cdot t_2) = FV(t_1) \cup FV(t_2)$ for any t_1, $t_2 \in \mathsf{T}_\Lambda$.

Using the set of free variables of a term, we can now define how to do substitution for variables that occur in a term.

Definition 2.2.3. For any terms s, $t \in \mathsf{T}_\Lambda$ and any variable $\mathtt{x}$, define a term $s[\mathtt{x}/t]$ obtained by substituting all occurrences of $\mathtt{x}$ in s by t as follows.

1. $\mathtt{x}[\mathtt{x}/t] \equiv t$.
2. $\mathtt{y}[\mathtt{x}/t] \equiv \mathtt{y}$ for any variable $\mathtt{y}$ different from $\mathtt{x}$.
3. $(\lambda\mathtt{y}.r)[\mathtt{x}/t] \equiv \lambda\mathtt{y}.(r[\mathtt{x}/t])$ for any $r \in \mathsf{T}_\Lambda$ and a variable $\mathtt{y}$, such that $\mathtt{y}$ is distinct from $\mathtt{x}$ and $\mathtt{y} \notin FV(t)$.
4. $(s_1 \cdot s_2)[\mathtt{x}/t] \equiv s_1[\mathtt{x}/t] \cdot s_2[\mathtt{x}/t]$ for any s_1, $s_2 \in \mathsf{T}_\Lambda$.

Using the above definition, we can proceed to the formulae and axioms and rules of the lambda calculus.

Definition 2.2.4. The *formulae* of the lambda calculus are $s = t$ for any s, $t \in \mathsf{T}_\Lambda$.

The *axioms and rules* of the lambda calculus are as follows. For all q, r, s, $t \in \mathsf{T}_\Lambda$ and for all variables $\mathtt{x}$:

α) $\lambda\mathtt{x}.t = \lambda\mathtt{y}.t[\mathtt{x}/\mathtt{y}]$ when $\mathtt{y} \notin FV(t)$

β) $((\lambda\mathtt{x}.s)t) = s[\mathtt{x}/t]$

refl) $r = r$

symm) $r = s \Rightarrow s = r$

trans) $r = s \wedge s = t \Rightarrow r = t$

cong) $q = s \wedge r = t \Rightarrow (qr) = (st)$

ξ) $s = t \Rightarrow \lambda\mathtt{x}.s = \lambda\mathtt{x}.t$

Definition 2.2.5. For any s, $t \in \mathsf{T}_\Lambda$, the statement $\lambda \vdash s = t$ holds when it is possible to derive $s = t$ in the lambda calculus using the rules and axioms above.

We can recreate Example 2.1.5 for the lambda calculus (for which the computation is now much simpler and more intuitive):

Example 2.2.6. Let s, $t \in \mathsf{T}_\Lambda$ and let $\mathtt{x}, \mathtt{y}$ be variables. Then $\lambda \vdash (\lambda\mathtt{xy}.\mathtt{xy})st = st$, since

$$\begin{aligned}(\lambda\mathtt{xy}.\mathtt{xy})st &= (\lambda\mathtt{y}.s\mathtt{y})t \\ &= st.\end{aligned}$$

2.3 Turing machines and the Church-Turing thesis

Gödel's incomlpeteness theorems put a halt to Hilbert's program. However, in a different way, Hilbert influenced the creation of another model of computation. Hilbert formulated the Entscheidungsproblem: is it possible to give an algorithm that determines whether a sentence of first order logic is satisfiable or not? This was considered one of the biggest, if not the biggest, problem for logicians in the 1930's.

The first to show the undecidability of the Entscheidungsproblem was Church in 1936 [13]. Independently it was shown (in a clearer way) by Turing. Turing was a student in Cambridge, and later ended up being a PhD student of Church. Inspired by lectures of Newman and the Entscheidungsproblem of Hilbert, Turing defined a machine for computing and used it to show the undecidability of the Entscheidungsproblem [40]. This machine of computing later became known as the *Turing machine*.

Definition 2.3.1. A *Turing machine* is a tuple (Γ, Q, δ), with a finite set Γ with symbols, a finite set Q of states (containing at least a start- and a halting-state), and a transition function $\delta\colon Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, R, S\}^k$ for some $k \in \mathbb{N}_{\geq 1}$.

The machine has a finite number of tapes (the k in the transition function). A tape is an infinite line of cells. Each cell contains one symbol from Γ, together with a tape head that scans one of these cells. The machine also has a register that contains the state of the machine.

The transition function describes a computational step. The register holds a state from Q and the machine reads k symbols under the tape heads. According to the transition function δ, the machine then gets into another state from Q, replaces the k symbols under the tape heads with k symbols from Γ, and on each tape the tape head either moves one cell in the left or right direction or stays in the same position.

At the start of the computation the machine is in the start-state. The *input* is a string with symbols from Γ written on the first tape. After several computational steps, the machine possibly gets into the halting state, with the output written on the k-th tape.

Definition 2.3.2. For any Turing machine M, any $x, y \in \{0,1\}^*$ and any $t \in \mathbb{N}$, if M, given x as input, halts after t steps and outputs y, then $M(x)$ and $T_M(x)$ are defined and equal to y and t respectively.

Turing stated that any "effective calculation" can be done by a Turing machine. Church said the same about the lambda calculus, and not soon after the Turing machine first appeared it was shown by Church and Turing that Turing machines and the lambda calculus computed the same functions. Combinatory logic, Post's systems the recursive functions defined by Gödel also compute the same functions as Turing machines. We now refer to the *Church-Turing thesis* as the statement that the computable functions are exactly those that Turing machine (or any of these other models) compute.

2.4 Kolmogorov complexity

The formulation of the Turing machine lead to the emergence of several new research fields. Especially the *universal Turing machine*, which was also defined by Turing, was important for this. The universal Turing machine is a Turing machine that can simulate the computation of any other Turing machine. By being able to refer to a single machine for all computations, people could start to study about the characteristics of the algorithms and the strings that are computed. The notion of Kolmogorov complexity tries to capture the information content or complexity of a string. It is defined as the length of the shortest program that produces the string.

Definition 2.4.1. For any $x \in \{0,1\}^*$ and any Turing machine M, the *Kolmogorov complexity* of x relative to M is defined as

$$C_M(x) = \min\{|p| : p \in \{0,1\}^*,\ M \text{ halts on } p \text{ and } M(p) = x\},$$

if there is a $p \in \{0,1\}^*$ such that M halts on p and $M(p) = x$, and otherwise $C_M(x) = \infty$ when such a p does not exist.

Just as different people all defined the notion of computable functions independently of each other, Kolmogorov complexity was also defined by three people independently.

Solomonoff was working on inductive inference, and introduced Kolmogorov complexity to obtain a universal a priori probability [39]. Using universal Turing machines, Solomonoff showed what is known as the *Invariance theorem*.

Theorem 2.4.2. *There exists a universal Turing machine U, such that for any Turing machine M, there exists a constant c_M such that for all $x \in \{0,1\}^*$ and $t \in \mathbb{N}$ for which $C_M(x) < \infty$,*

$$C_U(x) \leq C_M(x) + c_M.$$

This theorem tells us that the Kolmogorov complexity of a string is machine independent up to an additive constant, and it thus justifies the notion of Kolmogorov complexity.

Only a year later, Kolmogorov independently introduced the same definition of Kolmogorov complexity in [23]. He used it as a measure of information content and randomness. He also proved the Invariance theorem. Chaitin then also defined Kolmogorov complexity and proved the Invariance theorem a couple of years later. More history can be found in [31].

2.5 Physical complexity

The *physical complexity* of a string is the amount of useful information behind it. This could also be seen as a measure of the degree of organization. When Kolmogorov defined Kolmogorov complexity in [23], he already mentioned that the definition had as a disadvantage. Namely, it does not consider the effort needed to transform the program into an output. There could be strings that can be produced by a short programs, but all those need long computations. He wrote that he would study more about this and publish a paper on it. However, this paper never appeared.

About 15 years later, Bennett wanted to identify the conditions for physical systems to evolve into states that have high organization [8]. During this work he came to define something similarly as Kolmogorov earlier described. He defines the notion *logical depth*. Bennett himself discusses several possible definitions that are closely related to each other. In [41], a definition of Bennet that is similar to logical depth is related to several other definitions that, again, appeared independently. In Chapter 6, we will give the definition of logical depth and several other related definitions.

Another well-known definition is *sophistication*. It was defined (independently) by Koppel and Atlan in [24, 25]. This definition, however, takes a completely different approach. That is why the relation between sophistication and logical depth is not as clear-cut as the relation between the different models of computation. Still, there are several results that relate the two [2].

Part I

Computability

Chapter 3

Combinatory algebras and extensions

3.1 Combinatory algebras

We will define structures that enable us to perform computations in a similar way as combinatory logic does. The basis of these structures is the following.

Definition 3.1.1. An *applicative structure* is a structure $A = (\underline{A}, \cdot)$, where $\underline{A}$ is a set, called the *domain* of A, and $\cdot$ is a binary operation on $\underline{A}$, called the *application*.

We first outline how terms on any set are defined.

Definition 3.1.2. Let S be a set. The set $\mathcal{T}(S)$ of *terms over* S is inductively defined as follows:

i) $a \in \mathcal{T}(S)$ for each $a \in S$.

ii) For any s, $t \in \mathcal{T}(S)$, also the *application* $(s, t) \in \mathcal{T}(S)$.

The terms on A do contain variables (in contrast to what we will define later, see Definition 3.1.2). Define $\mathtt{X} = \{\mathtt{x}_i : i \in \mathbb{N} \text{ and } i \geq 1\}$, a countably infinite set of distinct variables. Then $\mathcal{T}(\mathtt{X} \cup \underline{A})$ are the terms on an applicative structure $A = (\underline{A}, \cdot)$.

Note that for an applicative structure $A = (\underline{A}, \cdot)$, application on two elements a, b of $\underline{A}$ is denoted as $a \cdot b$, whereas application of two terms s, $t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$ is denoted as (s, t).

Notation 3.1.3. For any applicative structure $A = (\underline{A}, \cdot)$ we will occasionally use the following notation. We often omit the symbol $\cdot$ for application in A. Following convention, parentheses are omitted following association to the left. That is, for any a, b, $c \in \underline{A}$, the element $abc \in \underline{A}$ is shorthand notation for $((a \cdot b) \cdot c)$. Where confusion doesn't arise, we will similarly abbreviate applications in $\mathcal{T}(\mathtt{X} \cup \underline{A})$. So for any r, s, $t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$, the term $((r, s), t)$ can be written as rst.

Similarly to Definition 3.1.4, we define the set of free variables for a term of an applicative structure.

Definition 3.1.4. For any applicative structure $A = (\underline{A}, \cdot)$, for any term $t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$, the set $FV(t)$ of all *free variables* in t is defined inductively as follows.

i) $FV(\mathtt{x}) = \{\mathtt{x}\}$ for any variable $\mathtt{x}$.

ii) $FV(a) = \emptyset$ for any $a \in \underline{A}$.

iii) $FV(t_1, t_2) = FV(t_1) \cup FV(t_2)$ for any $t_1,\ t_2 \in \mathcal{T}(\mathtt{X} \cup \underline{A})$.

Substitution is also similar as Definition 2.2.3. In this part of the dissertation, we use $\equiv$ to denote syntactically equal terms.

Definition 3.1.5. For any applicative structure $A = (\underline{A}, \cdot)$, for any terms $s,\ t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$ and for any variable $\mathtt{x}$, define a term $s[\mathtt{x}/t]$ obtained by substituting all occurrences of $\mathtt{x}$ in s by t as follows.

i) $\mathtt{x}[\mathtt{x}/t] \equiv t$.

ii) $\mathtt{y}[\mathtt{x}/t] \equiv \mathtt{y}$ for any variable $\mathtt{y}$ different from $\mathtt{x}$.

iii) $(s_1, s_2)[\mathtt{x}/t] \equiv (s_1[\mathtt{x}/t], s_2[x/t])$ for any $s_1,\ s_2 \in \mathcal{T}(\mathtt{X} \cup \underline{A})$.

Notation 3.1.6. For any $n \in \mathbb{N}$, any $s, t_1, \ldots, t_n \in \mathsf{T}(A)$ and for any variables $\mathtt{x}_1, \ldots, \mathtt{x}_n$, denote $s[\mathtt{x}_1/t_1, \mathtt{x}_2/t_2, \ldots, \mathtt{x}_n/t_n] \equiv (\cdots((s[\mathtt{x}_1/t_1])[\mathtt{x}_2/t_2])\cdots)[\mathtt{x}_n/t_n]$.

To see which formulas are true in an applicative structure, we define the following.

Definition 3.1.7. A *valuation* on an applicative structure A is a mapping ρ from $\mathtt{X}$ to $\underline{A}$. For a valuation ρ, the *interpretation* in A is a mapping $[\![\cdot]\!]^A_\rho$ from $\mathcal{T}(\mathtt{X} \cup \underline{A})$ to A. It is inductively defined as follows.

Let A be an applicative structure and let ρ be a valuation on A. Then,

i) $[\![\mathtt{x}_i]\!]^A_\rho = \rho(\mathtt{x}_i)$ for all $i \in \mathbb{N}_{\geq 1}$.

ii) $[\![a]\!]^A_\rho = a$ for all $a \in \underline{A}$.

iii) $[\![(s,t)]\!]^A_\rho = ([\![s]\!]^A_\rho \cdot [\![t]\!]^A_\rho)$ for two terms $s,\ t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$.

For a formula $s = t$, it is defined that $s = t$ is *true in* A *under the valuation* ρ, or $A, \rho \models s = t$, if $[\![s]\!]^A_\rho = [\![t]\!]^A_\rho$.

It is defined that $s = t$ is *true in* A, written as $A \models s = t$, if $A, \rho \models s = t$ for all valuations ρ.

Remark 3.1.8. Occasionally we consider the interpretation of a term that does not contain variables. Then, the interpretation is independent of the valuation. For an applicative structure $A = (\underline{A}, \cdot)$, the interpretation of a $t \in \mathcal{T}(\underline{A})$ is denoted as $[\![t]\!]^{\underline{A}}$.

We will see that the variables in the terms on an applicative are unnecessary and can be eliminated. As mentioned before, this goal was first set out by Schönfinkel in [36]. Curry formalized this requirement in the following way [15].

Definition 3.1.9. An applicative structure A is *combinatory complete* if for every $t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$ and for all $n \in \mathbb{N}$ and variables $\mathtt{x}_1, \mathtt{x}_2, \ldots, \mathtt{x}_n$, with $FV(t) \subseteq \{\mathtt{x}_1, \ldots, \mathtt{x}_n\}$, there exists an $a \in \underline{A}$ such that

$$A \models (\cdots(a, \mathtt{x}_1), \cdots, \mathtt{x}_n) = t.$$

Intuitively this means the following. We can interpret t as a function on n arguments, and the description of t contains these arguments as variables. But combinatory completeness ensures that we can always find an element a of $\underline{A}$ (that thus does not contain variables), such that a applied to n arguments gives the same result as the term t where the variables are substituted by these arguments.

However, there could be different elements $a_1, a_2, \ldots$ of $\underline{A}$ that satisfy this for the same $t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$. That is, it is possible that for all variables $\mathtt{x}_1, \mathtt{x}_2, \ldots, \mathtt{x}_n$, with $FV(t) \subseteq \{\mathtt{x}_1, \ldots, \mathtt{x}_n\}$, not only $A \models (\cdots(a_1, \mathtt{x}_1), \cdots, \mathtt{x}_n) = t$, but also $A \models (\cdots(a_2, \mathtt{x}_1), \cdots, \mathtt{x}_n) = t$, et cetera.

In order to enforce that the element of $\underline{A}$ that satisfies this is unique, Curry introduced the following notion [15].

Definition 3.1.10. An applicative structure $A = (\underline{A}, \cdot)$ is *extensional* when for all $a, b \in \underline{A}$,

$$[\forall c \in \underline{A} \quad a \cdot c = b \cdot c] \Rightarrow a = b$$

The following corollary makes clear why extensionality is sufficient.

Corollary 3.1.11. *An extensional applicative structure $A = (\underline{A}, \cdot)$ is combinatory complete if and only if for every $t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$ and for any $n \in \mathbb{N}$ and all variables $\mathtt{x}_1, \mathtt{x}_2, \ldots, \mathtt{x}_n$, with $FV(t) \subseteq \{\mathtt{x}_1, \ldots, \mathtt{x}_n\}$, there exists a unique $a \in \underline{A}$ such that $A \models (\cdots(a, \mathtt{x}_1), \cdots, \mathtt{x}_n) = t$. That is, for all $a, b \in \underline{A}$ such that $A \models (\cdots(a, \mathtt{x}_1), \cdots, \mathtt{x}_n) = t$ and $A \models (\cdots(b, \mathtt{x}_1), \cdots, \mathtt{x}_n) = t$, it holds that $a = b$.*

In order to eliminate the variables and satisfy combinatory completeness, we only need two specific elements, or *combinators* [36].

Definition 3.1.12. A *combinatory algebra* is an applicative structure $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s})$ with distinct elements $\mathtt{k}, \mathtt{s} \in \underline{A}$ (also referred to as *combinators*) such that for all $a, b, c \in \underline{A}$

i) $(\mathtt{k} \cdot a) \cdot b = a$,

ii) $((\mathtt{s} \cdot a) \cdot b) \cdot c = (a \cdot c) \cdot (b \cdot c)$.

In order to see that variables can be removed, it is the easiest to consider lambda abstractions. This is done by structural induction, as first proposed by Rosser in [35], inspired by [14].

Definition 3.1.13. Let $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s})$ be a combinatory algebra. For any $t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$ and any variable $\mathtt{x}$, define the lambda abstraction $\lambda^*\mathtt{x}.t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$ inductively as follows.

i) $\lambda^*\mathtt{x}.\mathtt{x} \equiv ((\mathtt{s}, \mathtt{k}), \mathtt{k})$.

ii) $\lambda^*\mathtt{x}.a \equiv (\mathtt{k}, a)$ for any $a \in (\mathtt{X} \cup \underline{A})$ different from $\mathtt{x}$.

iii) $\lambda^*\mathtt{x}.(t_1, t_2) \equiv ((\mathtt{s}, (\lambda^*\mathtt{x}.t_1)), (\lambda^*\mathtt{x}.t_2))$ for any t_1, $t_2 \in \mathcal{T}(\mathtt{X} \cup \underline{A})$.

Notation 3.1.14. For any $n \in \mathbb{N}$ and variables $\mathtt{x}_1, \ldots, \mathtt{x}_n$, define the repeated lambda abstraction $\lambda^*\mathtt{x}_1 \cdots \mathtt{x}_n.t$ as $\lambda^*\mathtt{x}_1.(\cdots(\lambda^*\mathtt{x}_n.t)\cdots)$.

Remark 3.1.15. When $t \in \mathcal{T}(\{\mathtt{x}_1, \ldots, \mathtt{x}_n\} \cup \underline{A})$ for some $n \in \mathbb{N}$, then for all $i \in \{1, \ldots, n\}$, $\lambda^*\mathtt{x}_i.t \in \mathcal{T}(\{\mathtt{x}_1, \ldots, \mathtt{x}_n\} \setminus \{\mathtt{x}_i\} \cup \underline{A})$.

We also have the following.

Proposition 3.1.16. *Let $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s})$ be a combinatory algebra. For any variable $\mathtt{x}$, any s, $t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$,*

$$A \models (\lambda^*\mathtt{x}.t, s) = t[\mathtt{x}/s].$$

Proof. By induction on $n \in \mathbb{N}$ and the structure of $t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$. □

This result automatically extends to any number of applications (when using repeated abstraction and substitution). It shows that combinatory algebras are sufficient for the requirement of combinatory completeness. Moreover, as we will argue below, combinatory algebras turn out to be necessary for combinatory completeness. This shows the expressive power of the combinators $\mathtt{k}$ and $\mathtt{s}$.

Theorem 3.1.17 ([7, p. 5.1.10])**.** *An applicative structure A is combinatory complete if and only if it can be expanded to a combinatory algebra, by choosing two of its elements to be the constants $\mathtt{k}$ and $\mathtt{s}$.*

Proof. Proposition 3.1.16 tells us that every combinatory algebra is combinatory complete. For the other direction, because of combinatory completeness we can choose for $\mathtt{k}$ an $a \in \underline{A}$ such that for all $b, c \in \underline{A}$ $(a \cdot b) \cdot c = b$. Similarly for $\mathtt{s}$. □

3.2 Lambda algebras and Curry's equations

As seen in Definition 3.1.13 and Proposition 3.1.16, we can perform a lambda abstraction in a combinatory algebra. However, the lambda abstraction λ^* of combinatory algebras does not behave in the same way as λ in the lambda calculus. That is, there are formulas that are derivable in the lambda calculus using λ but not with a combinatory algebra using λ^*.

In order to make the interpretation of lambda terms using λ^* formal, we will define two mappings between terms of a combinatory algebra and terms of the lambda calculus. First, we need to extend the definition of the lambda terms, Definition 2.2.1, to contain constants of a specific combinatory algebra.

Definition 3.2.1. Let $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s})$ be a combinatory algebra. The set of terms $\mathsf{T}_\Lambda(A)$ is defined inductively as follows.

i) $\mathtt{x}_i \in \mathsf{T}_\Lambda(A)$ for all $i \in \mathbb{N}$.

ii) $a \in \mathsf{T}_\Lambda(A)$ for all $a \in \underline{A}$.

iii) $\lambda \mathtt{x}_i.t \in \mathsf{T}_\Lambda(A)$ for all $i \in \mathbb{N}$ and $t \in \mathsf{T}_\Lambda(A)$.

iv) $(s \cdot t) \in \mathsf{T}_\Lambda(A)$ for all $s, t \in \mathsf{T}_\Lambda(A)$.

For a combinatory algebra A, for any $s, t \in \mathsf{T}_\Lambda(A)$, we define $\lambda \vdash s = t$ similarly as in Defininition 2.2.5, extending the rules and axioms of Definition 2.2.4 to terms of $\mathsf{T}_\Lambda(A)$.

We now continue to the mappings between terms of a combinatory algebra and the extended lambda terms.

Definition 3.2.2. Let A be a combinatory algebra. The mapping $(\cdot)_{CL^*} \colon \mathsf{T}_\Lambda(A) \to \mathcal{T}(\mathtt{X} \cup \underline{A})$ is defined as follows.

i) $\mathtt{x}_{CL^*} = \mathtt{x}$ for variables $\mathtt{x}$.

ii) $c_{CL^*} = c$ for constants c, .

iii) $(s \cdot t)_{CL^*} = (s_{CL^*}, t_{CL^*})$ for $s, t \in \mathsf{T}_\Lambda(A)$.

iv) $(\lambda \mathtt{x}.s)_{CL^*} = \lambda^* \mathtt{x}.s_{CL^*}$ For a term $s \in \mathsf{T}_\Lambda(A)$.

Similarly define $(\cdot)_\Lambda : \mathcal{T}(\mathtt{X} \cup \underline{A}) \to \mathsf{T}_\Lambda(A)$.

i) $x_\Lambda = \mathtt{x}$ for variables $\mathtt{x}$.

ii) $\mathtt{k}_\Lambda = \lambda \mathtt{xy}.\mathtt{x}$ and $\mathtt{s}_\Lambda = \lambda \mathtt{xyz}.\mathtt{xz}(\mathtt{yz})$, where $\mathtt{k}, \mathtt{s} \in \mathsf{T}(A)$ denote the constants associated with the similar elements $\mathtt{k}, \mathtt{s} \in \underline{A}$.

iii) $c_\Lambda = c$ for constants $c \in \mathsf{T}(A)$, where c is not $\mathtt{k}$ or $\mathtt{s}$.

iv) $(s, t)_\Lambda = (s_\Lambda \cdot t_\Lambda)$ for $s, t \in \mathsf{T}(A)$.

As mentioned before, there are formulae that are derivable in the lambda calculus, but not in a combinatory algebra. We will give an example of such a formula.

We have $\lambda \vdash \lambda \mathtt{y}.(\lambda \mathtt{x}.\mathtt{x})\mathtt{y} = \lambda \mathtt{y}.\mathtt{y}$. However, $(\lambda \mathtt{y}.(\lambda \mathtt{x}.\mathtt{x})\mathtt{y})_{CL^*} = \mathtt{s}(\mathtt{k}(\mathtt{skk})))(\mathtt{skk})$ and $(\lambda \mathtt{y}.\mathtt{y})_{CL^*} = \mathtt{skk}$.

The problem is that in a combinatory algebra A, when $A \models a = b$, we cannot always derive that $A \models \lambda^* \mathtt{x}.a = \lambda^* \mathtt{x}.b$, but this is true in the lambda calculus (as the ξ-rule, see Definition 2.2.4).

A specific applicative structure where all formulas of the lambda calculus hold, is called a *lambda algebra*:

Definition 3.2.3. A combinatory algebra A is a *lambda algebra* when for all $s, t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$

$$\lambda \vdash s_\Lambda = t_\Lambda \Rightarrow A \models s = t.$$

Or, alternatively, it can be characterized in the following way.

Lemma 3.2.4. *A combinatory algebra $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s})$ is a lambda algebra if and only if:*

1. $\lambda \vdash s = t \Rightarrow A \models s_{CL^*} = t_{CL^*}$ *for all terms* $s, t \in \mathsf{T}_\Lambda(A)$.

2. $\mathtt{k} = [\![\lambda^*\mathtt{xy.x}]\!]^{\underline{A}}$ *and* $\mathtt{s} = [\![\lambda^*\mathtt{xyz.xz(yz)}]\!]^{\underline{A}}$

Proof. See Lemma 5.2.3 in [7]. □

Since we can simulate lambda-abstractions using λ^*, we can ask ourselves what exactly is needed for a combinatory algebra to respect the β-equality of the lambda calculus.

In the first place, this question was asked with respect to the systems of combinatory logic and the lambda calculus, since we can define λ^* similarly for terms of T_{CL}. For this purpose, Rosser gave several equations that could be added to combinatory logic in order such that the λ^*-abstraction respected the rules of the lambda calculus. However, Rosser's equations contained a mistake. In [16], Curry gave five axioms for this purpose.

He derived these using the following reasoning. Similarly as Proposition 3.1.16, the β-rule is already satisfied. Then it is thus sufficient to show that the following holds for any $s, t \in \mathsf{T}_{\mathsf{CL}}$ (cf. the ξ- rule of Definition 2.2.4).

$$\mathsf{CL} \vdash s = t \Rightarrow \mathsf{CL} \vdash \lambda^*\mathtt{x}.s = \lambda^*\mathtt{x}.t,$$

where for any $s \in \mathsf{T}_{\mathsf{CL}}$ we define $\lambda^*\mathtt{x}.s$ similarly as in Definition 3.1.13. It satisfies to find a set of axioms $\mathcal{C}$ such that for any axiom $\mathcal{A}$ of combinatory logic, also the formula $\mathcal{A}'$ is true, obtained by applying the ξ-rule to $\mathcal{A}$. For example, we want that for any variables s, $t \in \mathsf{T}_{\mathsf{CL}}$, the formula $\lambda^*\mathtt{x}.\mathtt{k}st = \lambda^*\mathtt{x}.s$ is true. It is also necessary that the set of axioms $\mathcal{C}$ ensures that the similarly modified axioms $\mathcal{C}'$ are true.

By looking at what is needed for all this to hold, Curry finds a set of five closed axioms. Denote $\mathsf{CL} + \mathbf{C}$ for the theory of combinatory logic that is extended with the axioms of Curry. For any $s \in \mathsf{T}_\Lambda$, denote s_{CL^*} for the interpretation of the term in T_{CL}, similarly as in Definition 3.2.2. Then the following holds.

Proposition 3.2.5. *For any* $s, t \in \mathsf{T}_\Lambda$

$$\lambda \vdash s = t \Leftrightarrow \mathsf{CL} + \mathbf{C} \vdash s_{CL^*} = t_{CL^*}.$$

Proof. See [16] or Theorem 7.3.10 in [7]. □

In particular, the ξ-rule for the lambda calculus is respected, since for any $s, t \in \mathsf{T}_{\mathsf{CL}}$

$$\mathsf{CL} + \mathbf{C} \vdash s = t \Rightarrow \mathsf{CL} + \mathbf{C} \vdash \lambda^*\mathtt{x}.s = \lambda^*\mathtt{x}.t. \tag{3.1}$$

We can formulate these axioms similarly for combinatory algebras. Our presentation comes from Definition 7.3.6 in [7].

Theorem 3.2.6. *A combinatory algebra* $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s})$ *is a lambda algebra if and only if it satisfies the following equations.*

1. $\mathtt{k} = [\![\lambda^*\mathtt{xy.kxy}]\!]^{\underline{A}}$

2. $\mathtt{s} = [\![\lambda^*\mathtt{xyz.sxyz}]\!]^{\underline{A}}$

3. $[\![\lambda^*\mathtt{xy.s(kx)(ky)}]\!]^{\underline{A}} = [\![\lambda^*\mathtt{xz.k(xy)}]\!]^{\underline{A}}$

4. $[\![\lambda^*\mathtt{xy.s(s(kk)x)y}]\!]^{\underline{A}} = [\![\lambda^*\mathtt{xyz.xz}]\!]^{\underline{A}}$

5. $[\![\lambda^*\mathtt{xyz.s(s(s(ks)x)y)z}]\!]^{\underline{A}} = [\![\lambda^*\mathtt{xyz.s(sxz)(syz)}]\!]^{\underline{A}}$

Proof. See Theorem 5.2.5 in [7] □

This shows that the notion of a lambda algebra can be defined from a combinatory algebra by only adding closed equations.

The counterpart of equation (3.1) does not hold. That is, there exists a lambda algebra A and terms $s, t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$, such that $A \models s = t$ but $A \not\models \lambda^*\mathtt{x}.s = \lambda^*\mathtt{x}.t$. In other words, lambda algebras do not respect the ξ-rule of the lambda calculus. We show this in Section 3.5.2. In the next section we introduce a model that does respect the ξ-rule of the lambda calculus.

In Section 3.5.3, we will discuss in what different way lambda algebras respect the ξ-rule.

3.3 Lambda models

In this section and the next one, we introduce two other structures of the lambda calculus. Namely, the lambda model of Scott and the combinatory model from Meyer. A crucial axiom for both these structures is the *Meyer-Scott* axiom. For a combinatory algebra $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s})$, define $\mathbf{1}$ to denote the element $\mathtt{s}(\mathtt{k}(\mathtt{skk})) \in \underline{A}$. This element corresponds to the lambda term $\lambda\mathtt{xy.xy}$ (see Example 2.1.5).

Definition 3.3.1. Let $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s})$ be a combinatory algebra. The *Meyer-Scott axiom* for A is

$$\forall c \in \underline{A}\ (ac = bc) \Rightarrow \mathbf{1}a = \mathbf{1}b$$

for all $a, b \in \underline{A}$.

We explain how the Meyer-Scott axiom relates to the ξ-rule of the lambda calculus. Recall the notion of extensionality (cf. Definition 3.1.10). The following is a variation of it.

Definition 3.3.2. A combinatory algebra $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s})$ is *weakly extensional* when for all variables $\mathtt{x}$ and any $s, t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$

$$A \models s = t \Rightarrow A \models \lambda^*\mathtt{x}.s = \lambda^*\mathtt{x}.t.$$

Weak extensionality and the Meyer-Scott axiom are closely related.

Proposition 3.3.3. *Let $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s})$ be a lambda algebra. Then A is weakly extensional if and only if A satisfies the Meyer-Scott axiom.*

Proof. See 5.2.9 in [7]. □

We can now define the lambda model.

Definition 3.3.4 ([37])**.** A *lambda model* is a lambda algebra A such that the Meyer-Scott axiom holds in A.

Proposition 3.3.3 tells us that lambda models are lambda algebras which are sound with respect to the ξ-rule of the lambda calculus. In Remark 4.6.12 we say something about the existence of lambda algebras which are not lambda models.

3.4 Combinatory models

There is a simpler structure than the lambda model, introduced by Meyer in [32], with an extra combinator that is similar to **1**.

Definition 3.4.1 ([32])**.** A *combinatory model* is defined as a combinatory algebra $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{e})$ with an additional element $\mathtt{e} \in \underline{A}$ such that for all $a, b \in \underline{A}$:

i) $\mathtt{e}ab = ab$

ii) $\forall c \in \underline{A}(ac = bc) \Rightarrow \mathtt{e}a = \mathtt{e}b$

The equation (ii) is similar to the Meyer-Scott axiom (and hence it's name).

In Section 4.4, it will become clear in what way combinatory models are sound with respect to the lambda calculus.

We discuss the relationship between combinatory models and lambda models. It turns out that lambda models are precisely the combinatory models that satisfy some extra requirements. Before we can outline this, we first need to define an iteration of the combinator $\mathtt{e}$.

Definition 3.4.2 ([37])**.** Let $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{e})$ be a combinatory model. For each $n \in \mathbb{N}_{\geq 1}$ we define $\varepsilon_n \in \underline{A}$ inductively as follows

$$\varepsilon_1 = \mathtt{e}, \qquad \varepsilon_{n+1} = \mathtt{s}(\mathtt{ke})(\mathtt{s}(\mathtt{k}\varepsilon_n)).$$

The workings of ε_n for $n \in \mathbb{N}_{\geq 1}$ can best be understood according to the following (see Section 6 in [32]).

Lemma 3.4.3. *Let $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{e})$ be a combinatory model. For all $n \in \mathbb{N}$ and for all a, $b \in \underline{A}$,*

$$\varepsilon_{n+1}ab = \varepsilon_n(ab)$$

In particular, for all $n \in \mathbb{N}$ and $a_1, \ldots a_n \in \underline{A}$,

i) $\varepsilon_n a_1 \cdots a_n = \mathtt{e}(a_0 \cdots a_n)$,

ii) $\varepsilon_n a_1 \cdots a_{n+1} = a_0 \cdots a_{n+1}$.

We define a special class of combinatory models.

Definition 3.4.4. A combinatory model $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{e})$ is *stable* if it satisfies the following equations.

$$\varepsilon_2\mathtt{k} = \mathtt{k}, \qquad \varepsilon_3\mathtt{s} = \mathtt{s}, \qquad \varepsilon_2\mathtt{e} = \mathtt{e}.$$

Because of stability, the $\mathtt{e}$-combinator can be removed when it is applied to other combinators.

Lemma 3.4.5. *Let $(\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{e})$ be a combinatory model. Then the following implications hold.*

1. $\varepsilon_2\mathtt{k} = \mathtt{k} \Leftrightarrow \mathtt{ek} = \mathtt{k} \wedge \mathtt{e}(\mathtt{k}a) = \mathtt{k}a$ for all $a \in \underline{A}$.

2. $\varepsilon_3\mathtt{s} = \mathtt{s} \Leftrightarrow \mathtt{es} = \mathtt{s} \wedge \mathtt{e}(\mathtt{s}a) = \mathtt{s}a \wedge \mathtt{e}(\mathtt{s}ab) = \mathtt{s}ab$ *for all* $a\,b \in \underline{A}$.

3. $\varepsilon_2\mathtt{e} = \mathtt{e} \Leftrightarrow \mathtt{ee} = \mathtt{e}$.

Proof. See Lemma 5.6.5 in [7]. □

A stable combinatory model fixes a unique interpretation for the combinators $\mathtt{k}$ and $\mathtt{s}$, determined by $\mathtt{e}$.

Proposition 3.4.6. *When* $(\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{e})$ *is a stable combinatory model, then for any other stable combinatory model* $(\underline{A}, \cdot, \mathtt{k}', \mathtt{s}', \mathtt{e})$,

$$\mathtt{k} = \mathtt{k}' \qquad\qquad \mathtt{s} = \mathtt{s}'.$$

Proof. From Proposition 5.6.6 in [7]. □

Lambda models are stable combinatory models.

Theorem 3.4.7 ([7, p. 5.6.6])**.** *For a combinatory model* $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{e})$, *the following are equivalent.*

1. *A is a stable combinatory model.*

2. $(\underline{A}, \cdot, \mathtt{k}, \mathtt{s})$ *is a lambda model and* $\mathtt{e} = \mathbf{1}$.

Proof. See Proposition 5.6.6 in [7] □

Remark 3.4.8. From Theorem 3.4.7 and Proposition 3.4.6, we thus have that lambda models are the models of the lambda calculus which respect the ξ-rule of the lambda calculus, and which can uniquely identify the combinators $\mathtt{k}$ and $\mathtt{s}$. Lambda algebras also uniquely identify the combinators $\mathtt{k}$ and $\mathtt{s}$, as can be seen from equation (2) from Lemma 3.2.4. In Section 4.6, we will see that lambda algebras can be characterized as a certain class of stable combinatory algebras.

It is possible to make a combinatory model stable by redefining the combinators.

Proposition 3.4.9. *If* $(\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{e})$ *is a combinatory model, then* $(\underline{A}, \cdot, \varepsilon_2\mathtt{k}, \varepsilon_3\mathtt{s}, \mathtt{ee})$ *is a lambda model.*

Proof. See 5.6.6 in [7] □

3.5 Lambda algebras and polynomial algebras

As mentioned before, lambda algebras do not generally respect the ξ-rule of the lambda calculus. Selinger observed in [38] that this is due to the way we interpret free variables in a model. Usually, as in Definition 3.1.7, the variables are interpreted as elements of the domain of a lambda algebra. We can also consider a different interpretation, where the variables are interpreted as *indeterminates*, variables in the context of polynomials. In this setting, lambda abstractions are interpreted as polynomials and the lambda algebra respects the ξ-rule of the lambda calculus.

In this section we first define polynomial algebras. We then use these to define the alternative interpretation as introduced by Selinger, and we show how the lambda algebra respects the ξ-rule using this interpretation. We then give an alternative characterization of lambda algebras, also proposed by Selinger.

3.5.1 Polynomial algebras

A polynomial algebra is an combinatory algebra to which indeterminates are added to the domain. These indeterminates are distinct from the other elements. For discussing the properties of polynomial algebras we often consider homomorphisms between these. We therefore also define the notion of a homomorphism between applicative structures and combinatory algebras.

Definition 3.5.1. Let $A = (\underline{A}, \cdot_A)$ and $B = (\underline{B}, \cdot_B)$ be applicative structures. A *homomorphism* between A and B is a function $f\colon \underline{A} \to \underline{B}$ such that for any a, $b \in \underline{A}$,

$$f(a \cdot_A b) = f(a) \cdot_B f(b).$$

For combinatory algebras $A = (\underline{A}, \cdot_A, \mathtt{k}_A, \mathtt{s}_B)$ and $B = (\underline{B}, \cdot_B, \mathtt{k}_B, \mathtt{s}_B)$, a homomorphism between A and B has the additional requirement that $f(\mathtt{k}_A) = \mathtt{k}_B$ and $f(\mathtt{s}_A) = \mathtt{s}_B$.

Two applicative structures or two combinatory algebras are *isomorphic* when there exists a bijective homomorphism between them.

For polynomial algebras we use a specific equivalence relation, instead of equality, to define how the combinators $\mathtt{k}$ and $\mathtt{s}$ operate on the elements of this domain with variables. And thus, before giving the definition of polynomial algebras, we repeat the definition of quotient sets.

Definition 3.5.2. Let $A = (\underline{A}, \cdot)$ be an applicative structure. A *congruence relation* on A is an equivalence relation $\sim$ on A (that is, the relation is reflexive, transitive and symmetric), such that for each $a, b, c, d \in \underline{A}$

$$a \sim b \wedge c \sim d \Rightarrow a \cdot c \sim b \cdot d.$$

Definition 3.5.3. Let $\sim$ be an equivalence relation over a set S, and let $\langle s \rangle$ denote the equivalence class for a $s \in S$ under $\sim$. Then the *quotient set* $S/\!\sim$ is defined as $S/\!\sim = \{\langle s \rangle \colon s \in S\}$.

Let $A = (\underline{A}, \cdot)$ be an applicative structure, and let $\sim$ be a congruence relation on A. Then the *quotient* of A by $\sim$, denoted as $A/\!\sim$, is an applicative structure $(\underline{A}/\!\sim, *)$, where for any a, $b \in \underline{A}$ and $\langle a \rangle$, $\langle b \rangle \in \underline{A}/\!\sim$ it is defined that $\langle a \rangle * \langle b \rangle = \langle a \cdot b \rangle$. Define the homomorphism $\pi_\sim\colon A \to A/\!\sim$ as $\pi_\sim(a) = \langle a \rangle$ for all $a \in \underline{A}$.

We go back to defining polynomial algebras. We first outline how equality is defined in those structures. For any set S, we denote $(\mathcal{T}(S), \bullet)$ for the applicative structure where $s \bullet t = (s, t)$ for any s, $t \in \mathcal{T}(S)$.

Definition 3.5.4. Let $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s})$ be a combinatory algebra. Define $\approx_{\mathtt{X}}$ to be the smallest congruence relation on $(\mathcal{T}(\mathtt{X} \cup \underline{A}), \bullet)$, such that for all $a, b \in \underline{A}$ and all $r, s, t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$ the following holds.

i) $(a, b) \approx_{\mathtt{X}} a \cdot b$

ii) $((\mathtt{k}, s), t) \approx_{\mathtt{X}} s$

iii) $(((\mathtt{s}, r), s), t) \approx_{\mathtt{X}} ((r, t), (s, t))$

Denote $\langle t \rangle_{\mathtt{X}}$ for the equivalence classes of a $t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$ under $\approx_{\mathtt{X}}$.

Definition 3.5.5. Let $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s})$ be a combinatory. The *polynomial algebra* $A[\mathtt{X}]$ of A over $\mathtt{X}$ is a combinatory algebra $(\underline{A[\mathtt{X}]}, *_{\mathtt{X}}, \langle \mathtt{k} \rangle_{\mathtt{X}}, \langle \mathtt{s} \rangle_{\mathtt{X}})$, where $\underline{A[\mathtt{X}]} = \mathcal{T}(\mathtt{X} \cup \underline{A})/\approx_{\mathtt{X}}$ and for all $s, t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$,

$$\langle s \rangle_{\mathtt{X}} *_{\mathtt{X}} \langle t \rangle_{\mathtt{X}} = \langle (s, t) \rangle_{\mathtt{X}}.$$

Define the homomorphism $\sigma_A \colon A \to A[\mathtt{X}]$ by $\sigma_A(a) = \langle a \rangle_{\mathtt{X}}$ for any $a \in \underline{A}$.

3.5.2 Lambda algebras and lambda models

With polynomial algebras, we can actually give another relation between lambda algebras and lambda models.

Proposition 3.5.6. *A combinatory algebra A is a lambda algebra if and only if $A[\mathtt{X}]$ is a lambda model.*

Proof. From the Lambda Algebra Theorem in [32]. □

We can also show that there exist lambda algebras that are not weakly extensional. This follows from a result by Plotkin [34].

Theorem 3.5.7. *There exists a lambda algebra $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s})$, and $q, r \in \underline{A}$, such that $A \models (q, \mathtt{x}) = (r, \mathtt{x})$, but $(q, \mathtt{x}) \not\approx_{\mathtt{X}} (r, \mathtt{x})$.*

Proof. See [34]. □

Corollary 3.5.8. *There exists a lambda algebra $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s})$ and $s, t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$, such that $A \models s = t$, but $A \not\models \lambda^*\mathtt{x}.s = \lambda^*\mathtt{x}.t$.*

Proof. Let A be the lambda algebra from Theorem 3.5.7, and let $q, r \in \underline{A}$ are similarly from that theorem. Then $A \models (q, \mathtt{x}) = (r, \mathtt{x})$, but $(q, \mathtt{x}) \not\approx_{\mathtt{X}} (r, \mathtt{x})$. Let $s = (q, \mathtt{x})$ and $t = (r, \mathtt{x})$. Then $A \models s = t$. Now assume, towards a contradiction, that $A \models \lambda^*\mathtt{x}.s = \lambda^*\mathtt{x}.t$. We will show that this implies $(q, \mathtt{x}) \approx_{\mathtt{X}} (r, \mathtt{x})$, a contradiction.

Note that $\lambda^*\mathtt{x}.s$ and $\lambda^*\mathtt{x}.t \in \mathcal{T}(\underline{A})$ and thus, by definition, $[\![\lambda^*\mathtt{x}.s]\!]^{\underline{A}} = [\![\lambda^*\mathtt{x}.r]\!]^{\underline{A}}$. Then also $(q, \mathtt{x}) \approx_{\mathtt{X}} [\![\lambda^*\mathtt{x}.s]\!]^{\underline{A}} \cdot \mathtt{x} \approx_{\mathtt{X}} [\![\lambda^*\mathtt{x}.r]\!]^{\underline{A}} \cdot \mathtt{x} \approx_{\mathtt{X}} (r, \mathtt{x})$. □

3.5.3 The absolute interpretation

Originally, for a combinatory algebra $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s})$, we consider equality for two terms $s, t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$ by interpreting those terms for a valuation that maps the variables to elements of $\underline{A}$. With polynomial algebras, we can instead choose to consider the relation $s \approx_{\mathtt{X}} t$, as is done by Selinger in [38]. He calls this the *absolute interpretation*.

Lambda algebras now respect the ξ-rule of the lambda calculus in the following way.

Proposition 3.5.9. *For any lambda algebra $(\underline{A}, \cdot, \mathtt{k}, \mathtt{s})$ and any $s, t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$,*

$$s \approx_{\mathtt{X}} t \Rightarrow \lambda^*\mathtt{x}.s \approx_{\mathtt{X}} \lambda^*\mathtt{x}.t$$

Proof. Lemma 5 in [38]. □

The following is also true for lambda algebras.

Proposition 3.5.10. *For any lambda algebra* $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s})$ *and any* $s, t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$*,*

$$s \approx_{\mathtt{X}} t \Rightarrow A \models s = t.$$

Proof. Corollary 2 in [38]. □

By Proposition 3.3.3, we know that lambda models are weakly extensional, in contrast to lambda algebras, and thus do respect the ξ-rule using the original interpretation. From this, it can be deduced that any equation that is satisfied in a lambda model, is also satisfied absolutely.

Proposition 3.5.11. *Let* $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s})$ *be a lambda model. For any* $s, t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$*,*

$$A \models s = t \Leftrightarrow s \approx_{\mathtt{X}} t.$$

Proof. Proposition 6 in [38]. □

And thus, in lambda models, the polynomials are determined by their behaviour as a function. Where in lambda algebras the lambda abstraction is interpreted as a polynomial, in lambda models the lambda abstraction is interpreted as a function. This is thus due to the Meyer-Scott axiom. The same holds for combinatory models. In Proposition 4.4.10, we will specify this.

We can use the absolute interpretation to give another characterization of lambda algebras. Recall that $\mathbf{1}$ denotes the term $\mathtt{s}(\mathtt{k}(\mathtt{skk}))$ in a combinatory algebra A.

Theorem 3.5.12. *Let* $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s})$ *be a combinatory algebra. Then* A *s a lambda algebra if and only if it satisfies the following equations for all* r, s, $t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$*.*

1. $\mathbf{1}\mathtt{k} = \mathtt{k}$
2. $\mathbf{1}\mathtt{s} = \mathtt{s}$
3. $(\mathbf{1}, (\mathtt{k}, t)) \approx_{\mathtt{X}} (\mathtt{k}, t)$
4. $(\mathbf{1}, (\mathtt{s}, t)) \approx_{\mathtt{X}} (\mathtt{s}, t)$
5. $(\mathbf{1}, ((\mathtt{s}, s), t)) \approx_{\mathtt{X}} ((\mathtt{s}, s), t)$
6. $(\mathtt{s}(\mathtt{s}(\mathtt{kk}), s), t) \approx_{\mathtt{X}} (\mathbf{1}, s)$
7. $(\mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{ks}), r), s), t) \approx_{\mathtt{X}} (\mathtt{s}((\mathtt{s}, r), t), ((\mathtt{s}, s), t))$
8. $((\mathtt{s}, (\mathtt{k}, s)), (\mathtt{k}, t)) \approx_{\mathtt{X}} (\mathtt{k}, (s, t))$
9. $((\mathtt{s}, (\mathtt{k}, t)), \mathtt{skk}) \approx_{\mathtt{X}} (\mathbf{1}, t)$

Proof. Theorem 3 in [38]. □

3.6 Summary

So far we have seen three structures for the lambda calculus. When $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s})$ is a *lambda algebra*, then by definition for all s, $t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$,

$$\lambda \vdash s_\Lambda = t_\Lambda \Rightarrow A \models s = t.$$

Alternatively, lambda algebras are sound with respect to the lambda calculus in another way. Recall Lemma 3.2.4.

Lemma 3.2.4. *A combinatory algebra $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s})$ is a lambda algebra if and only if:*

1. $\lambda \vdash s = t \Rightarrow A \models s_{CL^*} = t_{CL^*}$ *for all terms* $s, t \in \mathsf{T}_\Lambda(A)$.
2. $\mathtt{k} = [\![\lambda^*\mathtt{xy.x}]\!]^{\underline{A}}$ *and* $\mathtt{s} = [\![\lambda^*\mathtt{xyz.xz(yz)}]\!]^{\underline{A}}$

In lambda algebras, the lambda abstraction is interpreted as a polynomial. Equality on polynomials respects the ξ-rule of the lambda calculus.

Proposition 3.5.9. *For any lambda algebra $(\underline{A}, \cdot, \mathtt{k}, \mathtt{s})$ and any s, $t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$,*

$$s \approx_{\mathtt{X}} t \Rightarrow \lambda^*\mathtt{x}.s \approx_{\mathtt{X}} \lambda^*\mathtt{x}.t$$

When we add the Meyer-Scott axiom to a lambda algebra, then it becomes a *lambda model*. By Proposition 3.3.3, we know that for a lambda model $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s})$, for all s, $t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$,

$$A \models s = t \Rightarrow A \models \lambda^*\mathtt{x}.s = \lambda^*\mathtt{x}.t.$$

In lambda models, the lambda abstraction is interpreted as a function. To be precise, all polynomials are determined by their behaviour as a function.

Proposition 3.5.11. *Let $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s})$ be a lambda model. For any s, $t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$,*

$$A \models s = t \Leftrightarrow s \approx_{\mathtt{X}} t.$$

In lambda algebras only the right to left direction holds. Lambda algebras can also be characterized as the *retracts* of lambda models (see Definition 4.2.8 and Remark 4.4.13 later).

Combinatory models also satisfy the Meyer-Scott axiom (albeit using the combinator $\mathtt{e}$). And thus, in combinatory models the lambda abstraction is also interpreted as a function. Combinatory models that satisfy stability (cf. Definition 3.4.4) are lambda models.

Chapter 4

Reflexive combinatory algebras

In the previous section, known structures for the lambda calculus and their properties are outlined. In this chapter we revisit this topic. We define *reflexivity* for combinatory algebras, which turns out to be important for interpreting the lambda calculus, but not sufficient. We also define a structure called a strongly reflexive combinatory algebra, which can interpret the lambda calculus, and is the equational counterpart of the combinatory model.

It relates to the structures of the previous chapter in the following way. A strongly reflexive combinatory algebra that is stable (similarly to Definition 3.4.4) is a lambda algebra. And a strongly reflexive combinatory algebra that satisfies the Meyer-Scott axiom is a combinatory model. Moreover, strongly reflexive combinatory algebras are the retracts (see Definition 4.2.8) of combinatory models. Figure 4.1 displays these relationships.

In what follows we outline the structure of the chapter together with our approach. Sections 4.1 and 4.2 are used for defining combinatory pre-models, an extension of combinatory algebras with two extra combinators, and the related polynomial algebras. We also give the necessary preliminaries for obtaining the rest of the results.

Since we can define a lambda abstraction for combinatory algebras that is sound with respect to the β-rule (cf. Proposition 3.1.16), the problem with interpreting the lambda calculus is the ξ-rule. We first consider polynomials and thus lambda abstractions using only one variable. There exists a mapping from the terms of a combinatory pre-model A to its polynomial algebra, $\varphi\colon a \mapsto a\mathtt{x}$, which is surjective because of combinatory completeness (Definition 3.1.9). The inverse function can be defined using the λ^*-abstraction, as can be seen by Proposition 3.1.16.

We want that the lambda abstraction respects the equality on polynomials (that is, when the ξ-rule is respected using the absolute interpretation). The equality of the polynomial algebra $A[\mathtt{x}]$ is defined by equations using indeterminates (cf. Definition 3.5.4), and thus we cannot easily use these for an axiomatisation. However, since φ is surjective, we can define an equivalence relation $\sim$ on A, such that $a \sim b$ if and only if $\varphi(a) = \varphi(b)$. In Section 4.2.4, we show how the quotient of A with respect to $\sim$ gives a combinatory pre-model that is isomorphic to $A[\mathtt{x}]$. The relation $\sim$ is generated by equations on the elements of A, without using indeterminates.

Using an algebraic analogue of the Meyer-Scott axiom, called *reflexivity*, we show

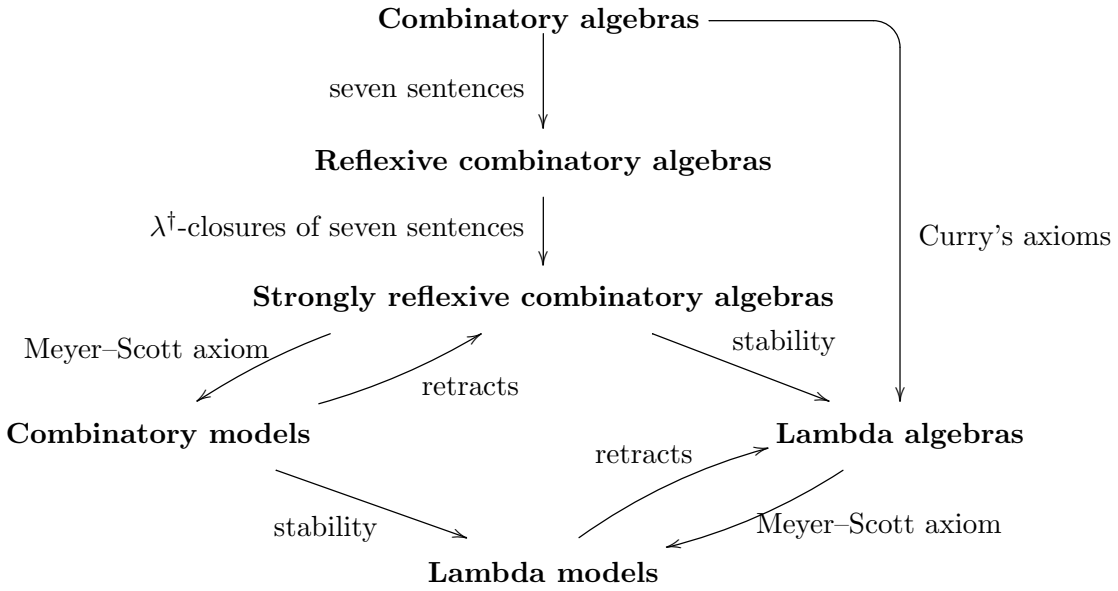


Figure 4.1: Relationship between various structures

in Section 4.3 that a combinatory pre-model has a defined lambda abstraction, denoted as $\lambda^\dagger$, as a well-defined operation on polynomials with one indeterminate, exactly when it satisfies seven universal sentences. These seven universal sentences are derived from requiring that the combinator `e` respects the equations that generate $\sim$. When we take the $\lambda^\dagger$-closures of these universal sentences, we obtain the requirements for a combinatory pre-model to have the $\lambda^\dagger$-abstraction as a well-defined operation on polynomials with any number of indeterminates. This gives us the notion of a strongly reflexive combinatory pre-model, and this is outlined in Section 4.4.

In Section 4.5, we discuss how to construct a cartesian closed category with a reflexive object from algebraic combinatory models, as was previously done for lambda algebras. In Section 4.6, it is shown that lambda algebras can be obtained from algebraic combinatory models by adding stability.

4.1 Combinatory pre-models

The elements `i` and `e` are commonly defined in terms of `k` and `s` as `i` = `skk` and `e` = `s(ki)` (cf. **1** from Definition 3.3.1). However, in the rest of this chapter, the constants `i` and `e` play a large role. We therefore define an extension of the combinatory algebra, where these are included as primitives. This has several benefits. Namely, the constants will appear in definitions, and by fixing them as primitives we secure the interpretation of these definitions. It also entails a generalization of the results, since any result that contains `e` or `i` in the statement will be valid for other definitions of `e` or `i`, as is the case for `k` and `s`. Additionally, the inclusion of `e` as a primitive is more natural when considering relations with combinatory models, where `e` is also

taken as primitive (cf. Definition 3.4.1).

Definition 4.1.1. A *combinatory pre-model* is defined as an applicative structure $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{i}, \mathtt{e})$ with distinct elements $\mathtt{k}$, $\mathtt{s}$, $\mathtt{i}$, $\mathtt{e} \in \underline{A}$ such that for all a, b, $c \in \underline{A}$

i) $\mathtt{k}ab = a$,

ii) $\mathtt{s}abc = ac(bc)$,

iii) $\mathtt{i}a = a$,

iv) $\mathtt{e}ab = ab$.

Note that any combinatory pre-model is also a combinatory algebra. A homomorphism between combinatory pre-models is defined similarly as for combinatory algebras (cf. Definition 3.5.1), and should additionally map the appropriate combinators $\mathtt{i}$ and $\mathtt{e}$ to each other.

Since we introduced the combinator $\mathtt{i}$, which takes on the same role as $\mathtt{skk}$, we redefine the λ^*-abstraction from Definition 3.1.13, specifically the definition of $\lambda^*\mathtt{x}.\mathtt{x}$ for a variable $\mathtt{x}$. We keep using the same notation λ^* for the abstraction for terms of a combinatory pre-model.

Definition 4.1.2. Let $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{i}, \mathtt{e})$ be a combinatory pre-model. For any variable $\mathtt{x}$ and any $t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$, define the lambda abstraction $\lambda^*\mathtt{x}.t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$ inductively as follows:

i) $\lambda^*\mathtt{x}.\mathtt{x} \equiv \mathtt{i}$.

ii) $\lambda^*\mathtt{x}.a \equiv (\mathtt{k}, a)$ for any $a \in (\mathtt{X} \cup \underline{A})$ different from $\mathtt{x}$.

iii) $\lambda^*\mathtt{x}.(t_1, t_2) \equiv ((\mathtt{s}, (\lambda^*\mathtt{x}.t_1)), (\lambda^*\mathtt{x}.t_2))$ for any t_1, $t_2 \in \mathcal{T}(\mathtt{X} \cup \underline{A})$.

In Section 3.5 we gave the definition of a polynomial algebra based on a combinatory algebra. This definition can be extended to combinatory pre-models, by adding the equations for $\mathtt{i}$ and $\mathtt{e}$ to the equivalence relation. For this definition we will use the same notation as before. Since from now on we will solely be using combinatory pre-models instead of combinatory algebras, this should cause no confusion. We define polynomial algebras over a finite number of indeterminates.

Definition 4.1.3. Let $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{i}, \mathtt{e})$ be a combinatory pre-model. For any $n \in \mathbb{N}_{\geq 1}$, let $\approx_{\mathtt{x}_1 \ldots \mathtt{x}_n}$ be the smallest congruence relation on $(\mathcal{T}(\{\mathtt{x}_1, \ldots, \mathtt{x}_n\} \cup \underline{A}), \bullet)$, such that for all $a, b \in \underline{A}$ and all $r, s, t \in \mathcal{T}(\{\mathtt{x}_1, \ldots, \mathtt{x}_n\} \cup \underline{A})$ the following holds.

i) $(a, b) \approx_{\mathtt{x}_1 \ldots \mathtt{x}_n} a \cdot b$

ii) $((\mathtt{k}, s), t) \approx_{\mathtt{x}_1 \ldots \mathtt{x}_n} s$

iii) $(((\mathtt{s}, r), s), t) \approx_{\mathtt{x}_1 \ldots \mathtt{x}_n} ((r, t), (s, t))$

iv) $(\mathtt{i}, a) \approx_{\mathtt{x}_1 \ldots \mathtt{x}_n} a$

v) $((\mathtt{e}, a), b) \approx_{\mathtt{x}_1 \ldots \mathtt{x}_n} (a, b)$

Denote $\langle t \rangle_n$ for the equivalence classes of a $t \in \mathcal{T}(\{\mathtt{x}_1, \ldots, \mathtt{x}_n\} \cup \underline{A})$ under $\approx_{\mathtt{x}_1 \ldots \mathtt{x}_n}$. The *polynomial algebra* $A[\mathtt{x}_1, \ldots \mathtt{x}_n]$ of A over the indeterminates $\mathtt{x}_1, \ldots, \mathtt{x}_n$ is a combinatory pre-model $(\underline{A[\mathtt{x}_1, \ldots, \mathtt{x}_n]}, *_n, \langle \mathtt{k} \rangle_n, \langle \mathtt{s} \rangle_n, \langle \mathtt{i} \rangle_n, \langle \mathtt{e} \rangle_n)$, where it is defined that $\underline{A[\mathtt{x}_1, \ldots, \mathtt{x}_n]} = \mathcal{T}(\{\mathtt{x}_1, \ldots, \mathtt{x}_n\} \cup \underline{A}) / \approx_{\mathtt{x}_1 \ldots \mathtt{x}_n}$ and

$$\langle s \rangle_n *_n \langle t \rangle_n = \langle (s, t) \rangle_n,$$

for all $s, t \in \mathcal{T}(\{\mathtt{x}_1, \ldots, \mathtt{x}_n\} \cup \underline{A})$. Define the homomorphism $\eta_A^n \colon A \to A[\mathtt{x}_1, \ldots, \mathtt{x}_n]$ by $\eta_A^n(a) = \langle a \rangle_n$ for all $a \in \underline{A}$.

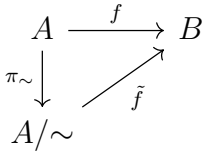
4.2 Properties of polynomial algebras

This section discusses mostly general knowledge on polynomial algebras. We start by showing the existence of unique homomorphisms from polynomial algebras to combinatory pre-models, and then use these results to establish relations between polynomial algebras with a different number of indeterminates.

4.2.1 Homomorphisms and substitution

The next lemma shows that for a homomorphism between applicative structures and a relation on the domain, there exists a natural homomorphism with the quotient as the domain.

Lemma 4.2.1. *Let $A = (\underline{A}, \cdot_A)$ and $B = (\underline{B}, \cdot_B)$ be applicative structures. Let $\sim$ be a congruence relation on $\underline{A}$, and let $f \colon A \to B$ be a homomorphism between combinatory pre-models, such that $f(a) = f(b)$ for all $a, b \in \underline{A}$ with $a \sim b$. Then there exists a unique homomorphism $\tilde{f} \colon A/\!\sim \to B$ such that $\tilde{f} \circ \pi_\sim = f$. That is, the following diagram commutes.*



Proof. Define $\tilde{f}$ by $\tilde{f}(\langle a \rangle) = f(a)$ for all $a \in \underline{A}$. Then $\tilde{f}$ is well-defined, since for $b, c \in \langle a \rangle$ we have that $f(b) = f(c)$. We also have that $\tilde{f}$ is a homomorphism, since for any $a, b \in \underline{A}$,

$$\begin{aligned}
&\tilde{f}(\langle a \rangle * \langle b \rangle) \\
=&\tilde{f}(\langle a \cdot_A b \rangle) \\
=&f(a \cdot_A b) \\
=&f(a) \cdot_B f(b) && \text{since } f \text{ is a homomorphism} \\
=&\tilde{f}(\langle a \rangle) \cdot_B \tilde{f}(\langle b \rangle).
\end{aligned}$$

For showing the uniqueness of $\tilde{f}$, assume that there is a $g : A/\!\sim \to B$ such that $g \circ \pi_\sim = f$. Then for any $a \in \underline{A}$, we have $g(\pi_\sim(a)) = g(\langle a \rangle) = f(a)$, and thus $g = \tilde{f}$. □

Using the homomorphism in the quotient, we can define a homomorphism in the polynomial algebra.

Proposition 4.2.2. *Let $A = (\underline{A}, \cdot_A, \mathtt{k}_A, \mathtt{s}_A, \mathtt{i}_A, \mathtt{e}_A)$ and $B = (\underline{B}, \cdot_B, \mathtt{k}_B, \mathtt{s}_B, \mathtt{i}_B, \mathtt{e}_B)$ be combinatory pre-models and let $f\colon A \to B$ be a homomorphism between them. For any $n \in \mathbb{N}_{\geq 1}$ and for all $b_1, \ldots, b_n \in \underline{B}$, there exists a unique homomorphism $\tilde{f}\colon A[\mathtt{x}_1, \ldots, \mathtt{x}_n] \to B$ such that $\tilde{f} \circ \eta_A^n = f$ and $\tilde{f}(\langle \mathtt{x}_i \rangle_n) = b_i$ for each $i \in \{1, \ldots, n\}$. And thus the following diagram commutes.*

$$\begin{array}{ccc} A & \xrightarrow{\quad f \quad} & B \\ {\scriptstyle \eta_A^n}\downarrow & \nearrow_{\tilde{f}} & \\ A[\mathtt{x}_1, \ldots, \mathtt{x}_n] & & \end{array}$$

Proof. Let A and B be two combinatory pre-models as in the statement of the proposition. Let $f\colon A \to B$ be a homomorphism, let $n \in \mathbb{N}$ and let $b_1, \ldots, b_n \in \underline{B}$. Define A' to be the combinatory pre-model $(\mathcal{T}(\{\mathtt{x}_1, \ldots, \mathtt{x}_n\} \cup \underline{A}), \bullet, \mathtt{k}_A, \mathtt{s}_A, \mathtt{i}_A, \mathtt{e}_A)$, where $s \bullet t = (s, t)$ for any $s, t \in \mathcal{T}(\{\mathtt{x}_1, \ldots, \mathtt{x}_n\} \cup \underline{A})$.

Define $f_b\colon A' \to B$ inductively as follows.

1. $f_b(\mathtt{x}_i) = b_i$ for each $i \leq n$,
2. $f_b(a) = f(a)$ for $a \in \underline{A}$,
3. $f_b((s, t)) = f_b(s) \cdot_B f_b(t)$ for $s, t \in \mathcal{T}(\{\mathtt{x}\} \cup \underline{A})$.

It is clear that f_b is a homomorphism between the combinatory pre-models A' and B. We show that $s \approx_{\mathtt{x}_1 \ldots \mathtt{x}_n} t$ implies $f_b(s) = f_b(t)$ for any $s, t \in \mathcal{T}(\{\mathtt{x}_1, \ldots, \mathtt{x}_n\} \cup \underline{A})$. We do this by showing that f_b respects the relation $\approx_{\mathtt{x}_1 \ldots \mathtt{x}_n}$, by induction on the axioms for $\approx_{\mathtt{x}_1 \ldots \mathtt{x}_n}$.

First consider equation (i) of Definition 4.1.3. For $a, b \in \underline{A}$:

$$f_b((a, b)) = f(a) \cdot_B f(b) = f(a \cdot_A b) = f_b(a \cdot_A b)$$

For showing that equation (ii) holds, let $s, t \in \mathcal{T}(\{\mathtt{x}_1, \ldots, \mathtt{x}_n\} \cup \underline{A})$. Then,

$$\begin{aligned} f_b(((\mathtt{k}_A, s), t)) &= (f_b(\mathtt{k}_A) \cdot_B f_b(s)) \cdot_B f_b(t) && f_b \text{ homomorphism} \\ &= (\mathtt{k}_B \cdot_B f_b(s)) \cdot_B f_b(t) && f_b \text{ homomorphism} \\ &= f_b(s) && \text{by rule of } \mathtt{k}_B \text{ in } B. \end{aligned}$$

The other axioms follows similarly. We now have for $s, t \in \mathcal{T}(\{\mathtt{x}_1, \ldots, \mathtt{x}_n\} \cup \underline{A})$, that $s \approx_{\mathtt{x}_1 \ldots \mathtt{x}_n} t$ implies that $f_b(s) = f_b(t)$. And thus f_b satisfies the requirements of the function f in Lemma 4.2.1. From that lemma we deduce that there exists a unique homomorphism $\tilde{f}\colon A[\mathtt{x}_1, \ldots, \mathtt{x}_n] \to B$ such that $\tilde{f} \circ \pi_{\approx_{\mathtt{x}_1 \ldots \mathtt{x}_n}} = f_b$, (recall that $\pi_{\approx_{\mathtt{x}_1 \ldots \mathtt{x}_n}}\colon \mathcal{T}(\{\mathtt{x}_1, \ldots, \mathtt{x}_n\} \cup \underline{A}) \to \mathcal{T}(\{\mathtt{x}_1, \ldots, \mathtt{x}_n\} \cup \underline{A})/{\approx_{\mathtt{x}_1 \ldots \mathtt{x}_n}}$ is defined as $\pi_{\approx_{\mathtt{x}_1 \ldots \mathtt{x}_n}}(s) = \langle s \rangle_n$ for any $s \in \mathcal{T}(\{\mathtt{x}_1, \ldots, \mathtt{x}_n\} \cup \underline{A})$).

And thus $\tilde{f}(\langle \mathtt{x}_i \rangle_n) = \tilde{f}(\pi_{\approx_{\mathtt{x}_1 \ldots \mathtt{x}_n}}(\mathtt{x}_i)) = f_b(\mathtt{x}_i) = b_i$ for any $i \leq n$. For $a \in \underline{A}$, we have $\tilde{f}(\eta_A^n(a)) = \tilde{f}(\langle a \rangle_n) = \tilde{f}(\pi_{\approx_{\mathtt{x}_1 \ldots \mathtt{x}_n}}(a)) = f_b(a) = f(a)$, and thus $\tilde{f} \circ \eta_A^n = f$. □

Note that the above proposition is similar to the substitution of variables, when f is a homomorphism from a combinatory algebra to itself.

4.2.2 The category of combinatory pre-models

Instead of adding several indeterminates at once, we can also consider adding them one by one. We outline this below.

Notation 4.2.3. For any combinatory pre-model $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{i}, \mathtt{e})$, we denote TA as the combinatory algebra $A[\mathtt{x}]$ for some variable $\mathtt{x}$. The set $\underline{TA}$ denotes $\underline{A[\mathtt{x}]} = \mathcal{T}(\{\mathtt{x}\} \cup \underline{A})/{\approx_{\mathtt{x}}}$. For any $n \in \mathbb{N}_{\geq 1}$, let T^nA be the n-time iteration of T over A. That is, $T^nA = (\cdots(A[\mathtt{x}])[\mathtt{x}]\cdots)[\mathtt{x}]$ for some variable $\mathtt{x}$. Let $T^0 = A$.

We can keep using the same variable $\mathtt{x}$ for the iteration of T^nA for some $n \in \mathbb{N}$, since with every iteration the equivalence classes are taken over the previous ones. For a variable $\mathtt{x}$, we will thus have the elements $\langle \mathtt{x} \rangle_1$, $\langle \langle \mathtt{x} \rangle_1 \rangle_1$, etc. The combinatory pre-models for a different choice of variable are isomorphic.

We show that we can interpret the T from the above notation as a functor on the *category of combinatory pre-models*, with combinatory pre-models objects and homomorphisms as morphisms.

Proposition 4.2.4. *Let A and B be two combinatory pre-models and let $\mathtt{x}$ be a variable. For their polynomial algebras $TA = A[\mathtt{x}]$ and $TB = B[\mathtt{x}]$, denote $\langle \cdot \rangle_1^A$ for the equivalence classes in $\underline{A[\mathtt{x}]}$, and similarly $\langle \cdot \rangle_1^B$ for the equivalence classes in $\underline{B[\mathtt{x}]}$. For any homomorphism $f\colon A \to B$, there exists a unique homomorphism $\overline{Tf}\colon TA \to TB$ such that $Tf \circ \eta_A^1 = \eta_B^1 \circ f$, and $Tf(\langle \mathtt{x} \rangle_1^A) = \langle \mathtt{x} \rangle_1^B$. That is, the following diagram commutes.*

$$\begin{array}{ccc} A & \xrightarrow{f} & B \\ {\scriptstyle \eta_A^1}\downarrow & & \downarrow{\scriptstyle \eta_B^1} \\ TA & \xrightarrow{Tf} & TB \end{array}$$

Proof. Let A, B and the homomorphism f be as in the statement of the lemma. We have that $\eta_B^1 \circ f$ is a homomorphism from A to TB. By Proposition 4.2.2, using $\eta_B^1 \circ f\colon A \to TB$ as the homomorphism in the assumption of the statement, we have that there exists a unique $Tf\colon TA \to TB$ such that $Tf \circ \eta_A^1 = \eta_B^1 \circ f$, and $Tf(\langle \mathtt{x} \rangle_1^A) = \langle \mathtt{x} \rangle_1^B$. □

It turns out that these two ways of achieving a polynomial algebra with n indeterminates are equivalent.

Proposition 4.2.5. *For any combinatory pre-model $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{i}, \mathtt{e})$ and for any $n \in \mathbb{N}_{\geq 1}$ the combinatory algebras $A[\mathtt{x}_1, \ldots, \mathtt{x}_n]$ and T^nA are isomorphic.*

Proof. The statement can be shown by induction on n. We will only show the case for $n = 2$.

Let A be a combinatory pre-model, and let $\mathtt{x}$ and $\mathtt{y}$ be distinct variables. Without loss of generality, we let $T^2A = (A[\mathtt{x}])[\mathtt{y}]$. To prove the statement, we will show the existence of two homomorphisms f and g such that $f \circ g = \mathrm{id}_{(A[\mathtt{x}])[\mathtt{y}]}$ and $g \circ f = \mathrm{id}_{A[\mathtt{x},\mathtt{y}]}$.

We define f by using Proposition 4.2.2. First, recall that $\eta_2\colon A \to A[\mathtt{x}, \mathtt{y}]$ is a homomorphism with $\eta_2(a) = \langle a \rangle_2$ for all $a \in \underline{A}$. Consider the polynomial algebra $(A[\mathtt{x}])[\mathtt{y}]$. Then $\eta_{TA}^1 \circ \eta_A^1\colon A \to (A[\mathtt{x}])[\mathtt{y}]$, with $(\eta_{TA}^1 \circ \eta_A^1)(a) = \langle \langle a \rangle_1 \rangle_1$ for all $a \in \underline{A}$.

By Proposition 4.2.2, we now have that there exists a unique $f\colon A[\mathtt{x},\mathtt{y}] \to (A[\mathtt{x}])[\mathtt{y}]$, such that $f(\langle\mathtt{x}\rangle_2) = \langle\langle\mathtt{x}\rangle_1\rangle_1$ and $f(\langle\mathtt{y}\rangle_2) = \langle\mathtt{y}\rangle_1$ and $f \circ \eta_2 = \eta^1_{TA} \circ \eta^1_A$. That is, the following diagram commutes.

$$\begin{array}{ccc} A & \xrightarrow{\eta^1_{TA}\circ\eta^1_A} & (A[\mathtt{x}])[\mathtt{y}] \\ {\scriptstyle\eta_2}\downarrow & \nearrow_{f} & \\ A[\mathtt{x},\mathtt{y}] & & \end{array}$$

We have $f(\langle a\rangle_2) = f(\eta_2(a)) = (\eta^1_{TA} \circ \eta^1_A)(a) = \langle\langle a\rangle_1\rangle_1$ for $a \in \underline{A}$. We show, by induction on the structure of terms in $\mathcal{T}(\{\mathtt{x}\} \cup \underline{A})$, that for all $s \in \mathcal{T}(\{\mathtt{x}\} \cup \underline{A})$

$$f(\langle s\rangle_2) = \langle\langle s\rangle_1\rangle_1. \tag{4.1}$$

We already have that $f(\langle\mathtt{x}\rangle_2) = \langle\langle\mathtt{x}\rangle_1\rangle_1$ and $f(\langle a\rangle_2) = \langle\langle a\rangle_1\rangle_1$ for $a \in \underline{A}$. Assume that for $s, t \in \mathcal{T}(\{\mathtt{x}\} \cup \underline{A})$, we have $f(\langle s\rangle_2) = \langle\langle s\rangle_1\rangle_1$ and similarly for t. Then,

$$\begin{aligned} f(\langle s,t\rangle_2) &= f(\langle s\rangle_2) *_1 f(\langle s\rangle_2) && \text{since } f \text{ is a homomorphism} \\ &= \langle\langle s\rangle_1\rangle_1 *_1 \langle\langle s\rangle_1\rangle_1 \\ &= \langle(\langle s\rangle_1, \langle t\rangle_1)\rangle_1 \\ &= \langle\langle(s,t)\rangle_1\rangle_1 && \text{by eq. (i) of Def. 4.1.3} \end{aligned}$$

And thus we have shown that equation (4.1) holds for all $s \in \mathcal{T}(\{\mathtt{x}\} \cup \underline{A})$.

We define $g\colon (A[\mathtt{x}])[\mathtt{y}] \to A[\mathtt{x},\mathtt{y}]$ inductively as follows.

1. $g(\langle\mathtt{y}\rangle_1) = \langle\mathtt{y}\rangle_2$,
2. $g(\langle\langle s\rangle_1\rangle_1) = \langle s\rangle_2$ for $\langle s\rangle_1 \in \underline{A[\mathtt{x}]}$,
3. $g(\langle(s,t)\rangle_1) = g(\langle s\rangle_1) *_2 g(\langle t\rangle_1)$ for s, $t \in \mathcal{T}(\{\mathtt{y}\} \cup \underline{A[\mathtt{x}]})$.

Now that we have obtained the homomorphisms f and g, we will show that they are each others inverse. We first show that $f(g(\langle s\rangle_1)) = \langle s\rangle_1$ for all $\langle s\rangle_1 \in \underline{(A[\mathtt{x}])[\mathtt{y}]}$. We use induction on $s \in \mathcal{T}(\{\mathtt{y}\} \cup \underline{A[\mathtt{x}]})$.

We have $f(g(\langle\mathtt{y}\rangle_1)) = f(\langle\mathtt{y}\rangle_2) = \langle\mathtt{y}\rangle_1$. For $\langle t\rangle_1 \in \underline{A[\mathtt{x}]}$, we have $f(g(\langle\langle t\rangle_1\rangle_1)) = f(\langle t\rangle_2) = \langle\langle t\rangle_1\rangle_1$, where the last equality follows from equation (4.1).

Since $f \circ g$ is a homomorphism, we can conclude that $f(g(\langle s\rangle_1)) = \langle s\rangle_1$ for all $\langle s\rangle_1 \in \underline{(A[\mathtt{x}])[\mathtt{y}]}$.

We now argue that also $g(f(\langle s\rangle_2)) = \langle s\rangle_2$ for all $\langle s\rangle_2 \in A[\mathtt{x},\mathtt{y}]$, by induction on $s \in \mathcal{T}(\{\mathtt{x},\mathtt{y}\} \cup \underline{A})$.

We have $g(f(\langle\mathtt{x}\rangle_2)) = g(\langle\langle\mathtt{x}\rangle_1\rangle_1) = \langle\mathtt{x}\rangle_2$, and $g(f(\langle\mathtt{y}\rangle_2)) = g(\langle\mathtt{y}\rangle_1) = \langle\mathtt{y}\rangle_2$. For $a \in \underline{A}$, we have $g(f(\langle a\rangle_2)) = g(\langle\langle a\rangle_1\rangle_1)$ by equation (4.1), and $g(\langle\langle a\rangle_1\rangle_1) = \langle a\rangle_2$.

Again, since $g \circ f$ is a homomorphism, we thus have $g(f(\langle s\rangle_2)) = \langle s\rangle_2$ for all $\langle s\rangle_2 \in A[\mathtt{x},\mathtt{y}]$. □

4.2.3 Infinitely many indeterminates

When we consider polynomial algebras over $\mathtt{X}$, we obtain similar results. Recall Proposition 4.2.2.

Proposition 4.2.6. *Let $A = (\underline{A}, \cdot_A, \mathtt{k}_A, \mathtt{s}_A, \mathtt{i}_A, \mathtt{e}_A)$ and $B = (\underline{B}, \cdot_B, \mathtt{k}_B, \mathtt{s}_B, \mathtt{i}_B, \mathtt{e}_B)$ be combinatory pre-models. For any homomorphism $f\colon A \to B$, and any (countably infinite) sequence $(b_n)_{n\geq 1}$ of elements in $\underline{B}$, there exists a unique homomorphism $\tilde{f}\colon A[\mathtt{X}] \to B$ such that $\tilde{f} \circ \sigma_A = f$ and $\tilde{f}(\langle \mathtt{x}_n \rangle_{\mathtt{X}}) = b_n$ for each $n \geq 1$.*

Proof. Analogously to the proof of Proposition 4.2.2. □

We will show how the polynomial algebras with finitely and infinitely many indeterminates are related. It is useful to first introduce a result that is analogous to Proposition 3.1.16.

Proposition 4.2.7. *Let $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s})$ be a combinatory algebra. For any $n \in \mathbb{N}$,and for any t, $s \in \mathcal{T}(\mathtt{X} \cup \underline{A})$,*

$$(\lambda^*\mathtt{x}.t, s) \approx_{\mathtt{X}} t[\mathtt{x}/s].$$

Proof. By induction on the structure of $t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$. □

Recall the definition of a *retract*.

Definition 4.2.8. For two objects X and Y in a category, X is a *retract* of Y when there exists morphisms $f\colon X \to Y$ and $g\colon Y \to X$ such that $g \circ f = \mathrm{id}_X$.

For the following proposition, recall Definition 4.2.8.

Proposition 4.2.9. *Let A be a combinatory pre-model, and let $\mathtt{x}$ be a variable. Then the following holds.*

1. *A and $A[\mathtt{x}_1, \ldots, \mathtt{x}_n]$ are retracts of $A[\mathtt{x}]$ for each $n \in \mathbb{N}_{\geq 1}$.*
2. *A and $A[\mathtt{x}_1, \ldots, \mathtt{x}_n]$ are retracts of $A[\mathtt{X}]$ for each $n \in \mathbb{N}_{\geq 1}$.*
3. *$(A[\mathtt{X}])[\mathtt{x}]$ is isomorphic to $A[\mathtt{X}]$.*

Proof. 1. Let A be a combinatory pre-model and let $\mathtt{x}$ be a variable. We first show that A is a retract of $A[\mathtt{x}]$. Then, using induction and the functor T (cf. Proposition 4.2.5), we can show that T^nA and thus $A[\mathtt{x}_1, \ldots, \mathtt{x}_n]$ is a retract of TA and thus $A[\mathtt{x}]$ for each $n \in \mathbb{N}_{\geq 1}$.

Consider Proposition 4.2.2. Let a be some element of $\underline{A}$. By the proposition, using the identity on A, we can conclude that there exists a unique homomorphism $\overline{\mathrm{id}_A}\colon A[\mathtt{x}] \to A$ such that $\overline{\mathrm{id}_A}(\langle \mathtt{x} \rangle_1) = a$ and $\overline{\mathrm{id}_A} \circ \eta^1_A = \mathrm{id}_A$. From the latter we can conclude that A is a retract of $A[\mathtt{x}]$ and thus also TA.

Now for the induction, assume that $\mathtt{x}$ and $\mathtt{y}$ are distinct variables. We start by showing that $A[\mathtt{x}, \mathtt{y}]$ and thus also T^2A is a retract of TA. We will show the existence of homomorphisms f and g such that $g \circ f = \mathrm{id}_{A[\mathtt{x},\mathtt{y}]}$. Define the following (cf. Section 6.2 in [7]).

$$\mathtt{t} = \mathtt{k}, \qquad \mathtt{f} = \lambda^*\mathtt{xy}.\mathtt{y}, \qquad \mathtt{pair} = \lambda^*\mathtt{xyz}.\mathtt{zxy}.$$

By Proposition 4.2.2, using $\eta^1_A \colon A \to A[\mathtt{x}]$ as the homomorphism in the assumption of the statement, we have that there exists a unique $f \colon A[\mathtt{x},\mathtt{y}] \to A[\mathtt{x}]$, such that $f(\langle \mathtt{x} \rangle_2) = \langle (\mathtt{x}, \mathtt{t}) \rangle_1$ and $f(\langle \mathtt{y} \rangle_2) = \langle (\mathtt{x}, \mathtt{f}) \rangle_1$ and $f \circ \eta_2 = \eta^1_A$. See below.

$$\begin{array}{ccc} A & \xrightarrow{\eta^1_A} & A[\mathtt{x}] \\ {\scriptstyle \eta_2}\downarrow & \nearrow_{f} & \\ A[\mathtt{x},\mathtt{y}] & & \end{array}$$

Again from Proposition 4.2.2, now using $\eta_2 \colon A \to A[\mathtt{x},\mathtt{y}]$ as the homomorphism in the assumption, we have that there exists a unique $g \colon A[\mathtt{x}] \to A[\mathtt{x},\mathtt{y}]$, such that $g(\langle \mathtt{x} \rangle_1) = \langle ((\mathtt{pair}, \mathtt{x}), \mathtt{y}) \rangle_2$ and $g \circ \eta^1_A = \eta_2$.

$$\begin{array}{ccc} A & \xrightarrow{\eta^1_A} & A[\mathtt{x},\mathtt{y}] \\ {\scriptstyle \eta_2}\downarrow & \nearrow_{g} & \\ A[\mathtt{x}] & & \end{array}$$

We argue that $g \circ f = \mathrm{id}_{A[\mathtt{x},\mathtt{y}]}$. First, we show that $g(f(\langle \mathtt{x} \rangle_2)) = \langle \mathtt{x} \rangle_2$ and $g(f(\langle \mathtt{y} \rangle_2)) = \langle \mathtt{y} \rangle_2$.

$$\begin{aligned}
g(f(\langle \mathtt{x} \rangle_2)) &= g(\langle (\mathtt{x}, \mathtt{t}) \rangle_1) \\
&= g(\langle \mathtt{x} \rangle_1) \cdot_2 g(\langle \mathtt{t} \rangle_1) \\
&= \langle ((\mathtt{pair}, \mathtt{x}), \mathtt{y}) \rangle_2 \cdot_2 g(\eta^1_A(\mathtt{t})) \\
&= \langle ((\mathtt{pair}, \mathtt{x}), \mathtt{y}) \rangle_2 \cdot_2 \eta_2(\mathtt{t}) \\
&= \langle ((\mathtt{pair}, \mathtt{x}), \mathtt{y}) \rangle_2 \cdot_2 \langle \mathtt{t} \rangle_2 \\
&= \langle (((\mathtt{pair}, \mathtt{x}), \mathtt{y}), \mathtt{t}) \rangle_2 \\
&= \langle \mathtt{x} \rangle_2 && \text{cf. Prop. 4.2.7}
\end{aligned}$$

$$\begin{aligned}
g(f(\langle \mathtt{y} \rangle_2)) &= g(\langle (\mathtt{x}, \mathtt{f}) \rangle_1) \\
&= \langle (((\mathtt{pair}, \mathtt{x}), \mathtt{y}), \mathtt{f}) \rangle_2 && \text{similarly as for the previous case} \\
&= \langle \mathtt{y} \rangle_2 && \text{cf. Prop. 4.2.7}
\end{aligned}$$

$$\begin{aligned}
g(f(\langle a \rangle_2)) &= g(f(\eta_2(a))) \\
&= g(\eta^1_A(a)) \\
&= \eta_2(a) \\
&= \langle a \rangle_2
\end{aligned}$$

Since $g \circ f$ is a homomorphism, we can conclude that $g(f(\langle s \rangle_2)) = \langle s \rangle_2$ for all $s \in \mathcal{T}(\{\mathtt{x},\mathtt{y}\} \cup \underline{A})$ and thus $g \circ f = \mathrm{id}_{A[\mathtt{x},\mathtt{y}]}$.

For the induction step, assume that for some $n \in \mathbb{N}$, we have that $T^n A$ is a retract of TA. Let $f \colon T^n A \to A$ and $g \colon A \to T^n A$ such that $g \circ f = \mathrm{id}_{T^n A}$.

When we substitute A by T^nA, from the fact that A is a retract of TA we obtain that T^nA is a retract of $T^{n+1}A$. Let $f'\colon T^{n+1}A \to T^nA$ and $g'\colon T^nA \to T^{n+1}A$ such that $g' \circ f' = \mathrm{id}_{T^{n+1}A}$. Then $(g' \circ g) \circ (f \circ f') = \mathrm{id}_{T^{n+1}A}$.

2. Let A be a combinatory pre-model, let $n \in \mathbb{N}$ and let $\mathtt{x}_1, \ldots, \mathtt{x}_n$ be distinct variables. Define a countably infinite sequence $(\mathtt{y}_n)_{n\geq 1}$ of elements in $\mathcal{T}(\{\mathtt{x}_1, \ldots, \mathtt{x}_n\} \cup \underline{A})$, where for all $i \leq n$ we have $\mathtt{y}_i = \mathtt{x}_i$, and for all $i > n$ we have $\mathtt{y}_i = \mathtt{x}_n$. We also have that $\eta^n_A\colon A \to A[\mathtt{x}_1, \ldots, \mathtt{x}_n]$, and thus, by Proposition 4.2.6, there exists a unique $f\colon A[\mathtt{X}] \to A[\mathtt{x}_1, \ldots, \mathtt{x}_n]$, such that $f \circ \sigma_A = \eta^n_A$ and $f(\langle \mathtt{x}_i \rangle_1) = \langle \mathtt{y}_i \rangle_n$ for all $i \in \mathbb{N}_{\geq 1}$.

On the other hand, since $\sigma_A\colon A \to A[\mathtt{X}]$, by Proposition 4.2.2 we have that there exists a uniqye $g\colon A[\mathtt{x}_1, \ldots, \mathtt{x}_n] \to A[\mathtt{X}]$, such that $g \circ \eta^n_A = \sigma_A$ and $g(\langle \mathtt{x}_i \rangle_n) = \langle \mathtt{x}_i \rangle_1$ for each $i \in \mathbb{N}$ with $i \leq n$.

Since $f \circ g$ is a homomorphism, from the above we can conclude, using induction on terms of $\mathcal{T}(\{\mathtt{x}_1, \ldots, \mathtt{x}_n\} \cup \underline{A})$, that for any $s \in \mathcal{T}(\{\mathtt{x}_1, \ldots, \mathtt{x}_n\} \cup \underline{A})$, we have that $f(g(\langle s \rangle_n)) = \langle s \rangle_n$, and thus $f \circ g = \mathrm{id}_{A[\mathtt{x}_1, \ldots, \mathtt{x}_n]}$.

3. Let A be a combinatory pre-model and let $\mathtt{x}$ be a variable. We show the existence of two homomorphisms f and h, such that $f \circ h = \mathrm{id}_{(A[\mathtt{X}])[\mathtt{x}]}$ and $h \circ f = \mathrm{id}_{A[\mathtt{X}]}$.

Note that $\eta^1_{A[\mathtt{X}]} \circ \sigma_A\colon A \to (A[\mathtt{X}])[\mathtt{x}]$, where $\eta^1_{A[\mathtt{X}]}\colon A[\mathtt{X}] \to (A[\mathtt{X}])[\mathtt{x}]$ such that $\eta^1_{A[\mathtt{X}]}(\langle s \rangle_{\mathtt{X}}) = \langle \langle s \rangle_{\mathtt{X}} \rangle_1$ for all $\langle s \rangle_{\mathtt{X}} \in \underline{A[\mathtt{X}]}$. From Proposition 4.2.2, using $\eta^1_{A[\mathtt{X}]} \circ \sigma_A$ as the homomorphism in the assumption of the statement, we obtain that there is a unique $f\colon A[\mathtt{X}] \to (A[\mathtt{X}])[\mathtt{x}]$, such that $f(\langle \mathtt{x}_1 \rangle_{\mathtt{X}}) = \langle \mathtt{x} \rangle_1$ and $f(\langle \mathtt{x}_i \rangle_{\mathtt{X}}) = \langle \langle \mathtt{x}_{i-1} \rangle_{\mathtt{X}} \rangle_1$ for $i \in \mathbb{N}$ with $i \leq 2$, and $f \circ \sigma_A = \eta^1_{A[\mathtt{X}]} \circ \sigma_A$.

From the homomorphism $\sigma_A\colon A \to A[\mathtt{X}]$ and Proposition 4.2.6, we obtain that there exists a unique $g\colon A[\mathtt{X}] \to A[\mathtt{X}]$ such that $g(\langle \mathtt{x}_i \rangle_{\mathtt{X}}) = \langle \mathtt{x}_{i+1} \rangle_{\mathtt{X}}$ for $i \in \mathbb{N}$ with $i \geq 1$, and $g \circ \sigma_A = \sigma_A$.

Using Proposition 4.2.2, where we use this g as the homomorphism in the assumption, we have that there exists a unique $h\colon (A[\mathtt{X}])[\mathtt{x}] \to A[\mathtt{X}]$, such that $h(\langle \mathtt{x} \rangle_1) = \langle \mathtt{x}_1 \rangle_{\mathtt{X}}$, and $h \circ \eta^1_{A[\mathtt{X}]} = g$.

We argue that $f \circ h = \mathrm{id}_{(A[\mathtt{X}])[\mathtt{x}]}$, by induction on $\mathcal{T}(\{\mathtt{x}\} \cup \underline{A[\mathtt{X}]})$. We do this by first showing that for all $\langle s \rangle_{\mathtt{X}} \in \underline{A[\mathtt{X}]}$, we have $f(h(\langle \langle s \rangle_{\mathtt{X}} \rangle_1)) = \langle \langle s \rangle_{\mathtt{X}} \rangle_1$, again using induction on terms. In the following, let $a \in \underline{A}$ and let $i \in \mathbb{N}_{\geq 1}$.

$$\begin{aligned} f(h(\langle \langle \mathtt{x}_i \rangle_{\mathtt{X}} \rangle_1)) &= f(h(\eta^1_{A[\mathtt{X}]}(\langle \mathtt{x}_i \rangle_{\mathtt{X}}))) \\ &= f(g(\langle \mathtt{x}_i \rangle_{\mathtt{X}})) \\ &= f(\langle \mathtt{x}_{i+1} \rangle_{\mathtt{X}}) \\ &= \langle \langle \mathtt{x}_i \rangle_{\mathtt{X}} \rangle_1 \end{aligned}$$

$$\begin{aligned} f(h(\langle \langle a \rangle_{\mathtt{X}} \rangle_1)) &= f(h(\eta^1_{A[\mathtt{X}]}(\langle a \rangle_{\mathtt{X}}))) \\ &= f(g(\langle a \rangle_{\mathtt{X}})) \\ &= f(g(\sigma_A(a))) \\ &= f(\sigma_A(a)) \\ &= \eta^1_{A[\mathtt{X}]}(\sigma_A(a)) \\ &= \langle \langle a \rangle_{\mathtt{X}} \rangle_1 \end{aligned}$$

Since $f \circ h$ is a homomorphism, we can conclude that $f(h(\langle \langle s \rangle_{\mathtt{X}} \rangle_1)) = \langle \langle s \rangle_{\mathtt{X}} \rangle_1$ for all $\langle s \rangle_{\mathtt{X}} \in \underline{A[\mathtt{X}]}$.

Also we have that $f(h(\langle \mathtt{x} \rangle_{\mathtt{X}})) = f(\langle \mathtt{x}_1 \rangle_{\mathtt{X}}) = \langle \mathtt{x} \rangle_1$. And thus, similarly as before, since $f \circ h$ is a homomorphism, we can conclude that $f(h(\langle s \rangle_1)) = \langle s \rangle_1$ for all $\langle s \rangle_1 \in \mathcal{T}(\{\mathtt{x}\} \cup \underline{A[\mathtt{X}]})$. Showing that $h \circ f = \mathrm{id}_{A[\mathtt{X}]}$ is done similarly. □

Remark 4.2.10. From Proposition 4.2.9.(2), for any combinatory pre-model A, we have $s \approx_{\mathtt{X}} t \Leftrightarrow s \approx_{\mathtt{x}_1 \ldots \mathtt{x}_n} t$ for any $n \in \mathbb{N}$ and any $s, t \in \mathcal{T}(\{\mathtt{x}_1, \ldots, \mathtt{x}_n\} \cup \underline{A})$.

4.2.4 An alternative representation of polynomial algebras

This section will be used for giving an alternative representation of polynomial algebras, without using indeterminates. By defining a specific relation and taking the quotient, we will define a specific combinatory pre-model and then show that it is isomorphic to the polynomial algebra.

Similar approaches have been outlined before in more specific settings, for example in Chapter 6, Section 3 of [27] or in Section 2.1 of [38]. We will say more about this in Remark 4.3.6 later.

We give the results in a general setting. Let $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{i}, \mathtt{e})$ be a combinatory pre-model, let $\mathtt{x}$ be a variable and let $A[\mathtt{x}]$ be the polynomial algebra of A in $\mathtt{x}$. We begin with the observation that a function $f : \underline{A} \to \underline{A[\mathtt{x}]}$ defined by $f(a) = \langle (a, \mathtt{x}) \rangle_1$ is surjective, since for each $\langle t \rangle_1 \in \underline{A[\mathtt{x}]}$ we have $\langle (\lambda^* \mathtt{x}.t, \mathtt{x}) \rangle_1 = \langle t \rangle_1$ (cf. Proposition 4.2.7).

Define an equivalence relation $\sim$ on $\underline{A}$ by $a \sim b$ iff $f(a) = f(b)$ for all $a, b \in \underline{A}$. Denote $[\cdot]$ for the equivalence classes of $\underline{A}/\!\sim$. Recall Definition 3.5.3. We define $\pi_\sim : \underline{A} \to \underline{A}/\!\sim$ by $\pi_\sim(a) = [a]$.

Similarly to Lemma 4.2.1, there is a unique $\tilde{f} : \underline{A}/\!\sim \to \underline{A[\mathtt{x}]}$ such that $\tilde{f} \circ \pi_\sim = f$, defined by $\tilde{f}([a]) = f(a)$ for all $a \in \underline{A}$. Similarly to f, the function $\tilde{f}$ is surjective. It is also injective by the definition of $\sim$.

We want to find an inverse function to $\tilde{f}$, since we are looking for a structure that is isomorphic to $A[\mathtt{x}]$. For this function, we will use the λ^*-abstraction. Note that we have $\langle (\lambda^* \mathtt{x}.t, \mathtt{x}) \rangle_1 = \langle t \rangle_1$ for any $t \in \mathcal{T}(\{\mathtt{x}\} \cup \underline{A})$, as mentioned before. However, taking the λ^*-abstraction still gives a term in $\mathcal{T}(\mathtt{X} \cup \underline{A})$. And thus we will use the interpretation as discussed in Remark 3.1.8. Then the inverse $\tilde{g}$ of $\tilde{f}$ can thus be defined by $\tilde{g}(\langle t \rangle_1) = [[\![\lambda^* \mathtt{x}.t]\!]^{\underline{A}}]$ for every $\langle t \rangle \in \underline{A[\mathtt{x}]}$.

This inverse function induces a combinatory pre-model

$$(\underline{A}/\!\sim, \diamond, [\mathtt{k}\mathtt{k}], [\mathtt{k}\mathtt{s}], [\mathtt{k}\mathtt{i}], [\mathtt{k}\mathtt{e}]),$$

where $[a] \diamond [b] = [\mathtt{s}ab]$.

We define this combinatory pre-model more formally, where we define the equivalence relation $\sim$ (later denoted as $\sim_1$) without using $A[\mathtt{x}]$.

Definition 4.2.11. Let $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{i}, \mathtt{e})$ be a combinatory pre-model. Define an applicative structure $A_1 = (\underline{A}, \cdot_1)$, where $a \cdot_1 b = \mathtt{s}ab$ for all $a, b \in \underline{A}$. For each $a \in \underline{A}$, define $a_1 = \mathtt{k}a$. Define $\sim_1$ as the smallest congruence relation on A_1 for which the following relations hold for all $a, b, c \in \underline{A}$.

i) $\mathtt{k}_1 \cdot_1 a \cdot_1 b \sim_1 a$,

ii) $\mathtt{s}_1 \cdot_1 a \cdot_1 b \cdot_1 c \sim_1 (a \cdot_1 c) \cdot_1 (b \cdot_1 c)$,

iii) $\mathtt{i}_1 \cdot_1 a \sim_1 a$,

iv) $\mathtt{e}_1 \cdot_1 a \cdot_1 b \sim_1 a \cdot_1 b$,

v) $(\mathtt{k}a) \cdot_1 (\mathtt{k}b) \sim_1 \mathtt{k}(ab)$,

vi) $(\mathtt{k}a) \cdot_1 \mathtt{i} \sim_1 a$.

Let $(\underline{A}/\!\sim_1, \diamond)$ be the quotient of A_1 by $\sim_1$. Denote $[\cdot]$ for the equivalence classes in $\underline{A}/\!\sim_1$.

Define $\bar{A}_1$ as the combinatory pre-model $(\underline{A}/\!\sim_1, \diamond, [\mathtt{k}_1], [\mathtt{s}_1], [\mathtt{i}_1], [\mathtt{e}_1])$.

It is straightforward to check that $\bar{A}_1$ is indeed a combinatory pre-model. The equations (v) and (vi) might seem strange at first, but their need will become clear when we consider the previously mentioned function $\tilde{g}$. Namely, equation (v) ensures that $\lambda^*\mathtt{x}.(a,b) = \lambda^*\mathtt{x}.ab$ for all $a, b \in \underline{A}$. Equation (vi) is needed for ensuring that $\tilde{g}$ is indeed the inverse of $\tilde{f}$, since it gives us that $\lambda^*\mathtt{x}.(a,\mathtt{x}) = a$ for any $a \in \underline{A}$. This will all be outlined in detail in the proof of the following theorem.

Theorem 4.2.12. *For any combinatory pre-model A and variable $\mathtt{x}$, the combinatory pre-models $\bar{A}_1$ and $A[\mathtt{x}]$ are isomorphic.*

Proof. Let $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{i}, \mathtt{e})$ be a combinatory pre-model. We will give two functions $\tilde{f}$ and $\tilde{g}$ and then argue that $\tilde{f} \circ \tilde{g} = \mathrm{id}_{A[\mathtt{x}]}$ and $\tilde{g} \circ \tilde{f} = \mathrm{id}_{\bar{A}_1}$.

Let $(\mathcal{T}(\{\mathtt{x}\} \cup \underline{A}), \bullet)$ be an applicative structure where for all s, $t \in \mathcal{T}(\{\mathtt{x}\} \cup \underline{A})$ we have $s \bullet t = (s,t)$.

We define a function $f : A_1 \to ((\mathcal{T}(\{\mathtt{x}\} \cup \underline{A}), *_1)$ as $f(a) = \langle (a, \mathtt{x}) \rangle_1$ for all $a \in \underline{A}$. Then f is a homomorphism, since for all $a, b \in \underline{A}$, we have

$$\begin{aligned}
f(a \cdot_1 b) &= f(\mathtt{s}ab) \\
&= \langle (\mathtt{s}ab, \mathtt{x}) \rangle_1 \\
&= \langle (((\mathtt{s}, a), b), \mathtt{x}) \rangle_1 && \text{by Def. 4.1.3 eq. (i)} \\
&= \langle ((a, \mathtt{x}), (b, \mathtt{x})) \rangle_1 && \text{by Def. 4.1.3 eq. (iii)} \\
&= \langle (a, \mathtt{x}) \rangle_1 *_1 \langle (b, \mathtt{x})) \rangle_1 \\
&= g(a) *_1 g(b)
\end{aligned}$$

By induction on how $\sim_1$ is defined, we will show that $a \sim_1 b \Rightarrow f(a) = f(b)$ for all a, $b \in \underline{A}$. By Lemma 4.2.1 we will then have that there exists a homomorphism $\tilde{f} \colon (\underline{A}/\!\sim_1, \diamond) \to ((\mathcal{T}(\{\mathtt{x}\} \cup \underline{A}), *_1)$. We only show the case for Definition 4.2.11 equation (i). The rest follows similarly. In the following, let $a, b \in \underline{A}$.

$$\begin{aligned}
f(\mathtt{k}_1 \cdot_1 a \cdot_1 b) &= f(\mathtt{s}(\mathtt{s}(\mathtt{kk})a)b) \\
&= \langle (\mathtt{s}(\mathtt{s}(\mathtt{kk})a)b, \mathtt{x}) \rangle_1 \\
&= \langle ((((\mathtt{s}, ((\mathtt{s}, (\mathtt{kk})), a)), b), \mathtt{x}) \rangle_1 && \text{by Def. 4.1.3 eq. (i)} \\
&= \langle (a, \mathtt{x}) \rangle_1 && \text{by Def. 4.1.3 eq. (ii) and (iii)} \\
&= f(a)
\end{aligned}$$

And thus, by Lemma 4.2.1, there exists a unique $\tilde{f} : (\underline{A}/\sim_1, \diamond) \to ((\mathcal{T}(\{\mathtt{x}\} \cup \underline{A}), *_1)$, such that $\tilde{f} \circ \pi_{\sim_1} = f$. Then $\tilde{f}$ extends naturally to a homomorphism $\bar{\tilde{f}} : \bar{A}_1 \to A[\mathtt{x}]$, with

$$\tilde{f}([a]) = \langle (a, \mathtt{x}) \rangle_1$$

for all $[a] \in \underline{A}/\sim_1$, since from $\tilde{f} \circ \pi_{\sim_1} = f$ we obtain that the combinators of both combinatory pre-models get mapped to each other.

We define a homomorphism g, which will extend similarly into a homomorphism $\tilde{g}$, and then show that $\tilde{g}$ is the inverse of f.

As argued before, we define g as $g(t) = [\![\lambda^* \mathtt{x}.t]\!]$ for all $t \in \mathcal{T}(\{\mathtt{x}\} \cup \underline{A})$.

Note that for any $a, b \in \mathcal{T}(\underline{A})$, we have

$$[\![(a, b)]\!]^{\underline{A}} = [\![a]\!]^{\underline{A}} \cdot [\![b]\!]^{\underline{A}} \tag{4.2}$$

And thus g is a homomorphism, since for all s, $t \ \in \mathcal{T}(\{\mathtt{x}\} \cup \underline{A})$,

$$\begin{aligned}
g((s,t)) &= [[\![\lambda^* \mathtt{x}.(s,t)]\!]^{\underline{A}}] \\
&= [[\![((s, (\lambda^* \mathtt{x}.s)), (\lambda^* \mathtt{x}.t))]\!]^{\underline{A}}] \\
&= [[\![(\lambda^* \mathtt{x}.s)]\!]^{\underline{A}} \cdot_1 [\![(\lambda^* \mathtt{x}.t)]\!]^{\underline{A}}] && \text{by eq. (4.2)} \\
&= [[\![\lambda^* \mathtt{x}.s]\!]^{\underline{A}}] \diamond [[\![\lambda^* \mathtt{x}.t]\!]^{\underline{A}}] \\
&= g(s) \diamond g(t)
\end{aligned}$$

We again show that $t \approx_{\mathtt{x}} s \Rightarrow g(t) = g(s)$ for all t, $s \in \mathcal{T}(\{\mathtt{x}\} \cup \underline{A})$, for the purpose of using Lemma 4.2.1.

Recall Definition 4.1.3. In the following, let a, $b \in \underline{A}$ and let s, $t \in \mathcal{T}(\{\mathtt{x}\} \cup \underline{A})$.

$$\begin{aligned}
g((a,b)) &= [[\![\lambda^* \mathtt{x}.(a,b)]\!]^{\underline{A}}] \\
&= [[\![s(\lambda^* \mathtt{x}.a)(\lambda^* \mathtt{x}.b)]\!]^{\underline{A}}] \\
&= [[\![\lambda^* \mathtt{x}.a]\!]^{\underline{A}}] \diamond [[\![\lambda^* \mathtt{x}.b]\!]^{\underline{A}}] \\
&= [\mathtt{k}a] \diamond [\mathtt{k}b] \\
&= [(\mathtt{k}a) \cdot_1 (\mathtt{k}b)] \\
&= [\mathtt{k}(ab)] && \text{by Def. 4.2.11 eq. (v)}
\end{aligned}$$

$$\begin{aligned}
g(((\mathtt{k}, s), t)) &= [[\![\lambda^* \mathtt{x}.((\mathtt{k}, s), t)]\!]^{\underline{A}}] \\
&= ([[\![\lambda^* \mathtt{x}.\mathtt{k}]\!]^{\underline{A}}] \diamond [[\![\lambda^* \mathtt{x}.s]\!]^{\underline{A}}]) \diamond [\lambda^* \mathtt{x}.t]\!]^{\underline{A}}] && \text{similarly as before} \\
&= ([\mathtt{k}_1] \diamond [[\![\lambda^* \mathtt{x}.s]\!]^{\underline{A}}]) \diamond [\lambda^* \mathtt{x}.t]\!]^{\underline{A}}] \\
&= [\mathtt{k}_1] \\
&= g(\mathtt{k})
\end{aligned}$$

The equations for `s`, `i` and `e` follow similarly to the one for `k`.

And thus, $t \approx_{\mathtt{x}} s \Rightarrow g(t) = g(s)$ for all t, $s \in \mathcal{T}(\{\mathtt{x}\} \cup \underline{A})$, and by Lemma 4.2.1 we then get a homomorphism $\tilde{g} : (\mathcal{T}(\{\mathtt{x}\} \cup \underline{A})/\approx_{\mathtt{x}}, *_1) \to (\underline{A}/\sim, *_1)$ such that $\tilde{g} \circ \pi_{\approx_x} = g$. We thus have that

$$\tilde{g}(\langle t \rangle_1) = [[\![\lambda^* \mathtt{x}.t]\!]^{\underline{A}}]$$

for all $\langle t\rangle_1 \in \underline{A[\mathtt{x}]}$. This again naturally extends to a homomorphism $\tilde{g} : A[\mathtt{x}] \to \bar{A}_1$.

We now argue that $\tilde{f}$ and $\tilde{g}$ are each others inverses. Let $\langle t\rangle_1 \in \underline{A[\mathtt{x}]}$. Then,

$$\begin{aligned}
\tilde{f}(\tilde{g}(\langle t\rangle_1)) &= \tilde{f}([[\![\lambda^*\mathtt{x}.t]\!]^{\underline{A}}]) \\
&= \langle([[\![\lambda^*\mathtt{x}.t]\!]^{\underline{A}}, \mathtt{x})\rangle_1 \\
&= \langle(\lambda^*\mathtt{x}.t, \mathtt{x})\rangle_1 && \text{by Def. 4.1.3 eq. (i)} \\
&= \langle t\rangle_1 && \text{by Prop. 3.1.16}
\end{aligned}$$

For any $[a] \in \underline{A}/\!\sim_1$, we have

$$\begin{aligned}
\tilde{g}(\tilde{f}([a])) &= \tilde{g}(\langle(a, \mathtt{x})\rangle_1) \\
&= [[\![\lambda^*\mathtt{x}.(a, \mathtt{x})]\!]^{\underline{A}}] \\
&= [\mathtt{s}(\mathtt{k}a)(\mathtt{i})] \\
&= [(\mathtt{k}a) \cdot_1 \mathtt{i}] \\
&= [a] && \text{by Def. 4.2.11 eq. (vi)}
\end{aligned}$$

□

From the proof of Theorem 4.2.12, we can derive the following correspondence.

Proposition 4.2.13. *For any combinatory pre-model $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{i}, \mathtt{e})$ and any variable $\mathtt{x}$, we have*

$$a \sim_1 b \Leftrightarrow (a, \mathtt{x}) \approx_{\mathtt{x}} (b, \mathtt{x})$$

for all a, $b \in \underline{A}$.

The following Corollary is an immediate consequence to this proposition.

Corollary 4.2.14. *For any combinatory pre-model $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{i}, \mathtt{e})$, we have*

$$\mathtt{e}a \sim_1 a$$

for all $a \in \underline{A}$.

4.3 Reflexivity

4.3.1 Different characterizations

We introduce an algebraic analogue of the Meyer-Scott axiom (cf. Definition 3.3.1). We will see later (Proposition 4.3.13 and Theorem 4.4.2) how this relates to different forms of weak extensionality.

Definition 4.3.1. A combinatory pre-model $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{i}, \mathtt{e})$ is *reflexive* if

$$a \sim_1 b \Rightarrow \mathtt{e}a = \mathtt{e}b$$

for all a, $b \in \underline{A}$.

From Proposition 4.2.13 we obtain an equivalent definition.

Corollary 4.3.2. *A combinatory pre-model* $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{i}, \mathtt{e})$ *is reflexive if and only if*

$$(a, \mathtt{x}) \approx_{\mathtt{x}} (b, \mathtt{x}) \Rightarrow \mathtt{e}a = \mathtt{e}b$$

for any variable $\mathtt{x}$ *and* a, $b \in \underline{A}$.

It turns out that for any combinatory pre-model $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{i}, \mathtt{e})$ we can characterize the reflexivity of A by a set of simple universal sentences on the elements of $\underline{A}$, because the relation $\sim_1$ is generated from equations on the elements of $\underline{A}$ (cf. Definition 4.2.11). We proceed as follows. For a combinatory pre-model $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{i}, \mathtt{e})$, we define a function $f\colon \underline{A} \to \underline{A}$ as $f(a) = \mathtt{e}a$ for $a \in \underline{A}$. We want to extend this to a homomorphism between applicative structures, and see under what condition this f preserves the relation $\sim_1$ (that is, when $a \sim_1 b \Rightarrow f(a) = f(b)$ for $a, b \in \underline{A}$). Note that f maps to the set $\{\mathtt{e}a : a \in \underline{A}\}$. We therefore first define the following.

Definition 4.3.3. Let $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{i}, \mathtt{e})$ be a combinatory pre-model. Let

$$\mathtt{e}A = \{\mathtt{e}a : a \in \underline{A}\}.$$

Define an applicative structure $(\mathtt{e}A, \cdot_{\mathtt{e}})$ where

$$\mathtt{e}a \cdot_{\mathtt{e}} \mathtt{e}b = \mathtt{e}(\mathtt{e}a \cdot_1 \mathtt{e}b)$$

for all $\mathtt{e}a, \mathtt{e}b \in \mathtt{e}A$.

Note that $\cdot_{\mathtt{e}}$ is indeed an operation on $\mathtt{e}A$.

To find the necessary equations, we simply give the appropriate counterparts of the conditions of Definition 4.2.11. However, we also required that $\sim_1$ is a congruence relation (cf. Definition 3.5.2). And thus, for some combinatory pre-model $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{i}, \mathtt{e})$, we need that

$$\mathtt{e}a = \mathtt{e}b \wedge \mathtt{e}c = \mathtt{e}d \Rightarrow \mathtt{e}(a \cdot_1 b) = \mathtt{e}(b \cdot_1 d)$$

for all a, b, c, $d \in \underline{A}$. Since $\cdot_{\mathtt{e}}$ is an operation on $\mathtt{e}A$, we have that $\mathtt{e}a = \mathtt{e}b$ and $\mathtt{e}c = \mathtt{e}d$ implies that $\mathtt{e}a \cdot_{\mathtt{e}} \mathtt{e}c = \mathtt{e}b \cdot_{\mathtt{e}} \mathtt{e}d$ for all a, b, c, $d \in \underline{A}$. And thus it is sufficient to require that $\mathtt{e}a \cdot_{\mathtt{e}} \mathtt{e}b = \mathtt{e}(a \cdot_1 b)$ for all a, $b \in \underline{A}$. Note that this ensures that the defined f is a homomorphism between applicative structures. We arrive at the following proposition.

Proposition 4.3.4. *A combinatory pre-model* $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{i}, \mathtt{e})$ *is reflexive if and only of it satisfies the following equations for each* a, b, $c \in \underline{A}$.

1. $\mathtt{e}(\mathtt{k}_1 \cdot_1 a \cdot_1 b) = \mathtt{e}a$,
2. $\mathtt{e}(\mathtt{s}_1 \cdot_1 a \cdot_1 b \cdot_1 c) = \mathtt{e}(a \cdot_1 c \cdot_1 (b \cdot_1 c))$,
3. $\mathtt{e}(\mathtt{i}_1 \cdot_1 a) = \mathtt{e}a$,
4. $\mathtt{e}(\mathtt{e}_1 \cdot_1 a \cdot_1 b) = \mathtt{e}(a \cdot_1 b)$,

5. $\mathtt{e}((\mathtt{k}a) \cdot_1 (\mathtt{k}b)) = \mathtt{e}(\mathtt{k}(ab))$,

6. $\mathtt{e}((\mathtt{k}a) \cdot_1 \mathtt{i}) = \mathtt{e}a$,

7. $\mathtt{e}((\mathtt{e}a) \cdot_1 (\mathtt{e}b)) = \mathtt{e}(a \cdot_1 b)$.

Proof. We argued before that when an applicative structure A as in the statement satisfies the equations, then A is reflexive.

Now assume that A is reflexive. From Definition 4.2.11, it is clear that the equations (1)- (6) hold. From Corollary 4.2.14, we have $\mathtt{e}a \cdot_1 \mathtt{e}b \sim_1 a \cdot_1 b$ for all $a, b \in \underline{A}$. This implies equation (7). □

Let $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{i}, \mathtt{e})$ be a combinatory pre-model. We go back to the homomorphism $f : (\underline{A}, \cdot_1) \to (\mathtt{e}A, \cdot_\mathtt{e})$ defined earlier. By Lemma 4.2.1, this extends to a unique homomorphism $\tilde{f} : (\underline{A}/\sim_1, \cdot_1) \to (\mathtt{e}A, \cdot_\mathtt{e})$, and in turn this naturally extends to a homomorphism between combinatory pre-models $\bar{A}_1$ (Definition 4.2.11) and $(\mathtt{e}A, \cdot_\mathtt{e}, \mathtt{ek}_1, \mathtt{es}_1, \mathtt{ei}_1, \mathtt{ee}_1)$.

From the proof of Proposition 4.3.4 we thus have the following.

Corollary 4.3.5. *Let $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{i}, \mathtt{e})$ be a reflexive combinatory pre-model. Let $A^\mathtt{e} = (\mathtt{e}A, \bullet_\mathtt{e}, \mathtt{ek}_1, \mathtt{es}_1, \mathtt{ei}_1, \mathtt{ee}_1)$, where $\mathtt{e}a \bullet_\mathtt{e} \mathtt{e}b = \mathtt{e}(a \cdot_1 b)$ for all $a, b \in \underline{A}$. Then $A^\mathtt{e}$ is isomorphic to $\bar{A}_1$ and $A[\mathtt{x}]$.*

Proof. Since A is reflexive, by Corollary 4.2.14, we have that $\mathtt{e}A = \{a \in \underline{A} : a = \mathtt{e}a\}$. Then $\bullet_\mathtt{e}$ is a well-defined operation on $\mathtt{e}A$, and the isomorphisms follow from the arguments before Proposition 4.3.4 and its proof, and from Theorem 4.2.12. □

Remark 4.3.6. In the literature we can find constructions similar to $A^\mathtt{e}$ [27, 17, 38]. For example, in [27, Ch. 3 Sec. 6], Krivine deduces some of Curry's axioms (cf. Theorem 3.2.6) by considering a combinatory algebra $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s})$ and an applicative structure $B = (\mathbf{1}A, \diamond, \mathtt{kk}, \mathtt{ks})$, where $\mathbf{1}A = \{\mathbf{1}a : a \in \underline{A}\}$ and $a \diamond b = \mathtt{s}ab$ for all $a, b \in \mathbf{1}A$ (recall that $\mathbf{1} = \mathtt{skk}$). For this purpose he assumed a weak form of stability (cf. 3.4.4):

$$\forall a, b \in \underline{A}[\mathtt{e}(\mathtt{k}a) = \mathtt{k}a \wedge \mathtt{e}(\mathtt{s}ab) = \mathtt{s}ab].$$

He then argued that what is needed further for an isomorphism between B and A are the equations (1), (2) and (5) of Proposition 4.3.4, only without $\mathtt{e}$ in front of both sides. Selinger showed in [38, Prop. 4] that for a lambda algebra $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s})$ and a variable $\mathtt{x}$, the structures B and $A[\mathtt{x}]$ are isomorphic, where stability was assumed since lambda algebras are stable (cf. Remark 3.4.8). Our result gives thus gives a more general result, when including $\mathtt{i}$ and $\mathtt{e}$ and not assuming any form of stability.

We state some more properties of the notion of reflexivity.

The seven equations of Proposition 4.3.4 are all universal sentences. Therefore we have the following corollary, where a *substructure* of a combinatory pre-model $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{i}, \mathtt{e})$ is a subset $B \subseteq \underline{A}$, which is closed under $\cdot$ and contains $\mathtt{k}$, $\mathtt{s}$, $\mathtt{i}$ and $\mathtt{e}$.

Corollary 4.3.7. *Reflexivity is closed under substructures and homomorphic images. In particular, it is closed under retracts.*

This entails that reflexivity is also preserved under adding more indeterminates to polynomial algebras.

Proposition 4.3.8. *For any combinatory pre-model A, any $n \in \mathbb{N}_{\geq 1}$ and any variable $\mathtt{x}$, if $A[\mathtt{x}]$ is reflexive, then also $A[\mathtt{x}_1, \ldots, \mathtt{x}_n]$ and A.*

Proof. From Proposition 4.2.9 and Corollary 4.3.7. □

Note that it is not necessarily the case for a combinatory pre-model A and a variable $\mathtt{x}$, that $A[\mathtt{x}]$ is reflexive when A is.

4.3.2 An alternative lambda abstraction

Next, we introduce an alternative lambda abstraction mechanism, different from λ^* (cf. Definition 4.1.2). Using the new abstraction mechanism, we can characterize reflexivity by $A[\mathtt{x}]$ being preserved by this abstraction.

Definition 4.3.9. Let $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{i}, \mathtt{e})$ be a combinatory pre-model. For any $t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$ and any variable $\mathtt{x}$, define the lambda abstraction $\lambda^\dagger \mathtt{x}.t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$ inductively as follows:

i) $\lambda^\dagger \mathtt{x}.\mathtt{x} \equiv (\mathtt{e}, \mathtt{i})$.

ii) $\lambda^\dagger \mathtt{x}.a \equiv (\mathtt{e}, (\mathtt{k}, t))$ for any $a \in (\mathtt{X} \cup \underline{A})$ with $a \not\equiv \mathtt{x}$.

iii) $\lambda^\dagger \mathtt{x}.(a, \mathtt{x}) \equiv (\mathtt{e}, a)$ for any $a \in (\mathtt{X} \cup \underline{A})$ with $a \not\equiv \mathtt{x}$.

iv) $\lambda^\dagger \mathtt{x}.(t_1, t_2) \equiv (\mathtt{e}, ((\mathtt{s}, \lambda^\dagger \mathtt{x}.t_1), \lambda^\dagger \mathtt{x}.t_2))$ if $t_1 \equiv \mathtt{x}$ or $t_1 \notin (\mathtt{X} \cup \underline{A})$ or $t_2 \not\equiv \mathtt{x}$.

For any $n \in \mathbb{N}$ and variables $\mathtt{x}_1, \ldots, \mathtt{x}_n$, define the repeated lambda abstraction $\lambda^\dagger \mathtt{x}_1 \cdots \mathtt{x}_n.t$ as $\lambda^\dagger \mathtt{x}_1.(\cdots(\lambda^\dagger \mathtt{x}_n.t)\cdots)$.

Remark 4.3.10. Similarly to Remark 3.1.15, we have that for some $n \in \mathbb{N}_{\geq 1}$, if $t \in \mathcal{T}(\{\mathtt{x}_1, \ldots, \mathtt{x}_n\} \cup \underline{A})$ and $i \in \{1, \ldots, n\}$, then $\lambda^\dagger \mathtt{x}_i.t \in \mathcal{T}(\{\mathtt{x}_1, \ldots, \mathtt{x}_n\} \setminus \{\mathtt{x}_i\} \cup \underline{A})$.

Recall Definition 3.1.5 and Proposition 3.1.16.

Proposition 4.3.11. *Let $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{i}, \mathtt{e})$ be a combinatory pre-model. For any variable $\mathtt{x}$ and for any t, $s \in \mathcal{T}(\mathtt{X} \cup \underline{A})$,*

$$(\lambda^\dagger \mathtt{x}.t, s) \approx_{\mathtt{x}} t[\mathtt{x}/s].$$

Proof. Similarly to Proposition 4.2.7, this follows easily from induction on the structure of $t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$. □

We can see that in the sense of Proposition 4.2.7 and Proposition 4.3.11, the abstractions λ^* and $\lambda^\dagger$ are similar. Nevertheless, the $\lambda^\dagger$-abstraction satisfies some properties that the λ^*-abstraction does not need to satisfy. An example is the following lemma, which will turn out to be of importance later on.

Lemma 4.3.12. *Let $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{i}, \mathtt{e})$ be a reflexive combinatory pre-model and let $\mathtt{x}$ be a variable. Then*

$$\mathtt{e}[\![\lambda^\dagger \mathtt{x}.t]\!]^{\underline{A}} = [\![\lambda^\dagger \mathtt{x}.t]\!]^{\underline{A}}$$

for each $t \in \mathcal{T}(\{\mathtt{x}\} \cup \underline{A})$.

Proof. Let A and $\mathtt{x}$ be as in the statement. Since A is reflexive, from Corollary 4.2.14, we have that $(\mathtt{e}, (\mathtt{e}, a)) = (\mathtt{e}, a)$ for all $a \in \underline{A}$. The statement then follows from induction on $t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$, using Definition 4.3.9. □

We can now use $\lambda^\dagger$-abstraction to characterize reflexivity.

Proposition 4.3.13. *Let A be a combinatory pre-model and let $\mathtt{x}$ be a variable. Then A is reflexive if and only if*

$$s \approx_{\mathtt{x}} t \Rightarrow [\![\lambda^\dagger \mathtt{x}.s]\!] = [\![\lambda^\dagger \mathtt{x}.t]\!]$$

for all s, $t \in \mathcal{T}(\{\mathtt{x}\} \cup \underline{A})$

Proof. Let A and $\mathtt{x}$ be as in the statement of the proposition. First, suppose that A is reflexive. Let s, $t \in \mathcal{T}(\{\mathtt{x}\} \cup \underline{A})$ and suppose that $s \approx_{\mathtt{x}} t$. From Proposition 4.3.11, Remark 4.2.10 and equation (i) of Definition 4.1.3, we have that $([\![\lambda^\dagger \mathtt{x}.s]\!], \mathtt{x}) \approx_{\mathtt{x}} (\lambda^\dagger \mathtt{x}.s, \mathtt{x}) \approx_{\mathtt{x}} s[\mathtt{x}/\mathtt{x}] \approx_{\mathtt{x}} s \approx_{\mathtt{x}} t \approx_{\mathtt{x}} (\lambda^\dagger \mathtt{x}.t, \mathtt{x}) \approx_{\mathtt{x}} ([\![\lambda^\dagger \mathtt{x}.t]\!], \mathtt{x})$. Since A is reflexive, from Corollary 4.3.2, we obtain that $\mathtt{e}[\![\lambda^\dagger \mathtt{x}.s]\!] = \mathtt{e}[\![\lambda^\dagger \mathtt{x}.t]\!]$. Then Lemma 4.3.12 gives us that $[\![\lambda^\dagger \mathtt{x}.s]\!] = [\![\lambda^\dagger \mathtt{x}.t]\!]$.

Now suppose that $s \approx_{\mathtt{x}} t \Rightarrow [\![\lambda^\dagger \mathtt{x}.s]\!] = [\![\lambda^\dagger \mathtt{x}.t]\!]$ for all s, $t \in \mathcal{T}(\{\mathtt{x}\} \cup \underline{A})$. We will use Corollary 4.3.2 to argue that A is reflexive, namely, by showing that $(a, \mathtt{x}) \approx_{\mathtt{x}} (b, \mathtt{x}) \Rightarrow \mathtt{e}a = \mathtt{e}b$ for any a, $b \in \underline{A}$. Assume that $(a, \mathtt{x}) \approx_{\mathtt{x}} (b, \mathtt{x})$ for some a, $b \in \underline{A}$. Then by assumption $[\![\lambda^\dagger \mathtt{x}.(a, \mathtt{x})]\!] = [\![\lambda^\dagger \mathtt{x}.(b, \mathtt{x})]\!]$. From Definition 4.3.9.iii), we then obtain $\mathtt{e}a = [\![(\mathtt{e}, a)]\!] = [\![(\mathtt{e}, b)]\!] = \mathtt{e}b$. □

Thus, reflexivity can be described as the requirement that the mapping $t \mapsto \lambda^\dagger \mathtt{x}.t$ is well-defined for polynomials with one indeterminate. Or, using similar terminology as before, that equality on polynomials with one indeterminate respects the ξ-rule of the lambda calculus.

If we assume the polynomial algebra with n indeterminates to be reflexive, we get that the mapping $t \mapsto \lambda^\dagger \mathtt{x}.t$ is well-defined for polynomials with n indeterminates. This is formulated in the following proposition, where we use the isomorphism of a polynomial algebra $A[\mathtt{x}_1, \ldots, \mathtt{x}_n]$ with $T^n A$ (cf. Proposition 4.2.5) to give an alternative formulation of Proposition 4.3.13 and Corollary 4.3.2.

Proposition 4.3.14. *For any combinatory pre-model $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{i}, \mathtt{e})$ and any $n \in \mathbb{N}_{\geq 1}$, the following are equivalent.*

1. *$A[\mathtt{x}_1, \ldots, \mathtt{x}_n]$ is reflexive.*
2. *For all s, $t \in \mathcal{T}(\{\mathtt{x}_1, \ldots, \mathtt{x}_{n+1}\} \cup \underline{A})$,*

$$s \approx_{\mathtt{x}_1 \ldots \mathtt{x}_{n+1}} t \Rightarrow \lambda^\dagger \mathtt{x}_{n+1}.s \approx_{\mathtt{x}_1 \ldots \mathtt{x}_n} \lambda^\dagger \mathtt{x}_{n+1}.t.$$

3. For all $s, t \in \mathcal{T}(\{\mathtt{x}_1, \ldots, \mathtt{x}_n\} \cup \underline{A})$,

$$(s, \mathtt{x}_{n+1}) \approx_{\mathtt{x}_1 \ldots \mathtt{x}_{n+1}} (t, \mathtt{x}_{n+1}) \Rightarrow (\mathtt{e}, s) \approx_{\mathtt{x}_1 \ldots \mathtt{x}_n} (\mathtt{e}, t).$$

Proof. Let A be a combinatory pre-model as in the statement of the lemma, let $n \in \mathbb{N}$, and let $\mathtt{x}_1$, ..., $\mathtt{x}_{n+1}$ be distinct variables. From Proposition 4.2.5, we have that there exists an isomorphism $f\colon A[\mathtt{x}_1, \ldots, \mathtt{x}_n] \to T^n A$. Define a homomorphism $\varphi\colon \mathcal{T}(\{\mathtt{x}_1, \ldots, \mathtt{x}_{n+1}\} \cup \underline{A}) \to \mathcal{T}(\{\mathtt{x}\} \cup \underline{T^n A})$ inductively as follows.

1. $\varphi(\mathtt{x}_{n+1}) = \mathtt{x}$,
2. $\varphi(s) = f(s)$ for $s \in \mathcal{T}(\{\mathtt{x}_1, \ldots, \mathtt{x}_n\} \cup \underline{A})$,
3. $\varphi((s,t)) = (\varphi(s), \varphi(t))$ for $s, t \in \mathcal{T}(\{\mathtt{x}_1, \ldots, \mathtt{x}_{n+1}\} \cup \underline{A})$.

Note that φ is surjective. We claim that for all $t \in \mathcal{T}(\{\mathtt{x}_1, \ldots, \mathtt{x}_{n+1}\} \cup \underline{A})$, we have

$$f(\lambda^\dagger \mathtt{x}_{n+1}.t) = \lambda^\dagger \mathtt{x}.\varphi(t). \tag{4.3}$$

We do this by induction on the structure of $t \in \mathcal{T}(\{\mathtt{x}_1, \ldots, \mathtt{x}_{n+1}\} \cup \underline{A})$, following Definition 4.3.9. Note that, since f is an isomorphism, we have that $T^n A$ is the combinatory pre-model $(\underline{T^n A}, *_1, f(\mathtt{k}), f(\mathtt{s}).f(\mathtt{i}), f(\mathtt{e}))$.

First assume that $t \equiv \mathtt{x}_{n+1}$. Then $f(\lambda^\dagger \mathtt{x}_{n+1}.t) = (f(\mathtt{e}), f(\mathtt{i})) = \lambda^\dagger \mathtt{x}.\mathtt{x} = \lambda^\dagger \mathtt{x}.\varphi(t)$.

If $t \in (\mathtt{X} \cup \underline{A})$ with $t \not\equiv \mathtt{x}_{n+1}$, then we have $f(\lambda^\dagger \mathtt{x}_{n+1}.t) = (f(\mathtt{e}), (f(\mathtt{k}), f(t))) = (f(\mathtt{e}), (f(\mathtt{k}), \varphi(t))) = \lambda^\dagger \mathtt{x}.\varphi(t)$.

The case for $t \equiv (a, \mathtt{x}_{n+1})$ for an $a \in (\mathtt{X} \cup \underline{A})$ with $a \not\equiv \mathtt{x}_{n+1}$ and the induction step follow similarly.

$(1 \to 2)$ Assume that $A[\mathtt{x}_1, \ldots, \mathtt{x}_n]$ is reflexive. Also assume that for some $s, t \in \mathcal{T}(\{\mathtt{x}_1, \ldots, \mathtt{x}_{n+1}\} \cup \underline{A})$, we have $s \approx_{\mathtt{x}_1 \ldots \mathtt{x}_{n+1}} t$. Since φ is a homomorphism, we also have that $\varphi(s) \approx_{\mathtt{x}} \varphi(t)$. Since $T^n A$ is isomorphic to $A[\mathtt{x}_1, \ldots, \mathtt{x}_n]$, also $T^n A$ is reflexive. And thus, from Proposition 4.3.13, we have that $[\![\lambda^\dagger \mathtt{x}.\varphi(s)]\!] = [\![\lambda^\dagger \mathtt{x}.\varphi(t)]\!]$. Equation (4.3) then tells us that $f(\lambda^\dagger \mathtt{x}_{n+1}.s) = f(\lambda^\dagger \mathtt{x}_{n+1}.t)$. Since f is a homomorphism, we conclude that $\lambda^\dagger \mathtt{x}_{n+1}.s \approx_{\mathtt{x}_1 \ldots \mathtt{x}_n} \lambda^\dagger \mathtt{x}_{n+1}.t$.

$(2 \to 1)$ Following the statement, assume that $s \approx_{\mathtt{x}_1 \ldots \mathtt{x}_{n+1}} t \Rightarrow \lambda^\dagger \mathtt{x}_{n+1}.s \approx_{\mathtt{x}_1 \ldots \mathtt{x}_n} \lambda^\dagger \mathtt{x}_{n+1}.t$ for all $s, t \in \mathcal{T}(\{\mathtt{x}_1, \ldots, \mathtt{x}_{n+1}\} \cup \underline{A})$. We show that $\varphi(s) \approx_{\mathtt{x}} \varphi(t) \Rightarrow [\![\lambda^\dagger \mathtt{x}.\varphi(s)]\!] = [\![\lambda^\dagger \mathtt{x}.\varphi(t)]\!]$ for all $s, t \in \mathcal{T}(\{\mathtt{x}_1, \ldots, \mathtt{x}_{n+1}\} \cup \underline{A})$, which is enough to show that $T^n A$ and thus also $A[\mathtt{x}_1, \ldots, \mathtt{x}_n]$ is reflexive, using Proposition 4.3.13 and the fact that φ is surjective.

Assume that $\varphi(s) \approx_{\mathtt{x}} \varphi(t)$. Since φ is a homomorphism, we also have that $s \approx_{\mathtt{x}_1 \ldots \mathtt{x}_{n+1}} t$ and thus $\lambda^\dagger \mathtt{x}_{n+1}.s \approx_{\mathtt{x}_1 \ldots \mathtt{x}_n} \lambda^\dagger \mathtt{x}_{n+1}.t$. Since f is a homomorphism, we then also have that $f(\lambda^\dagger \mathtt{x}_{n+1}.s) = f(\lambda^\dagger \mathtt{x}_{n+1}.t)$, and then we are done by equation (4.3).

$(1 \leftrightarrow 3)$ From Proposition 4.2.5, we have that $A[\mathtt{x}_1, \ldots, \mathtt{x}_n]$ is reflexive iff $T^n A$ is reflexive. From Corollary 4.3.2 and the fact that f is surjective, we have that this holds iff

$$(f(s), \mathtt{x}) \approx_{\mathtt{x}} (f(t), \mathtt{x}) \Rightarrow f(\mathtt{e}) *_1 f(s) = f(\mathtt{e}) *_1$$

for all $s, t \in \mathcal{T}(\{\mathtt{x}\} \cup \underline{T^n A})$. By definition of φ and since f is a homomorphism, one can easily see that the latter holds iff

$$\varphi(s, \mathtt{x}_{n+1}) \approx_{\mathtt{x}} \varphi(t, \mathtt{x}_{n+1}) \Rightarrow f((\mathtt{e}, s)) = f((\mathtt{e}, t))$$

for all s, $t \in \mathcal{T}(\{\mathtt{x}_1, \ldots, \mathtt{x}_n\} \cup \underline{A})$. Since φ and f are homomorphisms, this holds iff

$$(s, \mathtt{x}_{n+1}) \approx_{\mathtt{x}_1 \ldots \mathtt{x}_{n+1}} (t, \mathtt{x}_{n+1}) \Rightarrow (\mathtt{e}, s) \approx_{\mathtt{x}_1 \ldots \mathtt{x}_n} (\mathtt{e}, t)$$

for all s, $t \in \mathcal{T}(\{\mathtt{x}_1, \ldots, \mathtt{x}_n\} \cup \underline{A})$. □

4.4 Strong reflexivity

In this section, we introduce strong reflexivity. We first give the definition and discuss some immediate results. Then we outline the relation of strongly reflexive combinatory pre-models with combinatory models.

4.4.1 Definition and basics

As said before, and as can be seen from Proposition 4.3.13, reflexivity can be seen as the mapping $t \mapsto \lambda^\dagger \mathtt{x}.t$ being well-defined for polynomials with one indeterminate. Respecting the ξ-rule of the lambda calculus, using the $\lambda^\dagger$-abstraction, amounts to the mapping $t \mapsto \lambda^\dagger \mathtt{x}.t$ being well-defined for polynomials with any number of indeterminates. It turns out that for this, it is sufficient if the polynomial algebra with one indeterminate is reflexive.

Definition 4.4.1. A combinatory pre-model $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{i}, \mathtt{e})$ is *strongly reflexive* if $A[\mathtt{x}]$ is reflexive.

We only need the polynomial algebra with one indeterminate to be reflexive, because this implies that the polynomial algebra with any number of indeterminates is reflexive, as was stated in Proposition 4.3.8

We will give an alternative of Proposition 4.3.14, that states that strong reflexivity is sufficient for the mapping $t \mapsto \lambda^\dagger \mathtt{x}.t$ to be well-defined for polynomials with any number of indeterminates.

Theorem 4.4.2. *For any combinatory pre-model $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{i}, \mathtt{e})$ and any $n \in \mathbb{N}_{\geq 1}$, the following are equivalent.*

1. *A is strongly reflexive.*
2. *For all s, $t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$,*
$$s \approx_{\mathtt{X}} t \Rightarrow \lambda^\dagger \mathtt{x}.s \approx_{\mathtt{X}} \lambda^\dagger \mathtt{x}.t.$$
3. *For all s, $t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$,*
$$(s, \mathtt{x}) \approx_{\mathtt{X}} (t, \mathtt{x}) \Rightarrow (\mathtt{e}, s) \approx_{\mathtt{X}} (\mathtt{e}, t).$$

Proof. Let $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{i}, \mathtt{e})$ be a combinatory pre-model. Note that for any $n \in \mathbb{N}$, if $A[\mathtt{x}_1, \ldots, \mathtt{x}_n]$ is reflexive, then also $A[\mathtt{x}]$. Thus, with Proposition 4.3.8, we have that A is strongly reflexive if and only if $A[\mathtt{x}_1, \ldots, \mathtt{x}_n]$ is reflexive for any $n \in \mathbb{N}$.

Also for any s, we have $s \in \mathcal{T}(\mathtt{X} \cup \underline{A})$ if and only if there exists an n such that $s \in \mathcal{T}(\{\mathtt{x}_1, \ldots, \mathtt{x}_n\} \cup \underline{A})$.

This, together with Remark 4.2.10 and Proposition 4.3.14, proves the theorem □

Similarly to reflexivity (cf Proposition 4.3.4, we can characterize strong reflexivity with seven equations. These are obtained by taking the $\lambda^\dagger$-closures of both sides of the equations of 4.3.4.

In the theorem below, we will abbreviate the notation for application of terms, as in Notation 3.1.3.

Theorem 4.4.3. *A combinatory pre-model $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{i}, \mathtt{e})$ is strongly reflexive if and only if it satisfies the following equations.*

i) $\lambda^\dagger \mathtt{xy.e(s(s(kk)x)y)} = \lambda^\dagger \mathtt{xy.ex}$,

ii) $\lambda^\dagger \mathtt{xyz.e(s(s(s(ks)x)y)z)} = \lambda^\dagger \mathtt{xyz.e(s(sxz)(syz))}$,

iii) $\lambda^\dagger \mathtt{x.e(s(ki)x)} = \lambda^\dagger \mathtt{x.ex}$,

iv) $\lambda^\dagger \mathtt{xy.e(s(s(ke)x)y)} = \lambda^\dagger \mathtt{xy.e(sxy)}$,

v) $\lambda^\dagger \mathtt{xy.e(s(kx)(ky))} = \lambda^\dagger \mathtt{xy.e(k(xy))}$,

vi) $\lambda^\dagger \mathtt{x.e(s(kx)i)} = \lambda^\dagger \mathtt{x.ex}$,

vii) $\lambda^\dagger \mathtt{xy.e(s(ex)(ey))} = \lambda^\dagger \mathtt{xy.e(sxy)}$.

Proof. Let $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{i}, \mathtt{e})$ be a combinatory pre-model. First, assume that A is strongly reflexive. Then by Proposition 4.3.8, also A is reflexive, and $A[\mathtt{x}_1, \ldots, \mathtt{x}_n]$ is reflexive for any n. From the reflexivity of A, we have that the seven equations of Proposition 4.3.4 hold. Then by the reflexivity of $A[\mathtt{x}_1, \ldots, \mathtt{x}_n]$ for $n \leq 2$, and by repeated application of Proposition 4.3.14, we can show how to obtain the seven equations as in the statement of the theorem. We will show how to derive equation (i) from the theorem. The rest of the equations follow similarly. Since $A[\mathtt{x}, \mathtt{y}]$ is reflexive, we have from Proposition 4.3.4 equation (1) that

$$(\mathtt{e}, ((\mathtt{s}, ((\mathtt{s}, (\mathtt{k}, \mathtt{k})), \mathtt{x})), \mathtt{y})) \approx_{\mathtt{x},\mathtt{y}} (\mathtt{e}, \mathtt{x}).$$

We also have that $A[\mathtt{x}]$ is reflexive, and thus, by Proposition 4.3.14, we have

$$\lambda^\dagger \mathtt{y}.(\mathtt{e}, ((\mathtt{s}, ((\mathtt{s}, (\mathtt{k}, \mathtt{k})), \mathtt{x})), \mathtt{y})) \approx_{\mathtt{x}} \lambda^\dagger \mathtt{y}.(\mathtt{e}, \mathtt{x}).$$

Since also A is reflexive, by repeating the previous argument, we obtain

$$\lambda^\dagger \mathtt{xy}.(\mathtt{e}, ((\mathtt{s}, ((\mathtt{s}, (\mathtt{k}, \mathtt{k})), \mathtt{x})), \mathtt{y})) = \lambda^\dagger \mathtt{xy}.(\mathtt{e}, \mathtt{x}).$$

Now for the other direction, assume that A satisfies the seven equations from the theorem. We have that η_A^1 (cf. Definition 3.5.5) is a homomorphism from A to $A[\mathtt{x}]$. Then $A[\mathtt{x}]$ also satisfies the seven equations. From these, we show how to obtain that $A[\mathtt{x}]$ also satisfies the seven equations from Proposition 4.3.4, which implies that $A[\mathtt{x}]$ is reflexive (and thus A is strongly reflexive). We show how to do this for equation 1. In $A[\mathtt{x}]$, we have that

$$\lambda^\dagger \mathtt{xy.e(s(s(kk)x)y)} = \lambda^\dagger \mathtt{xy.ex}.$$

Let a, $b \in \mathcal{T}(\{x\} \cup \underline{A})/\approx_{\mathtt{x}}$. If we apply both sides of the equation with a and b, we obtain that

$$\mathtt{e}(\mathtt{k}_1 \cdot_1 a \cdot_1 b) = \mathtt{e}a.$$

The other equations follow similarly. □

The equations of Theorem 4.4.3 are *closed*: they only consist of equations of elements of $\underline{A}$, that are made up from applications between the combinators $\mathtt{k}$, $\mathtt{s}$, $\mathtt{i}$ and $\mathtt{e}$. This ensures the following.

Corollary 4.4.4. *If A is strongly reflexive, B is a combinatory pre-model and $f\colon A \to B$ is a homomorphism, then B is also strongly reflexive.*

4.4.2 Interpreting the lambda calculus

We will show that strongly reflexive combinatory pre-models are sound with respect to the lambda calculus. We first define a way of interpreting lambda-terms in combinatory pre-models, using the $\lambda^\dagger$-abstraction. Recall Definition 3.2.1 and 3.2.2.

Definition 4.4.5. Let A be a combinatory pre-model. Define the mapping $(\cdot)_{CL^\dagger} : \mathsf{T}_\Lambda(A) \to \mathcal{T}(\mathtt{X} \cup \underline{A})$ as follows:

i) $(\mathtt{x}_i)_{CL^\dagger} = \mathtt{x}_i$ for any $i \in \mathbb{N}$.

ii) $(a)_{CL^\dagger} = a$ for any $a \in \underline{A}$.

iii) $(s \cdot t)_{CL^\dagger} = (s_{CL^\dagger}, t_{CL^\dagger})$ for any s, $t \in \mathsf{T}_\Lambda(A)$.

iv) $(\lambda \mathtt{x}.s)_{CL^\dagger} = \lambda^\dagger \mathtt{x}.(s)_{CL^\dagger}$ for any $s \in \mathsf{T}_\Lambda(A)$.

As with lambda algebras (cf. Proposition 3.5.9), strongly reflexive combinatory pre-models only respect the ξ-rule of the lambda calculus in the absolute sense. That is, when considering equality over polynomials. We therefore need the following lemma.

Lemma 4.4.6. *For any combinatory pre-model A, for all s, $t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$,*

$$s \approx_{\mathtt{X}} t \Rightarrow A \models s = t.$$

Proof. Let A be a combinatory pre-model, let $n \in \mathbb{N}_{\geq 1}$ and let s, $t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$ be such that $FV(s) \cup FV(t) \subseteq \{\mathtt{x}_1, \ldots, \mathtt{x}_n\}$. We thus have that s, $t \in \mathcal{T}(\{\mathtt{x}_1, \ldots, \mathtt{x}_n\} \cup \underline{A})$. Assume that $s \approx_{\mathtt{X}} t$. Then $s \approx_{\mathtt{x}_1 \ldots \mathtt{x}_n} t$ (cf. Remark 4.2.10).

We want to show that for any valuation $\rho\colon \{\mathtt{x}_1, \ldots, \mathtt{x}_n\} \to \underline{A}$, we have $A, \rho \models s = t$ (cf. Definition 3.1.7). Let $\rho\colon \{\mathtt{x}_1, \ldots, \mathtt{x}_n\} \to \underline{A}$ be any valuation. Let $f\colon A \to A$ be the identity isomorphism. By Proposition 4.2.2, we have that there exists a unique $\tilde{f}\colon A[\mathtt{x}_1, \ldots, \mathtt{x}_n] \to A$ such that $\tilde{f} \circ \eta_A^n = f$ and $\tilde{f}(\langle \mathtt{x}_i \rangle_n) = \rho(\mathtt{x}_i)$. Then $[\![s]\!]_\rho^A = \tilde{f}(\langle s \rangle_n) = \tilde{f}(\langle t \rangle_n) = [\![r]\!]_\rho^A$, and thus $A, \rho \models s = t$ as required. □

Theorem 4.4.7. *For any strongly reflexive combinatory pre-model A, for all $s, t \in \mathsf{T}_\Lambda(A)$, we have*

$$\lambda \vdash s = t \Rightarrow A \models s_{CL^\dagger} = t_{CL^\dagger}.$$

Proof. By Lemma 4.4.6, it is sufficient to show that

$$\lambda \vdash s = t \Rightarrow s_{CL^\dagger} \approx_{\mathtt{X}} t_{CL^\dagger}$$

for any $s, t \in \mathsf{T}_\Lambda(A)$. Recall Definition 2.2.2. We will show this by induction on the derivations in the lambda calculus.

Let $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{i}, \mathtt{e})$ be an algebraic combinatory model. It is clear, in case of (refl), that $s_{CL^\dagger} \approx_{\mathtt{x}} s_{CL^\dagger}$ for $s \in \mathsf{T}_\Lambda(A)$. The axioms (symm), (trans) and (cong) follow similarly. Since the polynomial algebras are isomorphic up to renaming of the variables, and by Remark 4.3.10, we have that the α-rule is satisfied similarly. The case for the β-rule follows from Proposition 4.3.11.

From Theorem 4.4.2, if $s_{CL^\dagger} \approx_{\mathtt{x}} t_{CL^\dagger}$, then also $\lambda^\dagger \mathtt{x}.s_{CL^\dagger} \approx_{\mathtt{x}} \lambda^\dagger \mathtt{x}.t_{CL^\dagger}$ for any $s_{CL^\dagger}$, $t_{CL^\dagger} \in \mathcal{T}(\mathtt{X} \cup \underline{A})$. This concludes the case for the ξ-rule and thus the proof of the statement. □

4.4.3 Combinatory models

Recall the definition of a combinatory model (Definition 3.4.1). In contrast to combinatory pre-models, the definition of combinatory models does not contain a combinator $\mathtt{i}$. However, the element $\mathtt{skk}$ can take on this role:

Lemma 4.4.8. *If $(\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{e})$ is a combinatory model, then $(\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{skk}, \mathtt{e})$ is a strongly reflexive combinatory pre-model.*

Proof. Let $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{e})$ be a combinatory model, let s, $t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$ such that $s \approx_{\mathtt{x}} t$ and let $\mathtt{x}$ be a variable. By Theorem 4.4.2, it is sufficient to show that $\lambda^\dagger \mathtt{x}.s \approx_{\mathtt{x}} \lambda^\dagger \mathtt{x}.t$.

$(\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{skk}, \mathtt{e})$ is a combinatory pre-model, and thus $A \models s = t$ by Lemma 4.4.6. Then $\lambda^\dagger \mathtt{x}.s \approx_{\mathtt{x}} \lambda^\dagger \mathtt{x}.t$ follows from Proposition 4.4.9 and Proposition 4.4.10. □

The other direction does not work: not every strongly reflexive combinatory pre-model is a combinatory model. We will say more about this in Remark 4.6.12 later.

We now show that combinatory models are weakly extensional with respect to the $\lambda^\dagger$-abstraction, equivalently to the weak extensionality of lambda models (cf. Proposition 3.3.3).

However, the $\lambda^\dagger$-abstraction (Definition 4.3.9) is only defined for terms of combinatory pre-models. We redefine $\lambda^\dagger$ for combinatory models, but keep the same notation. For any combinatory model $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{e})$, we let $\lambda^\dagger \mathtt{x}.\mathtt{x} = (\mathtt{e}, \mathtt{skk})$, and the rest of the definition follows similarly.

Proposition 4.4.9. *For any combinatory model $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{e})$, and for all s, $t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$,*

$$A \models s = t \Rightarrow A \models \lambda^\dagger \mathtt{x}.s = \lambda^\dagger \mathtt{x}.t$$

Proof. Let $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{e})$, and let s, $t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$. Assume that $A \models s = t$. By definition, for any valuation ρ we have $[\![s]\!]^A_\rho = [\![t]\!]^A_\rho$. Fix a variable $\mathtt{x}$ and a $c \in \underline{A}$. Then also, for any valuation σ such that $\sigma(\mathtt{x}) = c$, we have $[\![s]\!]^A_\sigma = [\![t]\!]^A_\sigma$. And thus $[\![s[\mathtt{x}/c]]\!]^A_\rho = [\![t[\mathtt{x}/c]]\!]^A_\rho$ for any valuation ρ.

Proposition 4.3.11 still holds for combinatory models. And thus $[\![\lambda^\dagger \mathtt{x}.s]\!]^A_\rho \cdot c = [\![(\lambda^\dagger \mathtt{x}.s, c)]\!]^A_\rho = [\![(\lambda^\dagger \mathtt{x}.t, c)]\!]^A_\rho = [\![\lambda^\dagger \mathtt{x}.t]\!]^A_\rho \cdot c$ for any valuation ρ. Since c was arbitrary, from Definition 3.4.1 equation (ii), we get $\mathtt{e}[\![\lambda^\dagger \mathtt{x}.s]\!]^A_\rho = \mathtt{e}[\![\lambda^\dagger \mathtt{x}.t]\!]^A_\rho$ for any valuation ρ.

$(\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{skk}, \mathtt{e})$ is strongly reflexive and thus also reflexive by Lemma 4.4.8, and thus by Lemma 4.3.12 we get $[\![\lambda^\dagger \mathtt{x}.s]\!]^A_\rho = [\![\lambda^\dagger \mathtt{x}.t]\!]^A_\rho$ for any valuation ρ. □

Similarly to Proposition 3.5.11 for lambda models, the following holds.

Proposition 4.4.10. *For any combinatory model A and all s, $t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$,*

$$A \models s = t \Rightarrow s \approx_{\mathtt{X}} t.$$

Proof. Let $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{e})$ be a combinatory model, let $n \in \mathbb{N}_{\geq 1}$ and let s, $t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$ with $FV(s) \cup FV(t) \subseteq \{\mathtt{x}_1, \ldots, \mathtt{x}_n\}$. Assume that $A \models s = t$.

By Proposition 4.4.9, we have $A \models \lambda^\dagger \mathtt{x}_1 \cdots \mathtt{x}_n.s = \lambda^\dagger \mathtt{x}_1 \cdots \mathtt{x}_n.t$. The latter is a closed equation, and thus also $\lambda^\dagger \mathtt{x}_1 \cdots \mathtt{x}_n.s \approx_{\mathtt{X}} \lambda^\dagger \mathtt{x}_1 \cdots \mathtt{x}_n.t$. Since $\approx_{\mathtt{X}}$ is a congruence relation, $(\cdots(\lambda^\dagger \mathtt{x}_1 \cdots \mathtt{x}_n.s, \mathtt{x}_n), \cdots, \mathtt{x}_1) \approx_{\mathtt{X}} (\cdots(\lambda^\dagger \mathtt{x}_1 \cdots \mathtt{x}_n.t, \mathtt{x}_n), \cdots, \mathtt{x}_1)$, and thus $s \approx_{\mathtt{X}} t$ by Proposition 4.3.11 (which still holds for combinatory models). □

Similarly to how we argued in Section 3.5.3, this shows us that in combinatory models every polynomial is determined by its behaviour as a function. Such as we could already obtain from the Meyer-Scott axiom, combinatory models interpret the lambda abstraction as a function, in contrast to strongly reflexive combinatory pre-models, where it is interpreted as a polynomial.

The relationship between combinatory models and strongly reflexive combinatory pre-models is similar to that of lambda models and lambda algebras (cf. Proposition 3.5.6).

Theorem 4.4.11. *For any combinatory pre-model $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{i}, \mathtt{e})$, the following are equivalent.*

1. *A is strongly reflexive.*
2. *$A[\mathtt{X}]$ is strongly reflexive.*
3. *$A[\mathtt{X}]$ is reflexive.*
4. *$(\underline{A[\mathtt{X}]}, *_{\mathtt{X}}, \langle \mathtt{k} \rangle_{\mathtt{X}}, \langle \mathtt{s} \rangle_{\mathtt{X}}, \langle \mathtt{e} \rangle_{\mathtt{X}})$ is a combinatory model.*

Proof. Let $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{i}, \mathtt{e})$ be a combinatory pre-model.
$(1 \leftrightarrow 2)$ From Proposition 4.2.9.(2) we have that there exists homomorphisms between A and $A[\mathtt{X}]$. Then from Corollary 4.4.4 we have that A is strongly reflexive if and only if $A[\mathtt{X}]$ is.
$(2 \leftrightarrow 3)$ The left-to-right direction is clear. Assume that $A[\mathtt{X}]$ is reflexive. From Proposition 4.2.9.(3) and Corollary 4.3.7 we have that $(A[\mathtt{X}])[\mathtt{x}]$ is reflexive for any variable $\mathtt{x}$. And thus $A[\mathtt{X}]$ is strongly reflexive.
$(1 \to 4)$ Let A be strongly reflexive and let s, $t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$. It is sufficient to show that $(s, r) \approx_{\mathtt{X}} (t, r)$ for all $r \in \mathcal{T}(\mathtt{X} \cup \underline{A})$ implies that $(\mathtt{e}, s) \approx_{\mathtt{X}} (\mathtt{e}, t)$. So assume that $(s, r) \approx_{\mathtt{X}} (t, r)$ for all $r \in \mathcal{T}(\mathtt{X} \cup \underline{A})$, and let $n \in \mathbb{N}_{\geq 1}$ such that s, $t \in \mathcal{T}(\{\mathtt{x}_1, \ldots, \mathtt{x}_n\} \cup \underline{A})$. Then $(s, \mathtt{x}_{n+1}) \approx_{\mathtt{x}_1 \ldots \mathtt{x}_{n+1}} (t, \mathtt{x}_{n+1})$ by Remark 4.2.10. From Proposition 4.3.8 we have that $A[\mathtt{x}_1, \ldots, \mathtt{x}_n]$ is reflexive, and from Proposition 4.3.14 we then obtain that $(\mathtt{e}, s) \approx_{\mathtt{x}_1 \ldots \mathtt{x}_n} (\mathtt{e}, t)$ and thus $(\mathtt{e}, s) \approx_{\mathtt{X}} (\mathtt{e}, t)$.
$(4 \to 1)$ We will use Theorem 4.4.2 to show the claim. Let $(\underline{A[\mathtt{X}]}, *_{\mathtt{X}}, \langle \mathtt{k} \rangle_{\mathtt{X}}, \langle \mathtt{s} \rangle_{\mathtt{X}}, \langle \mathtt{e} \rangle_{\mathtt{X}})$ be a combinatory model and assume that $s \approx_{\mathtt{X}} t$ for some s, $t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$.

From $s \approx_{\mathtt{X}} t$ we have that $\langle s \rangle_{\mathtt{X}} = \langle t \rangle_{\mathtt{X}}$. Then $\lambda^\dagger \mathtt{x}.\langle s \rangle_{\mathtt{X}} = \lambda^\dagger \mathtt{x}.\langle t \rangle_{\mathtt{X}}$ by Proposition 4.4.9. Since $\langle q \rangle_{\mathtt{X}} *_{\mathtt{X}} \langle r \rangle_{\mathtt{X}} = \langle (q, r) \rangle_{\mathtt{X}}$ for all $q, r \in \mathcal{T}(\mathtt{X} \cup \underline{A})$, we obtain that $\langle \lambda^\dagger \mathtt{x}.s \rangle_{\mathtt{X}} = \langle \lambda^\dagger \mathtt{x}.t \rangle_{\mathtt{X}}$ and thus $\lambda^\dagger \mathtt{x}.s \approx_{\mathtt{X}} \lambda^\dagger \mathtt{x}.t$. □

The above tells us that strongly reflexive combinatory pre-models are retracts of combinatory models.

Theorem 4.4.12. *A combinatory pre-model is strongly reflexive if and only if it is the retract of a combinatory model.*

Proof. Let A be a combinatory pre-model. First assume that A is the retract of a combinatory model. By Lemma 4.4.8, A is also the retract of a strongly reflexive combinatory pre-model, and by Corollary 4.4.4, A is strongly reflexive.

Now assume that A is strongly reflexive. By Proposition 4.2.9.2, it is the retract of $A[\mathtt{X}] = (\underline{A[\mathtt{X}]}, *_{\mathtt{X}}, \langle \mathtt{k} \rangle_{\mathtt{X}}, \langle \mathtt{s} \rangle_{\mathtt{X}}, \langle \mathtt{i} \rangle_{\mathtt{X}}, \langle \mathtt{e} \rangle_{\mathtt{X}})$. Then A is also the retract of $(\underline{A[\mathtt{X}]}, *_{\mathtt{X}}, \langle \mathtt{k} \rangle_{\mathtt{X}}, \langle \mathtt{s} \rangle_{\mathtt{X}}, \langle \mathtt{e} \rangle_{\mathtt{X}})$, which is a combinatory model by Theorem 4.4.11. □

Remark 4.4.13. The same argument works for lambda algebras being retracts of lambda models.

4.5 Categorical models

In this section we will discuss categorical models of the lambda calculus. We outline the known result of how to construct a *cartesian closed category* with a *reflexive object* from a lambda algebra, and the other way around. We then discuss a similar result for algebraic combinatory models.

A category is *cartesian closed* if it has all finite products and exponentials. A *reflexive object* in a cartesian closed category is an object U together with two arrows $F\colon U \to U^U$ and $G\colon U^U \to U$, such that $F \circ G = \mathrm{id}_U$. In other words, it is an object U such that U^U is a retract of U.

There is another way to characterize cartesian closed categories, using *cartesian closed monoids*. Cartesian closed monoids have been used by Scott to show how to construct a cartesian closed category from a lambda algebra in [37]. The term "cartesian closed monoid" comes from Koymans [26], similar structures as a "C-monoid" and a "weak C-monoid" have been used in [29].

Definition 4.5.1. A *monoid* is a structure $M = (\underline{M}, \cdot, I)$, where $\underline{M}$ is a set called the *domain*, $\cdot$ is a binary operation on $\underline{M}$ that is associative, and I is an element of $\underline{M}$ such that for all $a \in \underline{M}$,

$$I \cdot a = a \cdot I = a.$$

Definition 4.5.2. A monoid $M = (\underline{M}, \cdot, I)$ is *cartesian closed* when there exist elements $p, q, \varepsilon \in \underline{M}$, and operations $[\cdot,\cdot]\colon \underline{M} \times \underline{M} \to \underline{M}$ and $\Lambda(\cdot)\colon \underline{M} \to \underline{M}$, such that for all a, b, $c \in \underline{M}$,

i) $p \cdot [a, b] = a$ and $q \cdot [a, b] = b$,

ii) $[a, b] \cdot c = [a \cdot c, b \cdot c]$,

iii) $\varepsilon \cdot [p, q] = \varepsilon$,

iv) $\varepsilon \cdot [\Lambda(a) \cdot p, q] = a \cdot [p, q]$,

v) $\Lambda(\varepsilon) \cdot \Lambda(a) = \Lambda(a)$,

vi) $\Lambda(\varepsilon \cdot [a \cdot p, q]) = \Lambda(\varepsilon) \cdot a$.

There is a standard way of making a category from a monoid [26, 37].

Definition 4.5.3. The *Karoubi envelope* $\mathbb{C}(M)$ of a monoid $M = (\underline{M}, \cdot, I)$ is the category with as objects the $\{a \in \underline{M} : a \cdot a = a\}$, for any two objects a, b a morphism $f\colon a \to b$ is an $f \in \underline{M}$ such that $b \cdot f \cdot a = f$, for any object a the identity id_a is a itself, and for any morphisms $f\colon a \to b$ and $g\colon b \to c$ for objects a, b, c, the composition $g \circ f$ is defined as $g \cdot f$.

One can easily verify the following.

Proposition 4.5.4. *The Karoubi envelope of a cartesian closed monoid is a cartesian closed category.*

Proof. See [26, 37]. □

Theorem 4.5.5. *For any lambda algebra $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s})$ and variable $\mathtt{x}$, the structure $M_A = (\underline{A}, \circ, \mathtt{skk})$ is a cartesian closed monoid, where $a \circ b = \lambda^*\mathtt{x}.(a, (b, \mathtt{x}))$ for all a, $b \in \underline{A}$. Moreover, the cartesian closed category $\mathbb{C}(M_A)$ has $\mathtt{skk}$ as a reflexive object with arrows $F = G = \mathbf{1} = \mathtt{s}(\mathtt{k}(\mathtt{skk}))$.*

Proof. See Section 7 in [37]. □

Lambek first gave a construction of a lambda algebra from a cartesian closed category with a reflexive object in [28]. Koymans later gave another construction of a lambda algebra $\mathbb{M}_{\mathbb{C}}$ associated with every cartesian closed category $\mathbb{C}$ with a reflexive object in [26]. Moreover, Koymans showed that for any lambda algebra $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s})$, the lambda algebra $\mathbb{M}_{\mathbb{C}(M_A)}$ is isomorphic to A, where $M_A = (\underline{A}, \circ, \mathtt{skk})$ as in the above theorem.

We show that we can construct a cartesian closed monoid from a strongly reflexive combinatory pre-model. For the rest of this section, let $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{i}, \mathtt{e})$ be a reflexive (not strongly) combinatory pre-model. Recall the set $\mathtt{e}A = \{\mathtt{e}a : a \in \underline{A}\} = \{a \in \underline{A} : a = \mathtt{e}a\}$ (cf. Corollary 4.2.14). Define an application $\circ$ on $\mathtt{e}A$ by

$$a \circ b = [\![\lambda^\dagger \mathtt{x}.(a, (b, \mathtt{x}))]\!]^{\underline{A}}$$

for all a, $b \in \mathtt{e}A$. Define $I \in \mathtt{e}A$ by $I = \mathtt{ei}$.

Proposition 4.5.6. *The structure $(\mathtt{e}A, \circ, I)$ is a monoid.*

Proof. We need to verify the following.

1. $a \circ (b \circ c) = (a \circ b) \circ c$ for all a, b, $c \in \mathtt{e}A$.
2. $I \circ a = a \circ I = a$ for all $a \in \mathtt{e}A$.

1. Let a, b, $c \in \mathtt{e}A$. We have $(a \circ (b \circ c), \mathtt{x}) \approx_{\mathtt{x}} (a, (b, (c, \mathtt{x}))) \approx_{\mathtt{x}} ((a \circ b) \circ c, \mathtt{x})$ for any variable $\mathtt{x}$. Then $\mathtt{e}(a \circ (b \circ c)) = \mathtt{e}((a \circ b) \circ c)$ by Corollary 4.3.2, and $a \circ (b \circ c) = (a \circ b) \circ c$ by Lemma 4.3.12.
2. Let $a \in \mathtt{e}A$. Then $(I \circ a, \mathtt{x}) \approx_{\mathtt{x}} (\mathtt{ei}, (a, \mathtt{x})) \approx_{\mathtt{x}} (a, \mathtt{x})$ for any variable $\mathtt{x}$. Then $I \circ a = \mathtt{e}(I \circ a) = \mathtt{e}a = a$ by Lemma 4.3.12 and Corollary 4.3.2. The case $a \circ I = a$ follows similarly. □

For a cartesian closed monoid we need specific elements as in Definition 4.5.2. First, recall the $\mathtt{t}$, $\mathtt{f}$, $\mathtt{pair} \in \mathcal{T}(\mathtt{X} \cup \underline{A})$ from Section 4.2.3. We redefine these for the $\lambda^\dagger$-abstraction.

$$\mathtt{t} = [\![\mathtt{k}]\!]^{\underline{A}}, \qquad \mathtt{f} = [\![\lambda^\dagger \mathtt{xy.y}]\!]^{\underline{A}}, \qquad \langle \cdot, \cdot \rangle = [\![\lambda^\dagger \mathtt{xyz.zxy}]\!]^{\underline{A}}.$$

We renaimed $\mathtt{pair}$ to denote $\langle \cdot, \cdot \rangle$ to improve readability in this section. For the monoid $(\mathtt{e}A, \circ, I)$, define p, q, $\varepsilon \in \mathtt{e}A$ and $[\cdot, \cdot] \colon \mathtt{e}A \times \mathtt{e}A \to \mathtt{e}A$ and $\Lambda(\cdot) \colon \mathtt{e}A \to \mathtt{e}A$ as follows.

$$p = [\![\lambda^\dagger \mathtt{x}.(\mathtt{x}, \mathtt{t})]\!]^{\underline{A}}, \qquad q = [\![\lambda^\dagger \mathtt{x}.(\mathtt{x}, \mathtt{f})]\!]^{\underline{A}}, \qquad \varepsilon = [\![\lambda^\dagger \mathtt{x}.((\mathtt{x}, \mathtt{t}), (\mathtt{x}, \mathtt{f}))]\!]^{\underline{A}},$$

and

$$[a, b] = [\![\lambda^\dagger \mathtt{x}.\langle (a, \mathtt{x}), (b, \mathtt{x}) \rangle]\!]^{\underline{A}}, \qquad \Lambda(a) = [\![\lambda^\dagger \mathtt{xy}.(a, \langle \mathtt{x}, \mathtt{y} \rangle)]\!]^{\underline{A}},$$

for all a, $b \in \mathtt{e}A$.

Theorem 4.5.7. *If A is strongly reflexive, then $(\mathtt{e}A, \circ, I)$ is a cartesian closed monoid.*

Proof. We verify that (i)-(vi) of Definition 4.5.2 hold for the elements and operations defined above.

Fix an a, b and $c \in \mathtt{e}A$.

$$\begin{aligned}
(p \circ [a,b], \mathtt{x}) &\approx_{\mathtt{x}} (p, ([a,b], \mathtt{x})) \approx_{\mathtt{x}} (p, \langle (a, \mathtt{x}), (b, \mathtt{x})\rangle) \\
&\approx_{\mathtt{x}} (\langle (a, \mathtt{x}), (b, \mathtt{x})\rangle, \mathtt{t}) \approx_{\mathtt{x}} (a, \mathtt{x}). \\
(q \circ [a,b], \mathtt{x}) &\approx_{\mathtt{x}} (b, \mathtt{x}). \\
([a,b] \circ c, \mathtt{x}) &\approx_{\mathtt{x}} ([a,b], (c, \mathtt{x})) \approx_{\mathtt{x}} \langle (a, (c, \mathtt{x})), (b, (c, \mathtt{x}))\rangle \\
&\approx_{\mathtt{x}} \langle (a \circ c, \mathtt{x}), (b \circ c, \mathtt{x})\rangle \approx_{\mathtt{x}} ([a \circ c, b \circ c], \mathtt{x}). \\
(\varepsilon \circ [p,q], \mathtt{x}) &\approx_{\mathtt{x}} (\varepsilon, ([p,q], \mathtt{x})) \approx_{\mathtt{x}} (\varepsilon, \langle (p, \mathtt{x}), (q, \mathtt{x})\rangle) \\
&\approx_{\mathtt{x}} ((\langle (p, \mathtt{x}), (q, \mathtt{x})\rangle, \mathtt{t}), (\langle (p, \mathtt{x}), (q, \mathtt{x})\rangle, \mathtt{f})) \\
&\approx_{\mathtt{x}} ((p, \mathtt{x}), (q, \mathtt{x})) \approx_{\mathtt{x}} ((\mathtt{x}, \mathtt{t}), (\mathtt{x}, \mathtt{f})) \approx_{\mathtt{x}} (\varepsilon, \mathtt{x}). \\
(\varepsilon \circ [\Lambda(a) \circ p, q], \mathtt{x}) &\approx_{\mathtt{x}} (\varepsilon, ([\Lambda(a) \circ p, q], \mathtt{x})) \approx_{\mathtt{x}} (\varepsilon, \langle (\Lambda(a) \circ p, \mathtt{x}), (q, \mathtt{x})\rangle) \\
&\approx_{\mathtt{x}} (\varepsilon, \langle (\Lambda(a), (p, \mathtt{x})), (q, \mathtt{x})\rangle) \approx_{\mathtt{x}} (\varepsilon, \langle (\Lambda(a), (\mathtt{x}, \mathtt{t})), (\mathtt{x}, \mathtt{f})\rangle) \\
&\approx_{\mathtt{x}} ((\langle (\Lambda(a), (\mathtt{x}, \mathtt{t})), (\mathtt{x}, \mathtt{f})\rangle, \mathtt{t}), (\langle (\Lambda(a), (\mathtt{x}, \mathtt{t})), (\mathtt{x}, \mathtt{f})\rangle, \mathtt{f})) \\
&\approx_{\mathtt{x}} ((\Lambda(a), (\mathtt{x}, \mathtt{t})), (\mathtt{x}, \mathtt{f})) \approx_{\mathtt{x}} (a, \langle (\mathtt{x}, \mathtt{t}), (\mathtt{x}, \mathtt{f})\rangle) \\
&\approx_{\mathtt{x}} (a, \langle (p, \mathtt{x}), (q, \mathtt{x})\rangle) \approx_{\mathtt{x}} (a, ([p,q], \mathtt{x})) \approx_{\mathtt{x}} (a \circ [p,q], \mathtt{x}). \\
((\Lambda(\varepsilon) \circ \Lambda(a), \mathtt{x}), \mathtt{y}) &\approx_{\mathtt{x},\mathtt{y}} ((\Lambda(\varepsilon), (\Lambda(a), \mathtt{x})), \mathtt{y}) \approx_{\mathtt{x},\mathtt{y}} (\varepsilon, \langle (\Lambda(a), \mathtt{x}), \mathtt{y}\rangle) \\
&\approx_{\mathtt{x},\mathtt{y}} ((\langle (\Lambda(a), \mathtt{x}), \mathtt{y}\rangle, \mathtt{t}), (\langle (\Lambda(a), \mathtt{x}), \mathtt{y}\rangle, \mathtt{f})) \\
&\approx_{\mathtt{x},\mathtt{y}} ((\Lambda(a), \mathtt{x}), \mathtt{y}). \\
((\Lambda(\varepsilon \circ [a \circ p, q]), \mathtt{x}), \mathtt{y}) &\approx_{\mathtt{x},\mathtt{y}} (\varepsilon \circ [a \circ p, q], \langle \mathtt{x}, \mathtt{y}\rangle) \approx_{\mathtt{x},\mathtt{y}} (\varepsilon, ([a \circ p, q], \langle \mathtt{x}, \mathtt{y}\rangle)) \\
&\approx_{\mathtt{x},\mathtt{y}} (\varepsilon, \langle (a \circ p, \langle \mathtt{x}, \mathtt{y}\rangle), (q, \langle \mathtt{x}, \mathtt{y}\rangle)\rangle) \\
&\approx_{\mathtt{x},\mathtt{y}} (\varepsilon, \langle (a, (p, \langle \mathtt{x}, \mathtt{y}\rangle)), (q, \langle \mathtt{x}, \mathtt{y}\rangle)\rangle) \\
&\approx_{\mathtt{x},\mathtt{y}} (\varepsilon, \langle (a, (\langle \mathtt{x}, \mathtt{y}\rangle, \mathtt{t})), (\langle \mathtt{x}, \mathtt{y}\rangle, \mathtt{f})\rangle) \\
&\approx_{\mathtt{x},\mathtt{y}} (\varepsilon, \langle (a, \mathtt{x}), \mathtt{y}\rangle) \approx_{\mathtt{x},\mathtt{y}} ((\Lambda(\varepsilon), (a, \mathtt{x})), \mathtt{y}) \\
&\approx_{\mathtt{x},\mathtt{y}} ((\Lambda(\varepsilon) \circ a, \mathtt{x}), \mathtt{y}).
\end{aligned}$$

By applying Corollary 4.3.2 for A and $A[\mathtt{x}]$ being reflexive, we get equalities, only with extra $\mathtt{e}$-combinators in front. Then the required equalities follow since all elements and operations are defined using the $\lambda^\dagger$-abstraction: we can thus use Lemma 4.3.12. □

Similarly to the arguments in Section 7 in [37], one can show that I is the reflexive object of the monoid. It is possible to create a lambda algebra, following Koymans' structure, from the cartesian closed category $\mathbb{C}(M(\mathtt{e}A, \circ, I))$. We say something about the difference between this construction for lambda algebras and algebraic combinatory models (cf. Section 4 in [26]).

For a lambda algebra $B = (\underline{B}, \cdot_B, \mathtt{k}_B, \mathtt{s}_B)$, the $\mathtt{k}$ and $\mathtt{s}$-combinators in the lambda algebra $\mathbb{M}_{\mathbb{C}(M_B)}$ will be equal to $\mathtt{k}_B[\![\lambda^*\mathtt{xy}.\mathtt{x}]\!]^{\underline{B}}$ and $\mathtt{k}_B[\![\lambda^*\mathtt{xyz}.((\mathtt{x},\mathtt{z}),(\mathtt{y},\mathtt{z}))]\!]^{\underline{B}}$ respectively. The isomporphism $\varphi : B \to \mathbb{M}_{\mathbb{C}(M_B)}$ is then given by $\varphi(b) = \mathtt{k}_B b$ for all $b \in \underline{B}$.

For a strongly reflexive combinatory pre-model $A = (\underline{A}, \cdot_A, \mathtt{k}_A, \mathtt{s}_A, \mathtt{i}_A, \mathtt{e}_A)$, the $\mathtt{k}$ and $\mathtt{s}$-combinators in the lambda algebra $\mathbb{M}_{\mathbb{C}(M(\mathtt{e}A,\circ,I))}$ will be equal to $\mathtt{e}_A\mathtt{k}_A[\![\lambda^\dagger\mathtt{xy}.\mathtt{x}]\!]^{\underline{A}}$ and $\mathtt{e}_A\mathtt{k}_A[\![\lambda^\dagger\mathtt{xyz}.((\mathtt{x},\mathtt{z}),(\mathtt{y},\mathtt{z}))]\!]^{\underline{A}}$ respectively. In the strongly reflexive combinatory pre-model not necessarily $\mathtt{k}_A = [\![\lambda^\dagger\mathtt{xy}.\mathtt{x}]\!]^{\underline{A}}$ and $\mathtt{s}_A = [\![\lambda^\dagger\mathtt{xyz}.((\mathtt{x},\mathtt{z}),(\mathtt{y},\mathtt{z}))]\!]^{\underline{A}}$ hold,

as is the case in lambda algebras. As discussed in Remark 3.4.8, this is probably due to (the lack of) stability. In the next section, we discuss the notion of stability for strongly reflexive combinatory pre-models, and see how stable strongly reflexive combinatory pre-models relate to lambda algebras.

4.6 Stability

4.6.1 Stability for algebraic combinatory models

This section will mostly follow the same structure as Section 3.4. We first formulate Definition 3.4.2 for combinatory pre-models.

Definition 4.6.1 ([37])**.** Let $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{i}, \mathtt{e})$ be a combinatory pre-model. For each $n \in \mathbb{N}_{\geq 1}$ we define $\varepsilon_n \in \underline{A}$ inductively as follows

$$\varepsilon_1 = \mathtt{e}, \qquad \varepsilon_{n+1} = \mathtt{s}(\mathtt{ke})(\mathtt{s}(\mathtt{k}\varepsilon_n)).$$

Lemma 3.4.3 can be formulated for polynomial algebras.

Lemma 4.6.2. *Let $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{i}, \mathtt{e})$ be a combinatory pre-model. For all $n \in \mathbb{N}_{\geq 1}$, for all variables $\mathtt{x}_1, \ldots, \mathtt{x}_n$ and for all $s, t \in \mathcal{T}(\{\mathtt{x}_1, \ldots, \mathtt{x}_n\} \cup \underline{A})$,*

$$\varepsilon_{n+1} st \approx_{\mathtt{x}} \varepsilon_n(st).$$

In particular, for all $n \in \mathbb{N}_{\geq 1}$, for all variables $\mathtt{x}_1, \ldots, \mathtt{x}_n$ and for all $a \in \underline{A}$,

i) $\varepsilon_n a \mathtt{x}_1 \cdots \mathtt{x}_{n-1} \approx_{\mathtt{x}} \mathtt{e}(a\mathtt{x}_1 \cdots \mathtt{x}_{n-1})$,

ii) $\varepsilon_n a \mathtt{x}_1 \cdots \mathtt{x}_n \approx_{\mathtt{x}} a\mathtt{x}_1 \cdots \mathtt{x}_n$.

We give some useful intermediate results that do not have a counterpart in Section 3.4. It turns out that the ε_n application and the $\lambda^\dagger$-abstraction coincide for algebraic combinatory models.

Lemma 4.6.3. *Let $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{i}, \mathtt{e})$ be a strongly reflexive combinatory pre-model. For all $a \in \underline{A}$ and for all $n \in \mathbb{N}_{\geq 1}$,*

$$\varepsilon_n a = [\![\lambda^\dagger \mathtt{x}_1 \ldots \mathtt{x}_n.(\cdots(a, \mathtt{x}_1) \cdots, \mathtt{x}_n)]\!]^{\underline{A}}.$$

Proof. We proof this by induction on $n \in \mathbb{N}$. The base case is immediate by the definitions.

Now assume that the statement holds for any strongly reflexive combinatory pre-model and for all natural numbers up to n. Consider the case for $n+1$. Let A be a strongly reflexive combinatory pre-model, let $\mathtt{x}$ be a variable and let $a \in \underline{A}$. Note that

$$\varepsilon_{n+1} a = \mathtt{e}(s(\mathtt{k}\varepsilon_n)a). \tag{4.4}$$

Since A is reflexive, it is enough to show that

$$(\varepsilon_{n+1} a, \mathtt{x}) \approx_{\mathtt{x}} ([\![\lambda^\dagger \mathtt{x}_1 \ldots \mathtt{x}_n.(\cdots(a, \mathtt{x}_1) \cdots, \mathtt{x}_{n+1})]\!]^{\underline{A}}, \mathtt{x}),$$

since then

$$\begin{aligned}
\varepsilon_{n+1}a &= \mathtt{e}(\varepsilon_{n+1}a) && \text{by Cor. 4.2.14 and eq. (4.4)}\\
&= \mathtt{e}([\![\lambda^\dagger \mathtt{x}_1 \dots \mathtt{x}_n.(\cdots(a,\mathtt{x}_1)\cdots,\mathtt{x}_{n+1})]\!]^{\underline{A}}) && \text{by Cor. 4.3.2}\\
&= [\![\lambda^\dagger \mathtt{x}_1 \dots \mathtt{x}_n.(\cdots(a,\mathtt{x}_1)\cdots,\mathtt{x}_{n+1})]\!]^{\underline{A}} && \text{by Lem. 4.3.12.}
\end{aligned}$$

From equation (4.4) and Definition 4.1.3.(i) we can deduce that $(\varepsilon_{n+1}a,\mathtt{x}) \approx_{\mathtt{x}} (((\mathtt{s},(\mathtt{k}\varepsilon_n)),a),\mathtt{x}) \approx_{\mathtt{x}} (\varepsilon_n,(a,\mathtt{x}))$. In turn, $([\![\lambda^\dagger \mathtt{x}_1 \dots \mathtt{x}_n.(\cdots(a,\mathtt{x}_1)\cdots,\mathtt{x}_{n+1})]\!]^{\underline{A}},\mathtt{x}) \approx_{\mathtt{x}} \lambda^\dagger \mathtt{x}_2\cdots\mathtt{x}_{n+1}(\cdots(a,\mathtt{x}),\mathtt{x}_{n+1}) \approx_{\mathtt{x}} [\![\lambda^\dagger \mathtt{x}_2\cdots\mathtt{x}_{n+1}(\cdots(a,\mathtt{x}),\mathtt{x}_{n+1})]\!]^{\underline{A[\mathtt{x}]}}$ which follows by Definition 4.1.3.(i). $A[\mathtt{x}]$ is also strongly reflexive by Corollary 4.4.4, and thus, by the induction hypothesis, $[\![\lambda^\dagger \mathtt{x}_2\cdots\mathtt{x}_{n+1}(\cdots(a,\mathtt{x}),\mathtt{x}_{n+1})]\!]^{\underline{A[\mathtt{x}]}} \approx_{\mathtt{x}} (\varepsilon_n,(a,\mathtt{x}))$. □

The above result allows us to reformulate Theorem 4.4.2.

Proposition 4.6.4. *A combinatory pre-model $A = (\underline{A},\cdot,\mathtt{k},\mathtt{s},\mathtt{i},\mathtt{e})$ is strongly reflexive if and only if*

$$(\cdots(a,\mathtt{x}_1),\cdots,\mathtt{x}_n) \approx_{\mathtt{x}_1\dots\mathtt{x}_n} (\cdots(b,\mathtt{x}_1),\cdots,\mathtt{x}_n) \Rightarrow \varepsilon_n a = \varepsilon_n b$$

for any $n \in \mathbb{N}_{\geq 1}$ and all a, $b \in \underline{A}$.

Proof. ($\Rightarrow$). This follows from the n-time application of Theorem 4.4.2 and Lemma 4.6.3, while taking Remark 4.2.10 into consideration.
($\Leftarrow$). Let $A = (\underline{A},\cdot,\mathtt{k},\mathtt{s},\mathtt{i},\mathtt{e})$ be a combinatory pre-model as in the statement and let $\mathtt{x}$ be a variable. We need to show that $A[\mathtt{x}]$ reflexive. For this we use Proposition 4.3.14.(3). It is thus sufficient to show that

$$(s,\mathtt{y}) \approx_{\mathtt{x},\mathtt{y}} (t,\mathtt{y}) \Rightarrow (\mathtt{e},s) \approx_{\mathtt{x}} (\mathtt{e},t)$$

for all s, $t \in \mathcal{T}(\{\mathtt{x}\}\cup\underline{A})$ and distinct variables $\mathtt{x}$ and $\mathtt{y}$. Let $\mathtt{x}$, $\mathtt{y}$ be distinct variables and let s, $t \in \mathcal{T}(\{\mathtt{x}\}\cup\underline{A})$ be such that $(s,\mathtt{y}) \approx_{\mathtt{x},\mathtt{y}} (t,\mathtt{y})$. Then by Proposition 4.3.11, $(([\![\lambda^\dagger \mathtt{x}.s]\!]^{\underline{A}},\mathtt{x}),\mathtt{y}) \approx_{\mathtt{x},\mathtt{y}} (([\![\lambda^\dagger \mathtt{x}.s]\!]^{\underline{A}},\mathtt{x}),\mathtt{y})$, and thus $\varepsilon_2([\![\lambda^\dagger \mathtt{x}.s]\!]^{\underline{A}}) = \varepsilon_2([\![\lambda^\dagger \mathtt{x}.t]\!]^{\underline{A}})$ by assumption. Then, by Lemma 4.6.2 and Proposition 4.3.11, we have $(\mathtt{e},s) \approx_{\mathtt{x}} (\mathtt{e},((\lambda^\dagger \mathtt{x}.s),\mathtt{x})) \approx_{\mathtt{x}} (\mathtt{e},((\lambda^\dagger \mathtt{x}.t),\mathtt{x})) \approx_{\mathtt{x}} (\mathtt{e},t)$. □

We go back to following Section 3.4. Similarly to stable combinatory models (cf. Def 3.4.4), we introduce the notion of stable strongly reflexive combinatory pre-models.

Definition 4.6.5. A strongly reflexive combinatory pre-model $A = (\underline{A},\cdot,\mathtt{k},\mathtt{s},\mathtt{i},\mathtt{e})$ is *stable* if it satisfies the following equations:

$$\varepsilon_2\mathtt{k} = \mathtt{k} \qquad \varepsilon_3\mathtt{s} = \mathtt{s} \qquad \varepsilon_1\mathtt{i} = \mathtt{i} \qquad \varepsilon_2\mathtt{e} = \mathtt{e}.$$

Similarly to Corollary 4.4.4, we obtain that stability is closed under homomorphisms, since the definition consists of closed equations.

Corollary 4.6.6. *If A is a stable strongly reflexive combinatory pre-model, B is a strongly reflexive combinatory pre-model and $f\colon A \to B$ is a homomorphism, then B is also stable.*

We give the analogue of Lemma 3.4.5 for polynomial algebras.

Lemma 4.6.7. *Let* $(\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{e})$ *be a strongly reflexive combinatory pre-model. Then the following implications hold.*

1. $\varepsilon_2\mathtt{k} = \mathtt{k} \Leftrightarrow \mathtt{ek} = \mathtt{k} \wedge (\mathtt{e}, (\mathtt{k}, t)) \approx_{\mathtt{X}} (\mathtt{k}, t)$ *for all* $t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$.
2. $\varepsilon_3\mathtt{s} = \mathtt{s} \Leftrightarrow \mathtt{es} = \mathtt{s} \wedge (\mathtt{e}, (\mathtt{s}, t)) \approx_{\mathtt{X}} (\mathtt{s}, t) \wedge (\mathtt{e}, (\mathtt{s}, (s, t))) \approx_{\mathtt{X}} (\mathtt{s}, (s, t)))$ *for all* s, $t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$.
3. $\varepsilon_2\mathtt{e} = \mathtt{e} \Leftrightarrow \mathtt{ee} = \mathtt{e}$.

Note that $(\mathtt{e}, (\mathtt{e}, t)) \approx_{\mathtt{X}} (\mathtt{e}, t)$ for all $t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$ already follows from reflexivity (cf. Proposition 4.3.14).

Proof. Let $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{i}, \mathtt{e})$ be a strongly reflexive combinatory pre-model. From Theorem 4.4.11, we have that $(\underline{A[\mathtt{X}]}, *_{\mathtt{X}}, \langle \mathtt{k} \rangle_{\mathtt{X}}, \langle \mathtt{s} \rangle_{\mathtt{X}}, \langle \mathtt{e} \rangle_{\mathtt{X}})$ is a combinatory model. Define a sequence $(a_n)_{n \geq 1}$ by $a_n = \mathtt{k}$ for all $n \in \mathbb{N}_{\geq 1}$. Then by Proposition 4.2.6 we have that there exists a homomorphism $\overline{\mathrm{id}_A} \colon : A[\mathtt{X}] \to A$ (such that $\overline{\mathrm{id}_A} \circ \sigma_A = \mathrm{id}_A$ and $\overline{\mathrm{id}_A}(\langle \mathtt{x}_n \rangle_{\mathtt{X}}) = \mathtt{k}$). Then by the homomorphisms $\overline{\mathrm{id}_A}$ and σ_A and Corollary 4.6.6, we have that A is stable if and only if $(\underline{A[\mathtt{X}]}, *_{\mathtt{X}}, \langle \mathtt{k} \rangle_{\mathtt{X}}, \langle \mathtt{s} \rangle_{\mathtt{X}}, \langle \mathtt{e} \rangle_{\mathtt{X}})$ is stable. By Lemma 3.4.5, the latter is equivalent to the equations of the statement. □

The meaning of stability is the same as for combinatory models. Namely, to fix the definition for the combinators based on the combinator $\mathtt{e}$, as outlined in Proposition 3.4.6 for combinatory models.

Proposition 4.6.8. *If* $(\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{i}, \mathtt{e})$ *is a stable strongly reflexive combinatory pre-model, then any other stable strongly reflexive combinatory pre-model* $(\underline{A}, \cdot, \mathtt{k}', \mathtt{s}', \mathtt{i}', \mathtt{e})$ *satisfies the following:*

$$\mathtt{k} = \mathtt{k}' \qquad \mathtt{s} = \mathtt{s}' \qquad \mathtt{i} = \mathtt{i}'.$$

Proof. The proof is similar to the proof of Proposition 3.4.6 (cf. Proposition 5.6.6 in [7]). Let $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{i}, \mathtt{e})$ and $A' = (\underline{A}, \cdot, \mathtt{k}', \mathtt{s}', \mathtt{i}', \mathtt{e})$ be stable strongly reflexive combinatory pre-models. We show that $\mathtt{k} = \mathtt{k}'$, the cases for $\mathtt{s}$ and $\mathtt{i}$ follow similarly.

Let $\mathtt{x}$ and $\mathtt{y}$ be distinct variables. By definition, we have that $A[\mathtt{x}]$ and $A'[\mathtt{x}]$ are reflexive, and thus from Proposition 4.3.14.(3) we obtain that $(\mathtt{e}, (\mathtt{k}, \mathtt{x})) \approx_{\mathtt{x}} (\mathtt{e}, (\mathtt{k}', \mathtt{x}))$, and thus $(\mathtt{s}(\mathtt{ke})\mathtt{k}, \mathtt{x}) = (\mathtt{s}(\mathtt{k}'\mathtt{e})\mathtt{k}', \mathtt{x})$. Then $\mathtt{e}(\mathtt{s}(\mathtt{ke})\mathtt{k}) = \mathtt{e}(\mathtt{s}(\mathtt{k}'\mathtt{e})\mathtt{k}')$ from Corollary 4.3.2, and thus $\mathtt{k} = \varepsilon_2\mathtt{k} = \varepsilon_2\mathtt{k}' = \mathtt{k}'$. □

We deviate again from the structure of Section 3.4. We show that in stable strongly reflexive combinatory pre-models, the interpretation of the combinators is fixed and the $\lambda^\dagger$- and λ^*-abstractions are the same.

Proposition 4.6.9. *If* $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{i}, \mathtt{e})$ *is a stable strongly reflexive combinatory pre-model, then the following equations hold.*

1. $\mathtt{i} = \mathtt{skk}$ *and* $\mathtt{e} = \mathtt{s}(\mathtt{ki})$,
2. $\mathtt{k} = [\![\lambda^\dagger \mathtt{xy}]\!]^{\underline{A}}$ *and* $\mathtt{s} = [\![\lambda^\dagger \mathtt{xyz.xz(yz)}]\!]^{\underline{A}}$,

3. $\lambda^\dagger \mathtt{x}.t \approx_{\mathtt{X}} \lambda^* \mathtt{x}.t$ *for all variables* $\mathtt{x}$ *and all* $t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$.

Proof. (1). For any variable $\mathtt{x}$, we have that

$$(\mathtt{i}, \mathtt{x}) \approx_{\mathtt{x}} \mathtt{x} \approx_{\mathtt{x}} ((\mathtt{k}, \mathtt{x}), (\mathtt{k}, \mathtt{x})) \approx_{\mathtt{x}} (\mathtt{skk}, \mathtt{x}).$$

From Corollary 4.3.2, Lemma 4.6.7, we obtain $\mathtt{i} = \mathtt{skk}$.

For any distinct variables $\mathtt{x}$ and $\mathtt{y}$,

$$((\mathtt{e}, \mathtt{x}), \mathtt{y}) \approx_{\mathtt{x},\mathtt{y}} (\mathtt{x}, \mathtt{y}) \approx_{\mathtt{x},\mathtt{y}} ((\mathtt{ki}, \mathtt{y}), (\mathtt{x}, \mathtt{y})) \approx_{\mathtt{x},\mathtt{y}} (((\mathtt{s}, \mathtt{ki}), \mathtt{x}), \mathtt{y}).$$

Then $(\mathtt{e}, (\mathtt{e}, \mathtt{x})) \approx_{\mathtt{x}} (\mathtt{e}, ((\mathtt{s}, \mathtt{ki}), \mathtt{x}))$ from Proposition 4.3.14, and thus $\mathtt{e} = \mathtt{s}(\mathtt{ki})$ similarly as before.
(2). Follows from (1) and Lemma 4.6.7.
(3). This is proven by induction on the structure of $t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$. Let $\mathtt{x}$ be a variable. When $t \equiv \mathtt{x}$, then $\lambda^\dagger \mathtt{x}.t \approx_{\mathtt{X}} (\mathtt{e}, \mathtt{i}) \approx_{\mathtt{X}} (\mathtt{e}, \mathtt{skk}) \approx_{\mathtt{X}} \mathtt{skk} \approx_{\mathtt{X}} \lambda^* \mathtt{x}.t$ by (1) and Lemma 4.6.7.

When $t \equiv (a, \mathtt{x})$ for an $a \in (\mathtt{X} \cup \underline{A})$ with $a \not\equiv \mathtt{x}$, then $\lambda^\dagger \mathtt{x}.t \approx_{\mathtt{X}} (\mathtt{e}, a)$. We have $\lambda^* \mathtt{x}.t \approx_{\mathtt{X}} \mathtt{s}(\mathtt{k}a)\mathtt{i}$. By applying both sides of equation (vi) of Theorem 4.4.3) with $\mathtt{x}$, we get that these are equal.

When $t \in (\mathtt{X} \cup \underline{A})$ and $t \not\equiv \mathtt{x}$, then $\lambda^\dagger \mathtt{x}.t \approx_{\mathtt{X}} (\mathtt{e}, (\mathtt{k}, t)) \approx_{\mathtt{X}} (\mathtt{k}, t) \approx_{\mathtt{X}} \lambda^* \mathtt{x}.t$ by Lemma 4.6.7.

The induction step then follows easily again using Lemma 4.6.7. □

Remark 4.6.10. Recall the discussion at the end of Section 4.5. Stability thus ensures that in a strongly reflexive combinatory pre-model $(\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{i}, \mathtt{e})$, we have that $\mathtt{k} = [\![\lambda^\dagger \mathtt{xy}.\mathtt{x}]\!]^{\underline{A}}$ and $\mathtt{s} = [\![\lambda^\dagger \mathtt{xyz}.((\mathtt{x}, \mathtt{z}), (\mathtt{y}, \mathtt{x}))]\!]^{\underline{A}}$.

We state the equivalent of Theorem 3.4.7.

Theorem 4.6.11. *For a combinatory pre-model* $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{e}, \mathtt{i})$*, the following are equivalent.*

1. *A is a stable strongly reflexive combinatory pre-model.*
2. $(\underline{A}, \cdot, \mathtt{k}, \mathtt{s})$ *is a lambda algebra, and* $\mathtt{i} = \mathtt{skk}$ *and* $\mathtt{e} = \mathtt{s}(\mathtt{ki})$.

Proof. $(1 \to 2)$ follows from Proposition 4.6.9 and Lemma 3.2.4. $(2 \to 1)$ follows from the definition of a lambda algebra. □

Remark 4.6.12. Plotkin [34] and Barendrecht [7, Thm 20.1.1] show that there exist lambda algebras that are not weakly extensional, and in particular, lambda algebras that are not a lambda model. A lambda algebra is a stable strongly reflexive combinatory pre-model (from the above theorem), and a lambda model is a stable combinatory model (Theorem 3.4.7). Therefore, the same result tells us that not every strongly reflexive combinatory pre-model is a combinatory model. Note that stability is defined by equations between closed terms (consisting only of $\mathtt{k}$, $\mathtt{s}$, $\mathtt{i}$ and $\mathtt{e}$).

Recall Proposition 3.4.9.

Proposition 4.6.13. *If $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{i}, \mathtt{e})$ is a strongly reflexive combinatory pre-model, then $(\underline{A}\cdot, \varepsilon_2\mathtt{k}, \varepsilon_3\mathtt{s})$ is a lambda algebra.*

Proof. Let $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{i}, \mathtt{e})$ be an algebraic combinatory model. By Theorem 4.4.11.(4), $(\underline{A[\mathtt{X}]}, *_{\mathtt{X}}, \langle \mathtt{k} \rangle_{\mathtt{X}}, \langle \mathtt{s} \rangle_{\mathtt{X}}, \langle \mathtt{e} \rangle_{\mathtt{X}})$ is a combinatory model.

From Proposition 3.4.9, we obtain that $(\underline{A[\mathtt{X}]}, *_{\mathtt{X}}, \langle \varepsilon_2\mathtt{k} \rangle_{\mathtt{X}}, \langle \varepsilon_3\mathtt{s} \rangle_{\mathtt{X}})$ is a lambda model.

Then $(\underline{A}, \cdot, \varepsilon_2\mathtt{k}, \varepsilon_3\mathtt{s})$ is a lambda algebra by Proposition 3.5.6. □

4.6.2 Curry's and Selinger's equations

Using the results of the previous section, we can give an alternative description of strongly reflexive combinatory pre-models.

Definition 4.6.14. Let $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{i}, \mathtt{e})$ be a combinatory pre-model. Define the following sets of equations.

(CA) $\mathtt{i} = \mathtt{skk}$ and $\mathtt{e} = \mathtt{s}(\mathtt{ki})$,

(L1) $\mathtt{k} = \varepsilon_2\mathtt{k}$ and $\mathtt{s} = \varepsilon_3\mathtt{s}$,

(L2)
- i) $\lambda^\dagger \mathtt{xy.e}(\mathtt{s}(\mathtt{s}(\mathtt{kk})\mathtt{x})\mathtt{y}) = \lambda^\dagger \mathtt{xy.ex}$,
- ii) $\lambda^\dagger \mathtt{xyz.e}(\mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{ks})\mathtt{x})\mathtt{y})\mathtt{z}) = \lambda^\dagger \mathtt{xyz.e}(\mathtt{s}(\mathtt{sxz})(\mathtt{syz}))$,
- v) $\lambda^\dagger \mathtt{xy.e}(\mathtt{s}(\mathtt{kx})(\mathtt{ky})) = \lambda^\dagger \mathtt{xy.e}(\mathtt{k}(\mathtt{xy}))$,
- vi) $\lambda^\dagger \mathtt{x.e}(\mathtt{s}(\mathtt{kx})\mathtt{i}) = \lambda^\dagger \mathtt{x.ex}$.

The equations of (L2) come from Theorem 4.4.3, and are thus numbered accordingly. By applying both sides of these equations by terms in $\mathcal{T}(\mathtt{X} \cup \underline{A})$, we obtain a new set of equations.

Corollary 4.6.15. *Let $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{i}, \mathtt{e})$ be a strongly reflexive combinatory pre-model. If the equations of (L1) and (L2) hold in A, then the following holds for all r, s, $t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$,*

i) $(\mathtt{s}(\mathtt{s}(\mathtt{kk}), s), t) \approx_{\mathtt{X}} (\mathtt{e}, s)$,

ii) $(\mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{ks}), r), s), t) \approx_{\mathtt{X}} (\mathtt{s}((\mathtt{s}, r), t), ((\mathtt{s}, s), t))$,

v) $((\mathtt{s}, (\mathtt{k}, s)), (\mathtt{k}, t)) \approx_{\mathtt{X}} (\mathtt{k}, (s, t))$,

vi) $((\mathtt{s}, (\mathtt{k}, t)), \mathtt{i}) \approx_{\mathtt{X}} (\mathtt{e}, t)$.

The equations of (CA), (L1) and (L2) are enough to characterize a stable strongly reflexive combinatory pre-model.

Theorem 4.6.16. *Let $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s}, \mathtt{i}, \mathtt{e})$ be a combinatory pre-model. Then A is stable and strongly reflexive if and only if it satisfies (CA), (L1) and (L2).*

Proof. ($\Leftarrow$). Let A be a stable strongly reflexive combinatory pre-model as in the statement. (L1) and (L2) follow directly from the definitions. (CA) follows from Proposition 4.6.9.
($\Rightarrow$). Let A be a combinatory pre-model as in the statement, that satisfies (CA), (L1) and (L2). We show that A is reflexive. Then, since (CA), (L1) and (L2) are closed equations, also $A[\mathtt{x}]$ is reflexive for any variable $\mathtt{x}$, and thus A is strongly reflexive.

To show that A is reflexive, we show that $s \approx_{\mathtt{x}} t$ implies $[\![\lambda^\dagger \mathtt{x}.s]\!]^{\underline{A}} = [\![\lambda^\dagger \mathtt{x}.t]\!]^{\underline{A}}$ for any s, $t \in \mathcal{T}(\{\mathtt{x}\} \cup \underline{A})$, using induction on the derivation of $\approx_{\mathtt{x}}$ (cf. Definition 4.1.3). Then A is reflexive by Proposition 4.3.13.

Let s, $t \in \mathcal{T}(\{\mathtt{x}\} \cup \underline{A})$, and define $p = \lambda^\dagger \mathtt{x}.s$ and $q = \lambda^\dagger \mathtt{x}.t$. Then $\lambda^\dagger \mathtt{x}.(((\mathtt{k}), s), t) \approx_{\mathtt{x}} (\mathtt{s}(\mathtt{s}(\mathtt{kk}), p), q)$ by the definition of $\lambda^\dagger$ and Lemma 4.6.7. From Corollary 4.6.15 we obtain that the latter is equal to $(\mathtt{e}, p)$, which is $\lambda^\dagger \mathtt{x}.s$ by Lemma 4.6.7. Then the equality $[\![\lambda^\dagger \mathtt{x}.(((\mathtt{k}), s), t)]\!]^{\underline{A}} = [\![\lambda^\dagger \mathtt{x}.s]\!]^{\underline{A}}$ follows as well.

The equation for $\mathtt{s}$ and the equation $(a, b) \approx_{\mathtt{x}} ab$ for a, $b \in \underline{A}$ (iii) and (i) of the definition of 1 (Definition 4.1.3) follow similarly as above, using equations (ii) and (i) of Corollary 4.6.15 respectively. The equations for $\mathtt{i}$ and $\mathtt{e}$ of the definition of 1 follow from the ones for $\mathtt{k}$ and $\mathtt{s}$ using (CA). □

We repeat Selinger's equations [38].

Theorem 3.5.12. *Let $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s})$ be a combinatory algebra. Then A s a lambda algebra if and only if it satisfies the following equations for all r, s, $t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$.*

1. $\mathbf{1}\mathtt{k} = \mathtt{k}$
2. $\mathbf{1}\mathtt{s} = \mathtt{s}$
3. $(\mathbf{1}, (\mathtt{k}, t)) \approx_{\mathtt{X}} (\mathtt{k}, t)$
4. $(\mathbf{1}, (\mathtt{s}, t)) \approx_{\mathtt{X}} (\mathtt{s}, t)$
5. $(\mathbf{1}, ((\mathtt{s}, s), t)) \approx_{\mathtt{X}} ((\mathtt{s}, s), t)$
6. $(\mathtt{s}(\mathtt{s}(\mathtt{kk}), s), t) \approx_{\mathtt{X}} (\mathbf{1}, s)$
7. $(\mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{ks}), r), s), t) \approx_{\mathtt{X}} (\mathtt{s}((\mathtt{s}, r), t), ((\mathtt{s}, s), t))$
8. $((\mathtt{s}, (\mathtt{k}, s)), (\mathtt{k}, t)) \approx_{\mathtt{X}} (\mathtt{k}, (s, t))$
9. $((\mathtt{s}, (\mathtt{k}, t)), \mathtt{skk}) \approx_{\mathtt{X}} (\mathbf{1}, t)$

We can discuss the origin of each of these axioms with the results from this section. The equations (1) until (5) follow from (CA) and (L1), while using Lemma 4.6.7. These equations thus characterize the stability of the lambda algebra, and ensure a fixed intierpretation of the combinators. The equations (6)-(9) follow from Corollary 4.6.15. These come from Theorem 4.4.3. When we look back into this chapter, we can say more about the origin of each of these axioms. The equations (6) and (7) of Selinger, or equivalently the equation (i) and (ii) from (L2) or Theorem 4.4.3 come from the equations for $\mathtt{k}$ and $\mathtt{s}$ in the combinatory pre-model. The equations (8) and (9), or (v) and (vi) were already discussed in Section 4.2.4.

The origin of Curry's equations has been discussed before by Krivine in [27], as was discussed in Remark 4.3.6. Still, is it worthwhile to also consider these equations again.

Theorem 3.2.6. *A combinatory algebra $A = (\underline{A}, \cdot, \mathtt{k}, \mathtt{s})$ is a lambda algebra if and only if it satisfies the following equations.*

1. $\mathtt{k} = [\![\lambda^*\mathtt{xy.kxy}]\!]^{\underline{A}}$
2. $\mathtt{s} = [\![\lambda^*\mathtt{xyz.sxyz}]\!]^{\underline{A}}$
3. $[\![\lambda^*\mathtt{xy.s(kx)(ky)}]\!]^{\underline{A}} = [\![\lambda^*\mathtt{xz.k(xy)}]\!]^{\underline{A}}$
4. $[\![\lambda^*\mathtt{xy.s(s(kk)x)y}]\!]^{\underline{A}} = [\![\lambda^*\mathtt{xyz.xz}]\!]^{\underline{A}}$
5. $[\![\lambda^*\mathtt{xyz.s(s(s(ks)x)y)z}]\!]^{\underline{A}} = [\![\lambda^*\mathtt{xyz.s(sxz)(syz)}]\!]^{\underline{A}}$

Equation (3) is similar to (v) from (L2), and equations (4) and (5) are similar to (i) and (ii) from (L2). As we discussed before, the equations (1) and (2) follow from stability.

Part II

Complexity

Chapter 5

Kolmogorov complexity

5.1 Preliminaries

5.1.1 Notation

In this part of the dissertation, we make use of the following conventions. Let ϵ be the empty string. For a string $x \in \{0,1\}^*$, we write $|x|$ for the length of x. For $n \in \mathbb{N}$, let $bin(n) \in \{0,1\}^*$ denote the binary representation of n, with the usual definition, where $bin(0) = 0$. For any $r \in \mathbb{R}_{>0}$, the value $\log r \in \mathbb{N}$ will denote the logarithm base two of r, rounded to the nearest integer above. Then for any $n \in \mathbb{N}_{\geq 1}$, we have that $|bin(n)| \leq \log n + 1$. For any $x, y \in \{0,1\}^*$, we denote xy for the concatenation of x with y. We fix a standard encoding for pairs. For any $x, y \in \{0,1\}^*$, define $\langle x, y \rangle = db(x)01y$, where db is the function that doubles every bit (e.g., $db(010) = 001100$). Thus,

$$|\langle x, y \rangle| = 2|x| + |y| + 2. \tag{5.1}$$

5.1.2 Dovetailing

Dovetailing is a technique that is often used in algorithms. It is a way to linearly execute several parallel processes. Let M be a Turing machine and let $x_0, x_1, x_2, \ldots$ be a possibly infinite enumeration of binary strings. This enumeration could be anything (but is usually the lexicographical ordering of strings). The machine M can now use dovetailing on this enumeration for different purposes. As an example, let y be any string. When we say that a Turing machine N uses dovetailing to look for an $i \in \mathbb{N}$ such that $M(x_i) = y$, we mean that N repeats the following process for $j = 0, 1, 2, \ldots$.

> For $i \in \{0, \ldots, j\}$, run M on input x_i for $j + 1$ steps. If $M(x_i) = y$ within that time, output x_i.

If the enumeration $x_0, x_1, x_2, \ldots$ is finite, then we can include in the description of the process that N outputs 0 when such a string cannot be found.

Note that if there is an $i \in \mathbb{N}$ such that $M(x_i) = y$, then with dovetailing this string can always be found.

5.1.3 Prefix Kolmogorov complexity

In Section 2.4, we gave the definition of the Kolmogorov complexity of a string. If we only allow certain Turing machines M for the Kolmogorov complexity $C_M(x)$ for $x \in \{0,1\}^*$, then the Kolmogorov complexity will have some nicer properties (see [33]). We therefore introduce "self-delimiting" machines.

Definition 5.1.1. A *self-delimiting machine* is defined as a Turing machine with a separate input-tape, for which the corresponding head is read-only, and can only move from left to right. The following convention is used. When M is a self-delimiting machine, for any $x \in \{0,1\}^*$, the computation of M on input x is a *success*, denoted by $M(x)\downarrow$, when M halts while the input-tape-head is scanning the rightmost bit of x. Otherwise, the computation is a *failure*, denoted by $M(y)\uparrow$.

We assume that for any Turing machine, the computation starts when the head, that is on the same tape as the input, is scanning an empty cell immediately on the left of the first bit of the input. We thus have that the above definition ensures that when M is self-delimiting, then the set X of $x \in \{0,1\}^*$ such that $M(x)\downarrow$ is *prefix-free*: for all $x, y \in X$, the string x is not a proper prefix of y. The *prefix Kolmogorov complexity* of a string x is now defined as follows:

Definition 5.1.2. For any $x \in \{0,1\}^*$ and any self-delimiting machine M, the *prefix Kolmogorov complexity* of x relative to M is defined as

$$K_M(x) = \min\{|p| : p \in \{0,1\}^*,\ M(p)\downarrow \text{ and } M(p) = x\},$$

if there is a $p \in \{0,1\}^*$ such that $M(p)\downarrow$ and $M(p) = x$, and otherwise $K_M(x) = \infty$.

The time-bounded version is defined as follows.

Definition 5.1.3. For any $x \in \{0,1\}^*$, any self-delimiting machine M and any $t \in \mathbb{N}$, the *time-bounded prefix Kolmogorov complexity* of x relative to M and t is defined as

$$K^t_M(x) = \min\{|p| : p \in \{0,1\}^*,\ M(p)\downarrow,\ M(p) = x \text{ and } T_M(p) \leq t\},$$

if there is a $p \in \{0,1\}^*$ such that $M(p)\downarrow$, $M(p) = x$ and $T_M(p) \leq t$, and otherwise $K^t_M(x) = \infty$.

Sometimes we misuse notation and write $K(n)$ for $K(bin(n))$ for some $n \in \mathbb{N}$.

We define a specific machine, and usually consider the prefix Kolmogorov complexity relative to that machine. First, for any $n \in \mathbb{N}$, denote 1^n for the string where 1 is repeated n times. We let $\bar{\cdot}$ be a function that encodes a string by first giving its length. That is, when for an $x \in \{0,1\}^*$ we have $|x| = n$, then $\overline{x} = 1^{|bin(n)|}0bin(n)x$. Note that then

$$|\overline{x}| \leq n + 2\log n + 3. \tag{5.2}$$

Definition 5.1.4. Define V to be a Turing machine with the following characteristics.

1. The machine is self-delimiting.

2. The machine is *universal*, which for self-delimiting machines is defined as follows. For any self-delimiting Turing machine M, there exists an $\alpha_M \in \{0,1\}^*$, the *binary representation* of M, such that for all x, whenever $M(x) \downarrow$ then $V(\langle \alpha_M, x\rangle) \downarrow$ and $V(\langle \alpha_M, x\rangle) = M(x)$. When M keeps computing forever on input x or $M(x) \uparrow$, then similarly for $V(\langle \alpha_M, x\rangle)$.

3. The machine is *time-efficient*. This means that for any self-delimiting machine M there exists a $c_M \in \mathbb{N}$ such that $T_V(\langle \alpha_M, x\rangle) \leq c_M T_M(x) \log T_M(x)$ for all x for which $M(x) \downarrow$, where α_M is the binary representation of M as described above.

4. The machine can *efficiently print* strings. This means that there exist an $\alpha \in \{0,1\}^*$ and $a, b, c \in \mathbb{N}$ such that $V(\langle \alpha, \overline{x}\rangle) \downarrow$ and $V(\langle \alpha, \overline{x}\rangle) = x$ and $T_V(\langle \alpha, \overline{x}\rangle) = a \cdot |x| + b$ for any $x \in \{0,1\}^* \backslash \epsilon$, where $|\langle \alpha, \overline{x}\rangle| \leq |x| + 2\log|x| + c$ because of equation (5.1).

5. The machine can *efficiently restart computations*. That is, there exists a string $r \in \{0,1\}^*$, such that $V(\langle r, pq\rangle) \downarrow$ and $V(\langle r, pq\rangle) = x$ for any p, q, $x \in \{0,1\}^*$ with $V(p) \downarrow$, $V(\langle q, V(p)\rangle) \downarrow$ and $V(\langle q, V(p)\rangle) = x$. Moreover, there exists a $c \in \mathbb{N}$ such that $T_V(\langle r, pq\rangle) = T_V(p) + T_V(\langle q, V(p)\rangle) + c$ for any such p, $q \in \{0,1\}^*$ for which $V(p) \downarrow$ and $V(\langle q, V(p)\rangle) \downarrow$.

The existence of a machine that satisfies properties 1, 2 and 3 is a known result. It is clear that the requirements 4 and 5 can be met additionally. The above ensures that the machine V is *optimal* for the prefix Kolmogorov complexity, in the following way.

Corollary 5.1.5. *For any self-delimiting Turing machine M, there exists a constant c_M such that for all $x \in \{0,1\}^*$ and $t \in \mathbb{N}$ for which $K_M^t(x) < \infty$,*

$$K_V^{c_M t \cdot \log t + c_M}(x) \leq K_M^t(x) + c_M.$$

From now on, we will shorten notation and write $K(x)$ instead of $K_V(x)$, and similarly for $K^t(x)$.

5.1.4 Comparing self-delimiting and prefix machines

It is not always necessary to consider self-delimiting machines. Sometimes it is even unavoidable to consider other kinds of machines, since the algorithms for self-delimiting machines have to work in a specific way, namely, only reading the input from left to right.

We will show later (cf. Theorem 5.1.8) that self-delimiting machines can simulate the computations of certain other machines as well. For this purpose we introduce a specific kind of Turing machines called *prefix machines*.

Definition 5.1.6. A *prefix machine* is a Turing machine M such that the set X, of $x \in \{0,1\}^*$ for which M halts on input x, is prefix-free.

One can ask themselves why we do not use prefix-machines to define the prefix Kolmogorov complexity of a string. The reason is that with prefix-machines, a result such as Corollary 5.1.5 cannot be derived. See [21] for more on this.

Since the machine V from Definition 5.1.4 can only simulate other self-delimiting machines, we introduce a specific universal Turing machine that can simulate other machines (such as prefix machines) as well.

Definition 5.1.7. Define U to be a Turing machine with the following characteristics.

1. The machine is a *universal Turing machine*, which is defined as follows. For any Turing machine M, there exists an $\alpha_M \in \{0,1\}^*$, called the *binary representation* of M, such that for all $x \in \{0,1\}^*$, whenever M halts on input x then $U(\langle \alpha_M, x \rangle) = M(x)$, and when M keeps computing forever then so does U.

2. Moreover, the machine is *time-efficient*. This means that for all machines M there exists a $c_M \in \mathbb{N}$ such that $T_U(\langle \alpha_M, x \rangle) \leq c_M T_M(x) \log T_M(x)$ for all x for which M halts on input x, where α_M is the binary representation of M as described above.

It is well-known that a machine that satisfies both properties exists.

In the following theorem, we show that at the cost of adding a constant number of bits to the program length, a self-delimiting machine can simulate prefix machines as well.

Theorem 5.1.8. *There exists a self-delimiting machine M, such that for any prefix machine N, for any $x \in \{0,1\}^*$, when N halts on input x, then $M(\langle \alpha_N, x \rangle) \downarrow$ and $M(\langle \alpha_N, x \rangle) = N(x)$, where α_N is the binary representation of N as in Definition 5.1.7.1.*

The proof comes from [11, Thm. 2.1].

Proof. We define M. It first checks whether its input is of the form $\langle \alpha, x \rangle = db(\alpha)01x$ for some $\alpha, x \in \{0,1\}^*$. It does so by starting to read the input, while also copying the contents on another tape. If M reads only doubled bits and then 01, the input is of the right form and M stops reading with the head of the input tape still on the 1, after having copied everything before the bits 01. Otherwise, M keeps computing forever.

In the following, assume that M has an input $\langle \alpha, x \rangle$ for some $\alpha, x \in \{0,1\}^*$. Also assume that, by checking the input as described above, M has copied $db(\alpha)$ on a separate tape, and the head of the input tape is on the 1 of the 01 after $db(\alpha)$. Then, M defines a string $x' = \epsilon$ and a set of natural numbers $S = \emptyset$. For the the rest of the computation, the machine repeats the following process.

> For $i = 0, 1, \ldots$, with $i \notin S$, for $j \in \{0, \ldots, i\}$, the machine M simulates the computation of U on input $\langle \alpha, z_j \rangle$ for $i+1$ steps, where U is the machine from Definition 5.1.7 and $z_0, z_1, \ldots$ is the lexicographical ordering of all binary strings, until U halts on an input $\langle \alpha, z_j \rangle$ for a j (M uses

dovetailing). Then M checks the following. If x' is equal to z_j, then M outputs $U(\langle \alpha, z_k \rangle) = U(\langle \alpha, x' \rangle)$. If x' is a proper prefix of z_j, then M reads one more bit of the input and redefines x' by appending that bit. If x' is not equal to z_j and also not a proper prefix of z_j, then M redefines S by adding j to it.

We argue that M satisfies the requirements of the statement. By construction, the machine M is self-delimiting. Let N be a prefix-machine. For an $x \in \{0,1\}^*$, assume that N halts on input x, and consider the computation of M on input $\langle \alpha_N, x \rangle$, where α_N is the binary representation of N as in Definition 5.1.7.1. To finish the proof of the statement, we show that M outputs $N(x)$ and $M(\langle \alpha_N, x \rangle) \downarrow$. Assume that during the computation M has already copied α_N on the worktape and that M started the process described above. During the process, bits of x get added to x'. Therefore, at any point in the rest of the computation, there are two cases for the string x': either x' is a proper prefix of x, or x' is equal to x.

First consider the case that x' is a proper prefix of x. Since N halts on x by assumption, and since N is a prefix machine, it is not possible that N halts on x'. However, N musts halt on a string z_j for some $j \in \mathbb{N}$, for which x' is a proper prefix of z_j, as is the case for $z_j = x$. After simulating U on input $\langle \alpha_N, z_j \rangle$, the machine M will read an extra bit of x and append this bit to x'. Then x' is either still a proper prefix of x, or x' is equal to x.

Now consider the case that x' is equal to x. Since N is a prefix machine and halts on x by assumption, it is not possible that N halts on a string of which x is a proper prefix. And thus M eventually simulates and outputs $U(\langle \alpha_N, x \rangle)$. Since M has read the entire string $x' = x$ of the input, the head of the input tape is on the last bit of the input, and $M(\langle \alpha_N, x \rangle) \downarrow$ holds. □

5.2 Algorithmic probability

Kolmogorov complexity is not only used for describing the information content of a string. Solomonoff introduced Kolmogorov complexity for his work on probability theory. This application will also be useful for defining the physical complexity of strings.

By considering all programs that produce a certain string, a universal probability of that string can be formulated. The presentation below comes from [33].

Definition 5.2.1. For any $x \in \{0,1\}^*$, the *algorithmic (universal a priori) probability* of x is

$$Q_V(x) = \sum_{\substack{p \in \{0,1\}^*,\ V(p)\downarrow \\ V(p)=x}} 2^{-|p|}.$$

For any $x \in \{0,1\}^*$ and any $t \in \mathbb{N}$, the *time-bounded algorithmic probability* of x relative to t is

$$Q_V^t(x) = \sum_{\substack{p \in \{0,1\}^*,\ V(p)\downarrow \\ V(p)=x,\ T_V(p) \leq t}} 2^{-|p|},$$

or $Q_V^t(x) = \infty$ when such p does not exist.

Thus, for any string x, the value of $Q_V(x)$ is the probability of V producing x on an input where each bit has an equal chance of being 0 or 1. Since V is a prefix machine, the Kraft inequality tells us that the value of $Q_V(x)$ is bounded by 1.

From the definition it is obvious that $2^{-K(x)} \leq Q_V(x)$ for all $x \in \{0,1\}^*$. There is the Coding theorem [11, 30] for the other direction.

Theorem 5.2.2. *There exists a $c \in \mathbb{N}$ such that for any $x \in \{0,1\},^*$*

$$Q_V(x) < 2^{-(K(x)-c)}.$$

In several intermediate steps, we eventually present the proof as it was given by Chaitin in [11]. This is done by showing that there exists a $c \in \mathbb{N}$ such that $K(x) < -\log Q_V(x) + c$ for all $x \in \{0,1\}^*$. In order to prove this, for any $x \in \{0,1\}^*$ we show that there exists a program that has a length less than $-\log Q_V(x) + c$ and computes x. This is done in the following steps. First, notice that we can use dovetailing to enumerate pairs (x, n) with $x \in \{0,1\}^*$ and $n \in \mathbb{N}$ such that $n \leq -\log Q_V(x)$. We then show that there exists a machine that, given a program that does the enumeration and a program of length n, can give the corresponding x, for all such pairs (x, n).

We first formally define how such an enumeration of pairs works.

Definition 5.2.3. Let $P \subseteq \{0,1\}^*$ and let $r \in \{0,1\}^*$. The set P is *recursively enumerated* by r, if for all $y \in \{0,1\}^*$, it holds that $U(\langle r, y\rangle) = 1$ if $y \in P$, and otherwise U keeps computing forever on input $\langle r, y\rangle$.

Let $S \subseteq \{0,1\}^* \times \mathbb{N}$ and let $r \in \{0,1\}^*$. The set S is *recursively enumerated* by r, if for all $y \in \{0,1\}^*$, it holds that $U(\langle r, y\rangle) = 1$ if $y = \langle x, bin(n)\rangle$ for an $(x, n) \in S$, and otherwise U keeps computing forever on input $\langle r, y\rangle$.

Occasionally it is useful to use a different representation of a recursive enumeration. The following lemma shows how to go between recursively enumerable as defined in Definition 5.2.3, and enumerating a set as a sequence.

Lemma 5.2.4. *There exists a Turing machine R, such that for each $S \subseteq \{0,1\}^* \times \mathbb{N}$ and each $r \in \{0,1\}^*$ that recursively enumerates S, the following holds.*

1. *For each $(x, n) \in S$ there exists a unique $i \in \mathbb{N}$ such that R halts on input $\langle r, bin(i)\rangle$ and $R(\langle r, bin(i)\rangle) = \langle x, bin(n)\rangle$.*

2. *For each $i \in \mathbb{N}$ and $y \in \{0,1\}^*$ such that R halts on input $\langle r, bin(i)\rangle$ and $R(\langle r, bin(i)\rangle) = y$, it holds that $y = \langle x, bin(n)\rangle$ for some $(x, n) \in S$.*

3. *When R halts on input $\langle r, bin(i)\rangle$ and $R(\langle r, bin(i)\rangle) = x$ for some $i \in \mathbb{N}$ and $x \in \{0,1\}^*$, then for each $j \leq i$ there exists an $y \in \{0,1\}^*$ such that $R(\langle r, bin(j)\rangle) = y$.*

The above is a well-known result, and therefore we only give the proof-sketch.

Proof sketch. Let S and r be as in the statement of the lemma. Since r recursively enumerates S, for all $y \in \{0,1\}^*$, we have $U(\langle r, y\rangle) = 1$ if and only if $y = \langle x, bin(n)\rangle$ for an $(x, n) \in S$.

Let $i \in \mathbb{N}$. The computation of R on input $\langle r, bin(i)\rangle$ is as follows. R defines an integer $j = i$. R will simulate $U(\langle r, \langle x, bin(n)\rangle\rangle)$ for all pairs $(x, n) \in \{0,1\}^* \times \mathbb{N}$, using dovetailing. Whenever $U(\langle r, \langle x, bin(n)\rangle\rangle) = 1$ for some $(x, n) \in \{0,1\}^* \times \mathbb{N}$, then R will redefine j by subtracting 1.

If at any point $j = 0$, then R outputs the last $\langle x, bin(n)\rangle$ that it found to be a member of S (that is, for which $U(\langle r, \langle x, bin(n)\rangle) = 1$). □

We need another definition and lemma.

Definition 5.2.5. For every $r \in \{0,1\}^*$ that enumerates a nonempty $S \subseteq \{0,1\}^* \times \mathbb{N}$, recursively define a sequence q_0^r, $q_1^r, \ldots, q_{|S|}^r$ as follows.

Let q_0^r be the lexicographic smallest string of length n_0, where $n_0 \in \mathbb{N}$ is such that $R(\langle r, bin(0)\rangle) = (x, n_0)$ for some $x \in \{0,1\}^*$.

For $i \leq |S| - 1$, let $n_i \in \mathbb{N}$ such that $R(\langle r, bin(i)\rangle) = (x, n_i)$ for some $x \in \{0,1\}^*$. Let q_i^r be the lexicographically smallest string of length n_i such that $\{q_0^r, \ldots, q_{i-1}^r\}$ is prefix-free, if such a string exists, and otherwise $q_i^r = \epsilon$.

Note that if $q_i^r = \varepsilon$ for an $r \in \{0,1\}^*$ that enumerates a nonempty $S \subseteq \{0,1\}^* \times \mathbb{N}$ and for an $i < |S|$, then $q_j^r = \varepsilon$ for all j, $i \leq j \leq |S|$.

We want to show that for an $r \in \{0,1\}^*$ that enumerates a nonempty $S \subseteq \{0,1\}^* \times \mathbb{N}$ with $\sum_{(x,n)\in S} 2^{-n} \leq 1$, for all $i \leq |S|$, we have $q_i^r \neq \varepsilon$. For this we need the following lemma.

Lemma 5.2.6. *Let $S \subseteq \{0,1\}^* \times \mathbb{N}$ be a nonempty set, and let $r \in \{0,1\}^*$ be a program that recursively enumerates it. For any $m \in \mathbb{N}$ with $m \leq |S|$ for which $q_m^r \neq \varepsilon$, there exists an $M \in \mathbb{N}$ and a sequence of strings $z_1, \ldots, z_M$ with $z_1 > z_2 > \cdots > z_M$ and $|z_1| < |z_2| < \cdots < |z_M|$, such that for any string z, the set $\{z\} \cup \{q_0^r, \ldots, q_m^r\}$ is prefix-free if and only if z has one of $z_1, \ldots, z_M$ as a prefix.*

Proof. In everything that follows, for each $i \in \mathbb{N}$ with $i \leq |S|$, let $x_i \in \{0,1\}^*$ and $n_i \in \mathbb{N}$ be such that $R(\langle r, bin(i)\rangle) = \langle x_i, bin(n_i)\rangle$. For each $n \in \mathbb{N}$, let the string 0^n denote a sequence of n zeroes.

We prove this claim by induction on m. First, let $m = 0$. Then the string q_0^r is equal to 0^{n_0}. Note that for any string z, we have that $\{z, q_0^r\}$ is prefix-free if and only if z has a prefix from 1, 01, 001, …, $0^{n_0-1}1$.

Now assume that the lemma holds for some $m \in \mathbb{N}$ with $m \leq |S| - 1$. By the induction hypothesis, there exists an $M \in \mathbb{N}$ and a sequence of strings $z_1, \ldots, z_M$, with $z_1 > z_2 > \cdots > z_M$ and $|z_1| < |z_2| < \cdots < |z_M|$, such that for any string z the set $\{z\} \cup \{q_1^r, \ldots, q_m^r\}$ is prefix-free if and only if z has a prefix from $z_1, \ldots, z_M$.

And thus, if $q_{m+1}^r \neq \varepsilon$, then there exists a $j \in \{1, \ldots, M\}$ such that z_j is a prefix of q_{m+1}^r. By minimality, the string q_{m+1}^r is equal to $z_j 0^{n_{m+1}-|z_j|}$.

For any string z, the set $\{z\} \cup \{q_0^r, \ldots, q_{m+1}^r\}$ is prefix-free, if and only if z has either a string from $z_1, \ldots, z_{j-1}, z_{j+1}, \ldots, z_M$ as a prefix, or a string that has z_j as a prefix but not $z_j 0^{n_{m+1}-|z_j|}$ as a prefix. We argue that we can redefine these strings as a sequence that satisfies the requirements from the statement.

Namely, we define a new sequence of strings $y_1, \ldots, y_{M'}$ for an $M' \in \mathbb{N}$ as $z_1, \ldots, z_M$ where z_j is replaced by $z_j 1$, $z_j 01$, …, $z_j 0^{n_{m+1}-|z_j|-1}1$, while keeping that order. That is, the new sequence will be

$$z_1, \ldots, z_{j-1},\ z_j 1,\ z_j 01,\ \ldots,\ z_j 0^{n_{m+1}-|z_j|-1}1,\ \ z_{j+1}, \ldots, z_M.$$

This can be defined more formally as follows. Let $M' = M + n_{m+1} - |z_j| - 1$. For $i \in \mathbb{N}$, if $i \leq j-1$, then $y_i = z_i$, if $j \leq i \leq j + n_{m+1} - |z_j| - 1$, then $y_i = z_j 0^{j-i} 1$, and if $j + n_{m+1} - |z_j| \leq i \leq M'$, then $y_i = z_{i - n_{m_1} + |z_j| + 1}$.

Since $z_1 > z_2 > \cdots > z_M$, we have for all $i \in \{j+1, \ldots, M'\}$ that $|z_i| > n_{m+1}$, because otherwise, when $|z_i| \leq n_{m+1}$, the machine could've chosen for q^r_{m+1} a string with z_i as a prefix, which contradicts with the minimality of q^r_{m+1}. This gives us that $y_1 > y_2 \ldots > y_{M'}$ and $|y_1| < |y_2| < \ldots < |y_{M'}|$. □

We can now show the following.

Lemma 5.2.7. *Let $S \subseteq \{0,1\}^* \times \mathbb{N}$ be a nonempty set, and let $r \in \{0,1\}^*$ be a program that recursively enumerates it. If $\sum_{(x,n)\in S} 2^{-n} \leq 1$, then $q^r_i \neq \varepsilon$ for each $i \in \mathbb{N}$ with $i \leq |S|$.*

Proof. In everything that follows, for each $i \in \mathbb{N}$ with $i \leq |S|$, let $x_i \in \{0,1\}^*$ and $n_i \in \mathbb{N}$ be such that $R(\langle r, bin(i)\rangle) = \langle x_i, bin(n_i)\rangle$. For each $n \in \mathbb{N}$, let the string 0^n denote a sequence of n zeroes.

We will prove the lemma by contradiction. Let S and r be as in the statement of the lemma. Let $m \in \mathbb{N}$ be the smallest number such that $m \leq |S|$ and $q^r_m = \varepsilon$. We will argue that Lemma 5.2.6 implies that $\sum_{(x,n)\in S} 2^{-n} > 1$.

Lemma 5.2.6 implies that there exists an $M \in \mathbb{N}$, and a sequence of strings $z_1, \ldots, z_M$ with $|z_1| < |z_2| < \cdots < |z_M|$, such that any string z for which $\{z\} \cup \{q^r_0, \ldots, q^r_{m-1}\}$ is prefix-free has a prefix from $z_1, \ldots, z_M$. First note that for all $i \in \{1, \ldots, M\}$ we have

$$|z_i| > n_m, \tag{5.3}$$

because otherwise $z_i 0^{n_m - |z_i|}$ is a string of length n_m for which $\{z_i 0^{n_m - |z_i|}\} \cup \{q^r_0, \ldots, q^r_{m-1}\}$ is prefix-free, which contradicts that $q^r_m = \varepsilon$.

Let $N \in \mathbb{N}$ be any number such that for all $i \in \{1, \ldots, m\}$, we have $N \geq n_i$. That is, N is bigger then or equal to n_m and the length of any element from $\{q^r_0, \ldots, q^r_{m-1}\}$.

From equation 5.3 and $|z_1| < |z_2| < \cdots < |z_M|$ we can deduce

$$2^{N-n_m} > \sum_{i \leq M} 2^{N-|z_i|}. \tag{5.4}$$

Define $Q_N = \{q \in \{0,1\}^* \mid \{q\} \cup \{q^r_0, \ldots, q^r_{m-1}\}$ is prefix free and $|q| = N\}$.

There are $\sum_{i<m} 2^{N-n_i}$ strings of length N that have one of $\{q^r_0, \ldots, q^r_{m-1}\}$ as a prefix. And thus,

$$|Q_N| = 2^N - \sum_{i<m} 2^{N-n_i}.$$

Moreover, by Lemma 5.2.6, the strings $q \in Q_N$ are the ones that have one of $z_1, \ldots, z_M$ as a prefix. Thus,

$$|Q_N| = \sum_{i \leq M} 2^{N-|z_i|}$$

This gives us that

$$\sum_{i \leq M} 2^{N-|z_i|} = 2^N - \sum_{i<m} 2^{N-n_i}. \tag{5.5}$$

From equations 5.4) and 5.5), we have

$$2^{N-n_m} > 2^N - \sum_{i<m} 2^{N-n_i}.$$

After dividing everything by 2^N, we obtain

$$2^{-n_m} > 1 - \sum_{i<m} 2^{-n_i},$$

which implies $\sum_{i\leq m} 2^{-n_i} > 1$, the desired contradiction. □

Recall the enumeration of pairs (x, n) with $x \in \{0,1\}^*$ and $n \in \mathbb{N}$, such that $n \leq -\log Q_V(x)$, that was discussed before. The following lemma shows the existence of a machine that, given a program that recursively enumerates those pairs, and a program of length n, outputs the corresponding x.

Lemma 5.2.8. *There exists a machine Q that does the following. Let $S \subseteq \{0,1\}^* \times \mathbb{N}$ be a nonempty set, and let $r \in \{0,1\}^*$ be a program that recursively enumerates it. If $\sum_{(x,n)\in S} 2^{-n} \leq 1$, then for any $(x, n) \in S$, there exists a unique $p \in \{0,1\}^*$ such that $|p| = n$ and $Q(\langle r, p\rangle) = x$. Moreover, the set of these p for all elements of S is prefix-free.*

The proof of this lemma is a new representation of an older result, based on the proof of Theorem 3.2 in [11] and the proof of the Kraft inequality. Let a *prefix-code* for a set X be a set of strings Y, that is prefix-free, such that for every $x \in X$ there exists a unique $y \in Y$, called the *code-word* for x.

The lemma shows that for a program r and a set S as in the statement, and for an $(x, n) \in S$, there exists a unique $p \in \{0,1\}^n$ that can be used to find the string x. That is, this p is the code-word for x.

The proof of the Kraft inequality shows us that when creating a prefix-code for a set X, it is *optimal* to assign the code-words in increasing length, starting with the code-word that has the smallest length. By optimal it is meant that when assigning the code-words in this order, we can always find a prefix-free code when there is one. As we will see, the proof of this lemma shows that it is also optimal to assign the code-words in order of increasing value, starting with the code-word that is the smallest string in the lexicographical ordering.

Proof of Lemma 5.2.8. We define Q. The machine Q first checks whether its input is of the form $\langle r, p\rangle$ for some $r, p \in \{0,1\}^*$. If not, Q keeps computing forever. Otherwise, Q repeats the following process for rounds $i = 0, 1, 2, \ldots$.

> Q computes $R(\langle r, bin(i)\rangle)$, where R is the machine of Lemma 5.2.4, and checks whether it is equal to a string $\langle x, bin(n)\rangle$ for some $x \in \{0,1\}^*$ and $n \in \mathbb{N}$. If not, Q will continue to compute forever. Otherwise, Q defines $q_i \in \{0,1\}^*$ as the lexicographically smallest string of length n such that $\{q_0, \ldots, q_i\}$ is prefix-free, where $q_0, \ldots, q_{i-1}$ are the strings defined in previous rounds. If such a string does not exist, then Q continues to compute forever. If $q_i = p$, then Q outputs x and the computation halts.

For the rest of the proof, fix an $r \in \{0,1\}^*$ and an $S \subseteq \{0,1\}^* \times \mathbb{N}$ as in the statement of the lemma.

Let $i \in \mathbb{N}$ such that R halts on input $\langle r, bin(i)\rangle$, and let $p \in \{0,1\}^*$ such that Q machine reaches round i of the process. Then the q_i that the machine defines in this round, is the same as the q_i^r from Definition 5.2.5. Lemma 5.2.7 implies that the machine can always find this q_i.

Let $(x,n) \in S$, and let $i \in \mathbb{N}$ such that $R(\langle r, bin(i)\rangle) = \langle x, bin(n)\rangle$. Then, by construction, $Q(\langle r, q_i^r\rangle)$ halts and $Q(\langle r, q_i^r\rangle) = x$. The unique $p \in \{0,1\}^*$ from the lemma is thus equal to this q_i^r. Moreover, by definition, the set of all q_i^r for $i \leq |S|$ is prefix-free. □

We continue to the proof of the Coding Theorem:

Theorem 5.2.2. *There exists a $c \in \mathbb{N}$ such that for any $x \in \{0,1\},^*$*

$$Q_V(x) < 2^{-(K(x)-c)}.$$

We need to be precise regarding the logarithm. Thus, in the following, let $\log_2$ denote the logarithm base two, however not rounded to any integer. Thus, for $r \in \mathbb{R}_{>0}$, we have that $\log_2 r \in \mathbb{R}$.

Proof of Theorem 5.2.2. We will show the existence of a prefix machine N such that for all $x \in \{0,1\}^*$,

$$C_N(x) \leq \lceil -\log_2 Q_V(x)\rceil + 1. \tag{5.6}$$

Then the theorem holds for $c = 2|\alpha_M| + 2|\alpha_N| + 6$, where α_N is the binary representation of N and α_M is the binary representation of the self-delimiting machine from Theorem 5.1.8, where the binary representations are as in Def. 5.1.4.2. This c is sufficient to prove the statement, since now, for any $x \in \{0,1\}^*$ and witness p of $C_N(x)$ (that is, $N(p) = x$ and $|p| = C_N(x)$), we have that $V(\langle \alpha_M, \langle \alpha_N, p\rangle\rangle) = N(p) = x$, and the following holds by equation (5.1).

$$\begin{aligned} K(x) &\leq |\langle \alpha_M, \langle \alpha_N, p\rangle\rangle| \\ &= 2|\alpha_M| + 2|\alpha_N| + |p| + 4 \\ &\leq \lceil -\log_2 Q_V(x)\rceil + c - 1 \\ &< -\log_2 Q_V(x) + c. \end{aligned}$$

Let r be a program that recursively enumerates S (cf. Def. 5.2.3).

We now show that $\sum_{(x,n)\in S} 2^{-n} \leq 1$, which implies that S and r satisfy the requirements from Lemma 5.2.8. First note that for a fixed $x \in \{0,1\}^*$,

$$\sum_{n \geq \lceil -\log_2 Q_V(x)\rceil + 1} 2^{-n} = 2^{-(\lceil -\log_2 Q_V(x)\rceil)} \cdot \sum_{i \in \mathbb{N}_{\geq 1}} 2^{-i} = 2^{-\lceil -\log_2 Q_V(x)\rceil} < Q_V(x).$$

And thus

$$\sum_{(x,n)\in S} 2^{-n} = \sum_x \sum_{n \geq \lceil -\log_2 Q_V(x)\rceil + 1} 2^{-n} < \sum_x Q_V(x) \leq 1, \tag{5.7}$$

where the latter inequality follows from the Kraft inequality and the fact that that V is a prefix machine.

The definition of N is now as follows. Let Q be the Turing machine from Lemma 5.2.8. On an input p, the machine N computes and outputs $Q(\langle r, p\rangle)$.

From Lemma 5.2.8, it is clear that N is a prefix machine. We will now argue why N satisfies (5.6). Fix an $x \in \{0,1\}^*$ and let $n = \lceil -\log_2 Q_V(x) \rceil + 1$. Then $(x, n) \in S$, and by Lemma 5.2.8, there exists a $p \in \{0,1\}^*$ with $|p| = n$ such that $Q(r, p) = x$ and thus also $N(p) = x$. Thus, $C_N(x) \leq \lceil -\log_2 Q_V(x) \rceil + 1$. □

Chapter 6

Logical depth

In this chapter, we discuss Bennet's definition of logical depth, together with other related notions. It is based on joint work with Akitoshi Kawamura, as were some of the proofs in the previous chapter. We first outline the thought process of Bennet, considering different definitions leading up to the final definition of logical depth. We also discuss the relations between the different notions. After that we give results that outlines problems with the definition of logical depth .

6.1 Defining logical depth

Benett's intention was to capture the *value* of a message, which he describes as "the amount of mathematical or other work plausibly done by its originator, which its receiver is saved from having to repeat" [9, p. 230]. In this line of thought, the depth of a string could be thought of as *the computation time of the shortest program that produces the string*. This definition was discarded, since there could be a program only a few bits longer, but running much faster. If the difference in length is small, the faster program is perhaps more likely to have produced the string [9]. In order to solve this problem, a definition is introduced, where the difference in length to the shortest program is made variable, by introducing an extra parameter s called the *significance level*.

Definition 6.1.1 ([9, p. 240, Tentative Definition 0.2])**.** For any $x \in \{0,1\}^*$ and any $s \in \mathbb{N}_{\geq 1}$,

$$\mathrm{d}_s(x) = \min\{\, T_V(p) : p \in \{0,1\}^*,\ V(p)\downarrow,\ V(p) = x \text{ and } |p| - K(x) < s \,\},$$

Thus, for any string x, the value $\mathrm{d}_s(x)$ is the time required to compute x by a program less than s bits longer than the shortest program. In other words, $\mathrm{d}_s(x)$ is the minimum $t \in \mathbb{N}$ such that $K^t(x)$ (see Definition 5.1.3) is not ∞ and $K^t(x) - K(x) < s$. Therefore, Definition 6.1.1 is very similar to the following, which is Definition 3.1 in [4].

Definition 6.1.2. For any $x \in \{0,1\}^*$ and any $t \in \mathbb{N}$, the *basic computational depth* of x relative to t is the difference between the prefix Kolmogorov complexity and the time-bounded prefix Kolmogorov complexity of x relative to t:

$$\mathrm{bcd}_t(x) = K^t(x) - K(x).$$

If $K^t(x) = \infty$, then so is $\mathrm{bcd}_t(x)$.

The definitions d_s and bcd_t are actually just a change of parameter.

Corollary 6.1.3. *For any s, $t \in \mathbb{N}_{\geq 1}$ and any $x \in \{0,1\}^*$,*

$$\mathrm{d}_s(x) \leq t \qquad \textit{if and only if} \qquad \mathrm{bcd}_t(x) < s.$$

Results such as the existence of strings x for which $\mathrm{bcd}_t(x)$ and thus also $\mathrm{d}_s(x)$ is high, and applications of this definition to average time-complexity, can be found in [4].

Note that for all x, the value $\mathrm{d}_s(x)$ is defined for any $s \in \mathbb{N}_{\geq 1}$, and that $\mathrm{d}_s(x)$ is non-increasing in s (that is, $\mathrm{d}_s(x) \geq \mathrm{d}_{s'}(x)$ for any s, $s' \in \mathbb{N}$ with $0 < s \leq s'$).

Remark 6.1.4. On one end, $\mathrm{d}_1(x)$ is the running time of a shortest program for x; on the other end, $\mathrm{d}_s(x)$ is small for large s: there exist a, b and $c \in \mathbb{N}$ such that for all $x \in \{0,1\}^*$ and for all $s \geq |x| + 2\log|x| + c - K(x)$, we have $\mathrm{d}_s(x) \leq a|x| + b$ because of Definition 5.1.4.4.

We return to the attempts of Bennett for finding a definition for the value of a message. Definition 6.1.1 was also dismissed, because of the way it treats multiple programs of the same length [9, p. 240]. Subsequently, Bennett introduces the definition below.

Definition 6.1.5 ([9, p. 240, Tentative Definition 0.3])**.** For any $x \in \{0,1\}^*$ and any $s \in \mathbb{N}_{\geq 1}$,

$$\mathrm{ldepth}_s(x) = \min\left\{ t \in \mathbb{N} : \frac{Q_V^t(x)}{Q_V(x)} \geq 2^{-s} \right\}.$$

Thus, for any $x \in \{0,1\}^*$ and $s \in \mathbb{N}_{\geq 1}$, the value $\mathrm{ldepth}_s(x)$ is the time t required for x's time-bounded algorithmic probability $Q_V^t(x)$ to rise within a factor 2^{-s} of $Q_V(x)$. Again, $\mathrm{ldepth}_s(x)$ is non-increasing in s.

In [5, Th. 3.5], we can find a result that relates ldepth_s to bcd_t (and hence, by Corollary 6.1.3, to d_s). We suspect that the proof is incorrect, and propose a different statement than the one described there.

Theorem 6.1.6. *There exists a $c \in \mathbb{N}$ such that for any $x \in \{0,1\}^*$ and any $s \in \mathbb{N}_{\geq 1}$,*

$$\mathrm{ldepth}_{s+c}(x) \leq \mathrm{d}_s(x).$$

Proof. Let c be as in the Coding Theorem (Th. 5.2.2), such that $2^{-(K(y)-c)} > Q_V(y)$ for all $y \in \{0,1\}^*$. Fix an $x \in \{0,1\}^*$ and $s \in \mathbb{N}_{\geq 1}$, and let $t = \mathrm{d}_s(x)$, so that $K^t(x) - K(x) < s$. Then $\mathrm{ldepth}_{s+c}(x) \leq t$, because

$$\frac{Q_V^t(x)}{Q_V(x)} > \frac{2^{-K^t(x)}}{2^{-(K(x)-c)}} = 2^{-(K^t(x)-K(x)+c)} > 2^{-(s+c)}. \qquad \square$$

In [9], Definition 6.1.5 is shown (see Theorem 6.2.11 below) to be almost equivalent to the following, on which Bennett settles.

Definition 6.1.7 ([9, p. 241, Definition 1])**.** For any $x \in \{0,1\}^*$ and any $s \in \mathbb{N}_{\geq 1}$, the *(logical) depth* of x at significance level s is the least time required for an s-incompressible program to compute x:

$$\text{depth}_s(x) = \min\{\, T_V(p) : p \in \{0,1\}^*,\ V(p)\downarrow,\ V(p) = x \text{ and } |p| - K(p) < s \,\},$$

or $\text{depth}_s(x) = \infty$ when such a p does not exist.

Bennett argues that logical depth is a measure of physical complexity, and he uses it to study the self-organization of systems [9, Section 5]. It is by far the most used definition of all definitions for complexity that are based on computation time. Besides the definitions mentioned before, another definition that uses computation time is the n-potent by Adleman [1]. Other authors occasionally use one of the definitions that Bennett discarded. For example, in [18] the computation time of the shortest program is used as the depth of a string, and d_s is used in [4, 3].

6.2 Properties of logical depth

6.2.1 Basic properties

In the standard way, the logical depth of a string is defined with respect to the program length of programs for the universal machine V. In order to justify this decision, it is necessary to establish how the logical depth of strings will differ when we use different Turing machines. Because of Corollary 5.1.5, for different universal machines the depth can differ up to a logarithmic term. We will formalize this statement. We first need the following lemma.

Lemma 6.2.1. *For any universal self-delimiting Turing machine M that satisfies the properties (1) - (5) from Definition 5.1.4, there exists a constant $c \in \mathbb{N}$ such that $K_M(y) \leq K_M(\langle x, y\rangle) + c$ for any $x, y \in \{0,1\}^*$.*

Proof. Let M be a universal self-delimiting machine that satisfies properties (1) - (5) from Definition 5.1.4. Let $r \in \{0,1\}^*$ be the restart program from property (5) in Definition 5.1.4. That is, $M(\langle r, pq\rangle)\downarrow$ and $M(\langle r, pq\rangle) = x$ for any p, q, $x \in \{0,1\}^*$ with $M(p)\downarrow$, $M(\langle q, \overline{M(p)}\rangle)\downarrow$ and $M(\langle q, \overline{M(p)}\rangle) = x$. Let N be a self-delimiting Turing machine such that $N(\langle x, y\rangle)\downarrow$ and $N(\langle x, y\rangle) = y$ for any x, $y \in \{0,1\}^*$. Let α_N be the binary representation of N (cf. property (2) of Definition 5.1.4), such that $M(\langle \alpha_N, x\rangle)\downarrow$ and $M(\langle \alpha_N, x\rangle) = N(x)$ for all $x \in \{0,1\}^*$ for which $N(x)\downarrow$. Define $c = 2|r| + |\alpha_N| + 2$.

We argue that $K_M(y) \leq K_M(\langle x, y\rangle) + c$ for any x, $y \in \{0,1\}^*$. Let x, $y \in \{0,1\}^*$. Let $p \in \{0,1\}^*$ be the witness of $K_M(\langle x, y\rangle)$, that is, $M(p)\downarrow$, $M(p) = \langle x, y\rangle$ and $|p| = K_M(\langle x, y\rangle)$. Then $M(\langle r, p\alpha_N\rangle)\downarrow$ and $M(\langle r, p\alpha_N\rangle) = y$. Therefore $K_M(y) \leq |\langle r, p\alpha_N\rangle| = 2|r| + |p| + |\alpha_N| + |2| = K_M(\langle x, y\rangle) + c$. □

It is necessary to define logical depth where another Turing machine than V is used.

Definition 6.2.2. For any self-delimiting machine M that satisfies the properties (1) - (5) from Definition 5.1.7, for any $x \in \{0,1\}^*$ and any $s \in \mathbb{N}_{\geq 1}$, the logical depth of x with respect to M at significance level s is

$$\text{depth}_s^M(x) = \min\{\, T_M(p) : p \in \{0,1\}^* \,,\, M(p)\downarrow \,,\, M(p) = x \text{ and } |p| - K(p) < s\},$$

or $\text{depth}_s^M(x) = \infty$ when such a p does not exist.

We can now give the difference between the logical depth of a string with respect to different Turing machines.

Proposition 6.2.3. *Let N and M be two machines satisfying properties (1) - (5) from Definition 5.1.7. There exist $c_1, c_2 \in \mathbb{N}$, dependent on N and M, such that for any $x \in \{0,1\}^*$ and any $s \in \mathbb{N}_{\geq 1}$,*

$$\text{depth}_{s+c_1}^M(x) \leq c_2 \text{depth}_s^N(x) \log(\text{depth}_s^N(x))$$

Proof. Let α_N be the binary representation of N (cf. Def. 5.1.4.2), such that $M(\langle \alpha_N, x\rangle)\downarrow$ and $M(\langle \alpha_N, x\rangle) = N(x)$ for all $x \in \{0,1\}^*$ for which $N(x)\downarrow$. Similarly, let α_M be the binary representation of M for machine N. Let $k \in \mathbb{N}$ be the constant from Lemma 6.2.1, such that $K_M(y) \leq K_M(\langle x, y\rangle) + k$ for any $x, y \in \{0,1\}^*$. Then, let $c_1 = k + 4|\alpha_N| + 4$ and let c_2 be the constant from property (3) from Definition 5.1.4, such that $T_M(\langle \alpha_N, x\rangle) \leq c_2 T_N(x) \log T_N(x)$ for all x for which $N(x)\downarrow$.

In the following, fix an $x \in \{0,1\}^*$. Let $p \in \{0,1\}^*$ such that $N(p)\downarrow$ and p is the witness of $\text{depth}_s^N(x)$ (that is, $N(p) = x$ and $|p| - K_N(p) < s$ and $T_N(p) = \text{depth}_s^N(x)$). Then also $M(\langle \alpha_N, p\rangle)\downarrow$ and $M(\langle \alpha_N, p\rangle) = x$, and we have that $T_M(\langle \alpha_N, p\rangle) \leq c_2 T_N(p) \log(T_N(p))$. In what follows, we will argue that $|\langle \alpha_N, p\rangle| - K_M(\langle \alpha_N, p\rangle) < s + c_1$. We can conclude that $\text{depth}_{s+c_1}^M(x) \leq c_2 T_N(p) \log(T_N(p))$.

Let q be such that $M(q)\downarrow$ and q is the witness of $K_M(p)$. Then $M(q) = p$ and $|q| = K_M(p)$. Note that $N(\langle \alpha_M, q\rangle) = M(q) = p$. Now, by equation 5.1,

$$K_N(p) \leq |\langle \alpha_M, q\rangle| = K_M(p) + 2|\alpha_N| + 2. \tag{6.1}$$

By Lemma 6.2.1, we have that $K_M(p) \leq K_M(\langle \alpha_N, p\rangle) + k$. Combining this with (6.1), we obtain

$$K_M(\langle \alpha_N, p\rangle) \geq K_N(p) - k - 2|\alpha_N| - 2. \tag{6.2}$$

We now have the following.

$$\begin{aligned}
|\langle \alpha_N, p\rangle| - K_M(\langle \alpha_N, p\rangle) &\leq |\langle \alpha_N, p\rangle| - K_N(p) + k + 2|\alpha_N| + 2 && \text{by equation (6.2)} \\
&\leq 4|\alpha_N| + |p| - K_N(p) + k + 4 && \text{by equation (5.1)} \\
&< 4|\alpha_N| + s + k + 4 && |p| - K_N(p) < s \\
&= s + c_1.
\end{aligned}$$

□

The depth of a string is always associated with a significance level. If for some $s \in \mathbb{N}_{\geq 1}$ and $x \in \{0,1\}^*$, the value of $\text{depth}_s(x)$ is large, then this can be described as that "each individual hypothesis for the rapid origin of x is implausible

at the 2^{-s} confidence level" [9, p. 241]. Being very specific about what s to pick or defining what constitutes as "large depth" is difficult. But we know that depth_s is non-increasing in s (for all $x \in \{0,1\}^*$, when $s \in \mathbb{N}_{\geq 1}$ increases, then the value $\mathrm{depth}_s(x)$ decreases). Thus, when Bennett discusses *shallow* objects (having low depth), he gives an example of a string that has a depth that is smaller than some low-order polynomial at some small significance level. For evidence of a deep string the significance level can be large [9]. See Theorem 6.3.5 for this.

Logical depth satisfies a *slow growth law*: deep strings cannot be quickly computed from shallow ones [9, Th. 1] .

Remark 6.2.4. As with d_s (cf. Remark 6.1.4), there exist a, b and $c \in \mathbb{N}$, such that $\mathrm{depth}_s(x) \leq a|x| + b$ for all $x \in \{0,1\}^*$ and all $s \geq |x| + 2\log|x| + c$.

A lower bound for the significance level s for which depth_s is not infinite is given later in Proposition 6.2.6, together with the relation of depth_s to d_s.

6.2.2 Comparing different definitions

In this section we present several results that relate depth_s to other definitions. We start with the somewhat obvious relation between d_s and depth_s. It also gives us a significance level for which depth_s is guaranteed to be not infinite. Before we prove the relation we introduce a specific Turing machine.

Lemma 6.2.5. *There exists a self-delimiting Turing machine R, such that for any $q \in \{0,1\}^*$, if there exists a $p \in \{0,1\}^*$ for which $V(q)\downarrow$, $V(q) = p$, and $V(p)\downarrow$, then $R(q) = V(p)$.*

Proof sketch. The existence of this Turing machine is not obvious, since it has to be self-delimiting, but it can be defined using a similar procedure as in Theorem 5.1.8 and Lemma 6.2.1. □

Proposition 6.2.6. *There is a $c \in \mathbb{N}$ such that $\mathrm{depth}_{s+c}(x) \leq \mathrm{d}_s(x)$ for all $s \in \mathbb{N}_{\geq 1}$ and $x \in \{0,1\}^*$. In particular, $\mathrm{depth}_{s+c}(x) < \infty$ for all $s \in \mathbb{N}_{\geq 1}$.*

Proof. Let R be the Turing machine from Lemma 6.2.5. Let $c = 2|\alpha_R| + 2$. Fix an $s \in \mathbb{N}_{\geq 1}$ and an $x \in \{0,1\}^*$. By the definition of $\mathrm{d}_s(x)$, there is a $p \in \{0,1\}^*$ such that $V(p) = x$, satisfying $T_V(p) = \mathrm{d}_s(x)$ and

$$|p| < K(x) + s. \tag{6.3}$$

To prove our claim, it suffices to show that the same program p satisfies

$$|p| < K(p) + s + c. \tag{6.4}$$

For this, let $q \in \{0,1\}^*$ be a shortest program for p (i.e., $V(q) = p$ and $|q| = K(p)$), and note that $V(\langle \alpha_R, q\rangle) = x$. From equation (5.1) we know that $|\langle \alpha_R, q\rangle| = |q| + c$. Hence, $K(x) \leq |q| + c = K(p) + c$, and thus (6.3) implies (6.4). □

For the relation between depth and ldepth we need some intermediate results. More specifically, we need more results in order to obtain a bound on depth assuming a bound on ldepth. In ldepth we have the time bounded algorithmic probability

$Q_V^t(x)$ for some $x \in \{0,1\}^*$ and $t \in \mathbb{N}$. It turns out, that all programs that output such a specific x within a time t, can be compressed. In order to show this, we first outline how such programs can be recursively enumerated. Then, using this enumeration and Lemma 5.2.8, we obtain that there is a unique way to identify any such program, given the enumeration. The latter implies that the programs can be compressed. In turn, the compression gives us a bound on the depth.

In what follows, we say that a $p \in \{0,1\}^*$ is a program for an $x \in \{0,1\}^*$ or a $t \in \mathbb{N}$, when $V(p) \downarrow$ and $V(p) = x$ or $V(p) = bin(t)$ respectively. For any $x \in \{0,1\}^*$ and $t \in \mathbb{N}$, let $P_{x,t} = \{p : p \in \{0,1\}^* \wedge V(p) \downarrow \wedge V(p) = x \wedge T_V(p) \leq t\}$. Then, $Q_V^t(x) = \sum_{p \in P_{x,t}} 2^{-|p|}$. We now start by showing how to enumerate programs in such a set (recall Definition 5.2.3).

Lemma 6.2.7. *There exists an $r \in \{0,1\}^*$ such that for any $x \in \{0,1\}^*$, any $t \in \mathbb{N}$, and all programs p_x, p_t for x and t respectively, $\langle r, p_x p_t \rangle$ recursively enumerates $P_{x,t}$.*

Proof. We show that there exists an $r \in \{0,1\}^*$, such that for any x, t, p_x and p_t as in the lemma, and for any $y \in \{0,1\}^*$, the machine U outputs 1 on input $\langle \langle r, p_x p_t \rangle, y \rangle$ if $y \in P_{x,t}$ and otherwise keeps computing forever.

For any z, $y \in \{0,1\}^*$, the computation of U on input $\langle \langle r, z \rangle, y \rangle$ starts with simulating computations of V on all prefixes of z, using dovetailing, until either $V(z_0) \downarrow$ for some prefix z_0 or the machine keeps computing forever. Let the rest of the string z be z_1, that is, $z = z_0 z_1$. Then, it also simulates V on input z_1. It keeps computing forever if not $V(z_1) \downarrow$. Subsequently, the machine computes V on input y. If $V(y) \downarrow$ and $V(y) = V(z_0)$ and $T_V(y) \leq V(z_1)$, it outputs 1, and otherwise keeps computing forever.

Now let $x \in \{0,1\}^*$, let $t \in \mathbb{N}$ and let p_x, p_t be any programs for x and t respectively. Then $z_0 = p_x$ and $z_1 = p_t$ during the computation of U on input $\langle \langle r, p_x p_t \rangle, y \rangle$. The machine outputs 1 only when $y \in P_{x,t}$. □

Corollary 6.2.8. *There exists an $r \in \{0,1\}^*$ such that for any $x \in \{0,1\}^*$, any $t \in \mathbb{N}$, and all programs p_x, p_t for x and t respectively, $\langle r, p_x p_t \rangle$ recursively enumerates the set $\{(p, bin(|p| - m)) \mid p \in P_{x,t}\}$, where $m \in \mathbb{N}$ is maximum such that $\sum_{p \in P_{x,t}} 2^{-|p|} \leq 2^{-m}$.*

Proof. Since for any $x \in \{0,1\}^*$ and $t \in \mathbb{N}$, the set $P_{x,t}$ is finite, the number m is computable. The result then follows from Lemma 6.2.7. □

Now that we have algorithms for the enumeration, we can show how to uniquely identify the programs in the enumeration.

Lemma 6.2.9. *There exists a prefix machine N, such that for any $x \in \{0,1\}^*$ and $t \in \mathbb{N}$, for any $p \in P_{x,t}$, there exists an $x_p \in \{0,1\}^*$, such that for all programs p_x, p_t for x and t respectively, $N(p_x p_t x_p) = p$, where $|x_p| = |p| - m$ and $m \in \mathbb{N}$ is maximum such that $\sum_{p \in P_{x,t}} 2^{-|p|} \leq 2^{-m}$.*

Proof. We define N. On an input y, it simulates computations of V on all prefixes of y using dovetailing, until $V(y_0) \downarrow$ for some prefix y_0 of y. This is done again for the rest of the string y without the prefix y_0, such that eventually three sub-strings y_0, y_1 and y_2 are found, with $y = y_0 y_1 y_2$ and $V(y_0) \downarrow$ and $V(y_1) \downarrow$. When y is not of this

form, N keeps computing forever. Let $x_0 = V(y_0)$ and $x_1 = V(y_1)$. Let $r \in \{0,1\}^*$ be the program from Corollary 6.2.8 and let Q be the machine from Lemma 5.2.8. Then N simulates Q on input $\langle\langle r, y_0 y_1\rangle, y_2\rangle$ and when it halts, outputs the same.

We claim that this N is sufficient. Let x, t, p_x and p_t be as in the lemma. Let $p \in P_{x,t}$. Let m be maximum such that $\sum_{p \in P_{x,t}} 2^{-|p|} \leq 2^{-m}$, and thus also $\sum_{p \in P_{x,t}} 2^{-|p|-m} \leq 1$. Then x_p is the string that satisfies $|x_p| = |p| - m$ and $Q(\langle\langle r, p_x p_t\rangle, x_p\rangle) = p$. The existence is guaranteed by Corollary 6.2.8 and Lemma 5.2.8.

In order to see that this suffices, consider the computation of N on input $p_x p_t x_p$. Then, y_0 and y_1 will get the values p_x and p_t and y_2 will get the value x_p. N will thus output $Q(\langle\langle r, p_x p_t\rangle, x_p\rangle) = p$. Moreover, N is a prefix-machine: it will only halt on strings which are the concatenation of three other strings, all coming from specific prefix-free sets. □

How exactly the programs can be compressed is formulated in the following corollary.

Corollary 6.2.10. *There exists a $c \in \mathbb{N}$ such that for all $x \in \{0,1\}^*$ and $t \in \mathbb{N}$, for any $m \in \mathbb{N}$ such that $\sum_{p \in P_{x,t}} 2^{-|p|} \leq 2^{-m}$, for all $p \in P_{x,t}$*

$$K(p) \leq K(x) + K(bin(t)) + |p| - m + c.$$

Proof. Let N be the prefix machine from Lemma 6.2.9, and α_N as in Definition 5.1.7.1. Let M be the self-delimiting machine from Theorem 5.1.8 and α_M as in Definition 5.1.4.2. Then $c = 2|\alpha_M| + 2|\alpha_N| + 4$.

Let x, t and m as in the statement, let p_x be a witness of $K(x)$ (that is, $V(p)\downarrow$, $V(p) = x$ and $|p_x| = K(x)$) and let p_t be a witness of $bin(t)$. Let $p \in P_{x,t}$. By Lemma 6.2.9 and Theorem 5.1.8, there exists an $x_p \in \{0,1\}^*$ with $|x_p| = |p| - m$ such that $V(\langle\alpha_M, \langle\alpha_N, p_x p_t x_p\rangle\rangle)\downarrow$ and $V(\langle\alpha_M, \langle\alpha_N, p_x p_t x_p\rangle\rangle) = p$. Thus, $K(p) \leq |\langle\alpha_M, \langle\alpha_N, p_x p_t x_p\rangle\rangle| \leq K(x) + K(bin(t)) + |p| - m + c$. □

We state the relation between ldepth and depth, which comes from Lemma 3 in [9].

Theorem 6.2.11. *We can make two comparisons.*

1. *There is a $c_1 \in \mathbb{N}$ such that for all $x \in \{0,1\}^*$, $s \in \mathbb{N}$ and $t \in \mathbb{N}$,*

$$\mathrm{ldepth}_{s+K(bin(t))+c_1}(x) > t \quad \Longrightarrow \quad \mathrm{depth}_s(x) > t. \tag{6.5}$$

2. *There is a $c_2 \in \mathbb{N}$ such that for all $x \in \{0,1\}^*$ and $s \in \mathbb{N}_{\geq 1}$,*

$$\mathrm{depth}_{s+c_2}(x) \leq \mathrm{ldepth}_s(x). \tag{6.6}$$

Thus, the two versions of depth are similar in the sense that they can be bounded by each other subject to a small addition in the significance level. Note that, to make the two parts of the lemma look alike, we could rewrite (6.6) as

$$\mathrm{depth}_{s+c_2}(x) > t \quad \Longrightarrow \quad \mathrm{ldepth}_s(x) > t. \tag{6.7}$$

Proof. 1. Let d_1 be the constant from Corollary 6.2.10. Let d_2 be the constant from Theorem 5.2.2, that is,

$$Q_V(y) < 2^{-(K(y)-d_2)} \tag{6.8}$$

for all $y \in \{0,1\}^*$. We let $c_1 = d_1 + d_2$.

Now let $x \in \{0,1\}^*$ and $s,t \in \mathbb{N}$ be such that $\mathrm{ldepth}_{s+K(bin(t))+c_1}(x) > t$. By definition, $Q_V^t(x)/Q_V(x) < 2^{-(s+K(bin(t))+c_1)}$. By equation (6.8),

$$\sum_{p \in P_{x,t}} 2^{-|p|} = Q_V^t(x) < 2^{-(s+K(x)+K(bin(t))+c_1-d_2)}. \tag{6.9}$$

Now, by Corollary 6.2.10 for $m = s + K(x) + K(bin(t)) + c_1 - d_2$, we have

$$\begin{aligned} K(p) &\leq K(x) + K(bin(t)) + |p| - m + d_1 \\ &= |p| - s \end{aligned}$$

for all $p \in P$, which implies $\mathrm{depth}_s(x) > t$.

2. We let $c_2 = 2 \cdot |\alpha_R| + 3$, where R is the machine from Lemma 6.2.5 and α_R is as in property 2 of Definition 5.1.4. We prove (6.7).

The assumption $\mathrm{depth}_{s+c_2}(x) > t$ in (6.7) means that $|p| \geq K(p) + s + c_2$ for all $p \in P_{x,t}$, and thus

$$Q_V^t(x) = \sum_{p \in P_{x,t}} 2^{-|p|} \leq \sum_{p \in P_{x,t}} 2^{-(K(p)+s+c_2)}. \tag{6.10}$$

For each $p \in P_{x,t}$, let $q_p \in \{0,1\}^*$ be a shortest program for p (i.e., $V(q_p) = p$ and $|q_p| = K(p)$). Since $\langle \alpha_R, q_p \rangle$ is another program for x which, by (5.1), has length $|\langle \alpha_R, q_p \rangle| = |q_p| + c_2 - 1 = K(p) + c_2 - 1$, we have

$$Q_V(x) \geq \sum_{p \in P} 2^{-|\langle \alpha_R, q_p \rangle|} = \sum_{p \in P} 2^{-(K(p)+c_2-1)}. \tag{6.11}$$

Comparing (6.10) and (6.11), we have $Q_V^t(s)/Q_V(s) \leq 2^{-s-1} < 2^{-s}$, and thus $\mathrm{ldepth}_s(x) > t$, as was claimed in (6.7). □

6.3 Problems with the significance level

6.3.1 Instability

In this section we show that logical depth is *unstable* with respect to the significance level: the logical depth $\mathrm{ldepth}_s(x)$ of a string x can change a lot under only small changes of s. In what follows, we assume that for all $n \in \mathbb{N}$, we have that $\infty > n$. When some expression $f(x)$ can be equal to ∞ for some x, then $f(x) < n$ for some $n \in \mathbb{N}$ implies $f(x) \neq \infty$

In what follows, we will use c to denote the constant c from Proposition 6.2.6, such that $\mathrm{depth}_{s+c} \leq \mathrm{d}_s(x)$ for all $x \in \{0,1\}^*$ and $s \in \mathbb{N}$. Also recall the constants a, b and $c' \in \mathbb{N}$ from Remark 6.1.4, such that $\mathrm{d}_s(x) \leq a|x| + b$ for all $x \in \{0,1\}^*$ and $s \in \mathbb{N}$ for which $s \geq |x| + 2\log|x| + c - K(x)$. We define $l(n) = n + 2\log n + c'$,

which will also be used later. Note that $\text{depth}_{l(n)+c} \leq \text{d}_{s(n)}(x) \leq an + b$ for any $x \in \{0,1\}^n$.

In Theorem 2 of [6], the instability of logical depth with respect to the significance level was first proven for $\text{d}_s(x)$.

Theorem 6.3.1. *The function*

$$f(n) = \max_{|x|=n, 1 \leq s \leq l(n)} \big(\text{d}_s(x) - \text{d}_{s+1}(x)\big)$$

grows faster than any computable function (i.e., for any computable $h\colon \mathbb{N} \to \mathbb{N}$, there exists $n_0 \in \mathbb{N}$ such that $f(n) > h(n)$ for all $n \geq n_0$).

For $\text{depth}_s(x)$, some difficulties arise because it can be infinite for some s. Using similar techniques we can show the following.

Theorem 6.3.2. *For any $s \in \mathbb{N}$ and $x \in \{0,1\}^*$, let $f'(s,x) = \text{depth}_s(x) - \text{depth}_{s+1}(x)$ if $\text{depth}_{s+1}(x) \leq \text{depth}_s(x) < \infty$, and $f'(s,x) = 0$ otherwise. Then the function*

$$f(n) = \max_{|x|=n, 0 \leq s \leq l(n)+c} f'(s,x)$$

grows faster than any computable function.

The proof is given later. The function $f(n)$ gives the maximum value by which the logical depth can change for strings of the same length, when only adding 1 to the significance level. We will show that this function is incomputable by showing that it is related to another function: the maximum computation time for programs that compute strings of a given length. Since $\text{depth}_{c+1}(x) < \infty$ for all x, we restrict ourselves to programs p for which $|p| - K(p) < c+1$. We thus first show the following lemma.

Lemma 6.3.3. *Define $g\colon \mathbb{N} \to \mathbb{N}$ by*

$$g(n) = \max_{|x|=n} \min_{p} \{T_V(p) : x, p \in \{0,1\}^* \wedge V(p) = x \wedge |p| - K(p) < c+1\}.$$

Then g grows faster than any computable function.

Proof. Assume, towards a contradiction, that g is computable. Then there exists a $p \in \{0,1\}^*$ such that $V(\langle p, bin(n)\rangle) = g(n)$ for all $n \in \mathbb{N}$. We will show that this implies that there exists a $c'' \in \mathbb{N}$, such that for any $n \in \mathbb{N}$, there exists a $q_n \in \{0,1\}^*$ of length $K(n) + K(p) + c''$, such that $V(q_n) = x$, where $x \in \{0,1\}^*$ is the smallest string of length n for which $K(x) \geq l(n)$. That is, x is *random*. The existence of such x for which $K(x) \geq l(|x|)$ is a known fact. Then $K(x) \leq |q_n| = K(n) + K(p) + c'' < l(n)$ for big enough n, which is a contradiction. We conclude that g cannot be computable.

We first define a prefix machine N that does the following, similarly to how the machine from Lemma 6.2.9 works. On an input $y \in \{0,1\}^*$, the machine N simulates computations of V on all prefixes of y using dovetailing, until $V(y_0)$ halts for some prefix y_0 of y. Let $V(y_0) = x_0$. The machine defines $y_1 \in \{0,1\}^*$ such that $y = y_0 y_1$ and computes $V(y_1)$. When V does not halt on input y_1 or when $V(y_1) \downarrow$

does not hold, then N continues to compute forever. Otherwise, let $x_1 = V(y_1)$. The machine N then computes $V(\langle x_0, x_1\rangle)$. If this computation does not halt, then N continues to compute forever. If $V(\langle x_0, x_1\rangle) = bin(n)$ for some $n \in \mathbb{N}$, then N goes over all strings of length n in lexicographical order. N outputs the first $x \in \{0,1\}^*$ for which the shortest program $p \in \{0,1\}^*$ that outputs x within $g(n)$ steps (that is, $V(p) = x$ and $T_V(p) \leq g(n)$) is of length at least $l(n)$.

Note that the machine N only halts on strings xy for which $V(x)\downarrow$ and $V(y)\downarrow$. Then N has to be a prefix-machine, since V is a self-delimiting machine.

Let $p \in \{0,1\}^*$ be such that $V(\langle p, bin(n)\rangle) = g(n)$ for all $n \in \mathbb{N}$. Fix an $n \in \mathbb{N}$. Let $y_n \in \{0,1\}^*$ be the witness of $K(n)$, that is, $V(y_n) = bin(n)$ and $|y_n| = K(n)$. Let y_p similarly be the witness of $K(p)$. Then the string q_n is defined as $\langle \alpha_M, \langle \alpha_N, y_p y_n\rangle\rangle$, where α_M is the binary representation of the machine M from Theorem 5.1.8 as in property (2) from Definition 5.1.4, and α_N is the binary representation of N as in Theorem 5.1.8. By equation (5.1), $|q_n| = 2|\alpha_M| + 2|\alpha_N| + K(p) + K(n) + 4$. The constant c'' that was mentioned at the beginning is thus equal to $2|\alpha_M| + 2|\alpha_N| + 4$. We can see that for each $n \in \mathbb{N}$ we have $V(q_n) = x$ for an $x \in \{0,1\}^*$ such that $K(x) \geq l(n)$.

By the definition of g, and since depth_s is non-increasing in s, we know that there cannot be a shorter program for x that computes it within any amount of time.

For any computable function h such that $h(n) \geq g(n)$ for all $n \in \mathbb{N}$, the above program still works. So, arguing by contradiction, we can conclude that for any computable h, there exists an n_0 such that $g(n) > h(n)$ for all $n \geq n_0$. □

The proof of Theorem 6.3.2 can now be given similarly as in [6], but instead using Lemma 6.3.3.

Proof of Theorem 6.3.2. If $|x| = n$, then $\text{depth}_{l(n)+c}(x) \leq an + b$ for any $x \in \{0,1\}^*$. There also exists an $x_n \in \{0,1\}^*$ of length n, and a minimal $s' \in \{0, \ldots, c+1\}$, such that $\text{depth}_{s'}(x_n) = g(n)$. Consider the average of $f'(s, x_n)$ over all s such that $s' \leq s \leq l(n) + c$. This average is bigger than or equal to $(g(n) - an + b)/(l(n) + c)$, and thus $f(n) \geq (g(n) - an + b)/(l(n) + c)$. From Lemma 6.3.3 we obtain that f grows faster than any computable function. □

6.3.2 Comparing depth

It has not been made precise which significance level to choose when considering the logical depth of a string. Knowing that depth is unstable with respect to the significance level, as shown in Theorem 6.3.2, makes it difficult to consider the complexity value for just a single significance level.

Another definition for the physical complexity of a string is *sophistication* [25, 24]. Similarly as for logical depth, sophistication also uses a significance level in the definition. Moreover, similarly as for logical depth, it was shown that sophistication is unstable with respect to this significance level. See Theorem 5 in [2]. The authors said that the instability poses no problem, and that it shows that sophistication should be considered as a *function* in the significance level instead.

In this case, if we want to compare the complexity of different strings, we need to outline how to do this for functions in the significance level. So far, this has not

been done. In this section, we show how for logical depth we can find two strings and significance levels, such that which of the strings has the highest complexity value differs for both significance levels. This shows that how to compare such functions is not straightforward.

We first state the theorem itself. However, in order to prove this, we need some intermediate results. These will be stated afterwards, with their proofs. At the end of this section we provide the proof of the theorem.

Theorem 6.3.4. *There exist infinitely many $n \in \mathbb{N}$, for which there are $x, y \in \{0,1\}^*$ with $|x| = |y| = n$, and s_1, $s_2 \in \mathbb{N}$, such that $s_1 \leq s_2$ and $\infty > \text{depth}_{s_1}(x) > \text{depth}_{s_1}(y)$ and $\text{depth}_{s_2}(x) < \text{depth}_{s_2}(y)$.*

We will need a result that gives us the existence of deep strings. This result was first discussed in Bennett's paper [9], under the heading "Examples of very deep objects" on page 244.

Theorem 6.3.5. *There exists $c \in \mathbb{N}$ such that for any $n \in \mathbb{N}$ and $t \in \mathbb{N}$, there exists an $x \in \{0,1\}^*$ of length n such that*

$$\text{depth}_{n-K(n)-2K(t)-c}(x) > t.$$

The proof is based on [9] as well.

Proof. Before we can give c, we have to define a Turing machine N. On an input y, the machine M simulates computations of V on all prefixes of y with dovetailing, until $V(y_0) \downarrow$ for some prefix y_0 of y. Let $y = y_0 y_1$ Then it simulates V on input y_1. If not $V(y_1) \downarrow$, then it keeps computing forever. Let $V(y_0) = bin(z_0)$ and let $V(y_1) = bin(z_1)$ for some z_0, $z_1 \in \mathbb{N}$. Let $x_0^{z_0}, x_1^{z_0}, \ldots$ be the lexicographical ordering of strings of length z_0. For $i = 0, 1, \ldots$, M computes $Q_V^{z_1}(x_i^{z_0})$. If $Q_V^{z_1}(x_i^{z_0}) \leq 2^{-z_0}$ for some i, then N outputs $x_i^{z_0}$ and the computation halts. Otherwise N keeps computing forever.

We continue to define c. First, let α_N be the binary representation of N, as in Definition 5.1.7.1. Let M be the machine from Theorem 5.1.8, and let α_M be the binary representation of M as in Definition 5.1.4.2. Let $d \in \mathbb{N}$ be as in the Coding Theorem 5.2.2, such that

$$Q_V(x) < 2^{-(K(x)-d)} \tag{6.12}$$

for all $x \in \{0,1\}^*$. Let $c_1 \in \mathbb{N}$ be as in Theorem 6.2.11, such that for all $x \in \{0,1\}^*$, $s \in \mathbb{N}$ and $t \in \mathbb{N}$

$$\text{ldepth}_{s+K(t)+c_1}(x) > t \Rightarrow \text{depth}_s(x) > t. \tag{6.13}$$

Then $c = 4|\alpha_M| + 2|\alpha_N| + 4 + c_1 - d$.

Let n, $t \in \mathbb{N}$. Let p_n, $p_t \in \{0,1\}^*$ be the witnesses for $K(n)$ and $K(t)$ respectively. That is, $K(n) = |p_n|$, $V(p_n) \downarrow$ and $V(p_n) = bin(n)$, and similarly for t.

We first argue that the computation of N on input $p_n p_t$ halts and outputs an $x \in \{0,1\}^n$. Namely, on input $p_n p_t$, the machine N will find that $z_0 = n$ and $z_1 = t$. It will then compute $Q_V^t(x)$ for strings x of length n. If $Q_V^t(x) > 2^{-n}$ for all $x \in \{0,1\}^n$, then N will not halt. If $Q_V^t(x) \leq 2^{-n}$ for some x, then N halts and outputs that x. We therefore have to argue that $Q_V^t(x) \leq 2^{-n}$ for some

x. Assume, towards a contradiction, that $Q_V^t(x) > 2^{-n}$ for all $x \in \{0,1\}^n$. Then $\sum_{x\in\{0,1\}^n} Q_V^t(x) > 2^n \cdot 2^{-n} = 1$. This is a contradiction, since V is a self-delimiting machine and in particular prefix-free, therefore all programs on which it halt satisfy the Kraft inequality.

For any n, $t \in \mathbb{N}$, there thus exists an x of length n, such that $N(p_n p_t)$ halts and $N(p_n p_t) = x$, where p_n, $p_t \in \{0,1\}^*$ are the witnesses for $K(n)$ and $K(t)$ respectively. Also note that N only halts on $x \in \{0,1\}^*$, for which there exist x_0, $x_1 \in \{0,1\}^*$, for which $x = x_0 x_1$ and $V(x_0)\downarrow$ and $V(x_1)\downarrow$. And thus N is a prefix machine.

Thus, $V(\langle \alpha_M, \langle \alpha_N, p_n p_t\rangle\rangle)\downarrow$ and $V(\langle \alpha_M, \langle \alpha_N, p_n p_t\rangle\rangle) = x$. Equation 5.1 then implies that

$$K(x) \leq K(n) + K(t) + 4|\alpha_M| + 2|\alpha_N| + 4. \tag{6.14}$$

Moreover, by the computation of M, we have that $Q_V^t(x) \leq 2^{-n}$. By equation 6.12, we can then deduce that

$$\frac{Q_V^t(x)}{Q_V(x)} < 2^{-(n-K(x)+d)}.$$

From Definition 6.1.5, we then obtain that

$$\text{ldepth}_{n-K(x)+d}(x) > t.$$

And from equation 6.13, that

$$\text{depth}_{n-K(x)+d-K(t)-c_1}(x) > t.$$

By equation 6.14,

$$\begin{aligned} n - K(x) + d - K(t) - c_1 &\geq n - K(n) - 2K(t) - 4|\alpha_M| - 2|\alpha_N| - 4 + d - c_1 \\ &= n - K(n) - 2K(t) - c. \end{aligned}$$

And thus $\text{depth}_{n-K(n)-2K(t)-c}(x) > t$. □

Now that we have this result, we will step by step find the specific x, $y \in \{0,1\}^*$ as from Theorem 6.3.4. We start with the requirements for finding the right x.

Specifying x

The idea is as follows. We want a string x, which has high depth at low significance levels, but guaranteed a low depth at higher significance levels. We do this by letting x be the same as another string z, but with every bit doubled. That is $x = db(z)$, where db is the function that doubles every bit (e.g., $db(101) = 110011$). We let z have a high depth at low significance levels. We then show that therefore x also has a high depth at high significance levels. But, if we let the significance level of the depth for x be high enough, then for the witness of the depth of x we can even use programs that include the description of z. Given z, we can also quickly find x, as we will show. Therefore the depth x at such significance levels will be low.

We will first show an upper bound for the logical depth of the string $db(z)$ for some $z \in \{0,1\}^*$. The observation here is that when we have the string z, we can also obtain $db(z)$ from this.

Lemma 6.3.6. *There exist c_1, $c_2 \in \mathbb{N}$ such that for any $z \in \{0,1\}^*\backslash\epsilon$ of length n,*

$$\text{depth}_{n+2\log n+c_1}(db(z)) \leq c_2 n^2.$$

Proof. Before we can give c_1 and c_2, we need to consider the following. Let $\alpha \in \{0,1\}^*$ be as in Definition 5.1.4.4, such that for any $x \in \{0,1\}^*\backslash\epsilon$, $V(\langle\alpha, \overline{x}\rangle) \downarrow$ and $V(\langle\alpha, \overline{x}\rangle) = x$.

We define a self-delimiting Turing machine M. On an input $y \in \{0,1\}^*$, the machine M executes the computation of V on input $\langle\alpha, y\rangle$ (while thus only moving the head on the input-tape from left to right). Every time when during that computation a bit is to be written on the output-tape, M prints the bit twice. When the computation of V on input $\langle\alpha, y\rangle$ halts then so does M.

From Definition 5.1.4.4, we have that for any $x \in \{0,1\}^*\backslash\epsilon$, $T_V(\langle\alpha, \overline{x}\rangle) \leq a|x| + b \leq (a+b)|x|$ for some a, $b \in \mathbb{N}$. This, together with the above description of M, tells us that there exists a $c \in \mathbb{N}$ such that for any $x \in \{0,1\}^*\backslash\epsilon$, $T_V(\langle\alpha_M, \overline{x}\rangle) \leq c|x|^2$.

Now we can define $c_1 = 2|\alpha_M| + 5$, and $c_2 = c$. Let $z \in \{0,1\}^*\backslash\varepsilon$, and let $n = |z|$. Then $V(\langle\alpha_M, \overline{z}\rangle) \downarrow$ and $V(\langle\alpha_M, \overline{z}\rangle) = db(z)$. Moreover,

$$\begin{aligned}
|\langle\alpha_M, \overline{z}\rangle| - K(\langle\alpha_M, \overline{z}\rangle) &\leq |\langle\alpha_M, \overline{z}\rangle| \\
&= 2|\alpha_M| + 2 + |\overline{z}| && \text{by eq. 5.1} \\
&\leq n + 2\log n + 2|\alpha_M| + 5 && \text{by eq. 5.2} \\
&= n + 2\log n + c_1
\end{aligned}$$

We have thus found the program $\langle\alpha_M, \overline{z}\rangle$ for $db(z)$, for which $T_V(\langle\alpha_M, \overline{z}\rangle) \leq c_2 n^2$ and $|\langle\alpha_M, \overline{z}\rangle| - K(\langle\alpha_M, \overline{z}\rangle) \leq n + 2\log n + c_1$. We conclude that $\text{depth}_{n+2\log n+c_1}(db(z)) \leq c_2 n^2$. □

Now that we have the upper bound of our $db(z)$ for a higher significance level, we will give the lower bound of $db(z)$ at lower significance levels. But for this we need an intermediate result. We will show that the depth of a string z is not too different from the depth of the string $db(z)$.

Lemma 6.3.7. *There exist c_1, $c_2 \in \mathbb{N}$, such that for all $z \in \{0,1\}^*\backslash\epsilon$ and all t, $s \in \mathbb{N}$*

$$\text{depth}_s(z) > t \Rightarrow \text{depth}_{s-c_1}(db(z)) > t - c_2|z|.$$

Proof. Before we can give c_1 and c_2, we first define a self-delimiting Turing machine M.

On an input $p \in \{0,1\}^*$, the machine does the same computation V on input p. If $V(p) \uparrow$ or V keeps computing forever on input p, then so does M.

If $V(p) \downarrow$, the machine defines a bit $b = 0$, and executes the computation of V on input $(\langle\alpha, \overline{V(p)}\rangle)$. Every time when during that computation a bit would be printed on the output tape, M does the following. If $b = 0$, then M also writes the bit on the output tape, and then sets $b = 1$. If $b = 1$, then M does not write the bit on the output tape, but sets $b = 0$ and continues the rest of the computation of V on input $(\langle\alpha, \overline{V(p)}\rangle)$. Eventually, when the computation of V on input $(\langle\alpha, \overline{V(p)}\rangle)$ halts, M also halts.

Let α_M be the binary representation of M, as in Definition 5.1.4.2. From the above description and Definition 5.1.4.4, we obtain that there exists a $d \in \mathbb{N}$, such that for all $p \in \{0,1\}^*$, if $V(p)\downarrow$, then

$$T_V(\langle \alpha_M, p\rangle) \leq T_V(p) + d|V(p)|. \tag{6.15}$$

We let $c_2 = d$. Let c be the constant from Lemma 6.2.1 for V. That is,

$$K(y) \leq K(\langle x, y\rangle) + c \tag{6.16}$$

for any x, $y \in \{0,1\}^*$. Then we let $c_1 = 2|\alpha_M| + 2 + c$.

We now continue to proving the statement of the lemma. Let $z \in \{0,1\}^*\backslash\epsilon$, let t, $s \in \mathbb{N}$ and assume that $\mathrm{depth}_s(z) > t$. Also assume, towards a contradiction, that $\mathrm{depth}_{s-c_1}(db(z)) \leq t - c_2|z|$. Let $p \in \{0,1\}^*$ be the witness of this. That is, $V(p)\downarrow$, $V(p) = db(z)$, $T_V(p) \leq t - c_2|z|$ and $|p| - K(p) < s - c_1$.

Then $V(\langle \alpha_M, p\rangle)\downarrow$ and $V(\langle \alpha_M, p\rangle) = z$. And thus,

$$\begin{aligned}
|\langle \alpha_M, p\rangle| - K(\langle \alpha_M, p\rangle) &= 2|\alpha_M| + 2 + |p| - K(\langle \alpha_M, p\rangle) && \text{by eq. 5.1}\\
&< 2|\alpha_M| + 2 + |p| - K(p) + c && \text{by eq. 6.16}\\
&= c_1 + |p| - K(p)\\
&< s && \text{since } |p| - K(p) < s - c_1.
\end{aligned}$$

Moreover, by equation 6.15, we have that $T_V(\langle \alpha_M, p\rangle) \leq T_V(p) + c_2|z|$. And thus $T_V(\langle \alpha_M, p\rangle) \leq t$. This implies that $\mathrm{depth}_s(z) \leq t$, a contradiction. □

Now we can give the lower bound for $db(z)$.

Lemma 6.3.8. *There exist c_3, c_4, such that for any n, t, $d \in \mathbb{N}_{\geq 1}$ with $K(t) \leq K(n) + d$, there exists a $z \in \{0,1\}^*$ of length n such that*

$$\mathrm{depth}_{n-3K(n)-c_3-2d}(db(z)) > t - c_4 n.$$

Proof. Let c_1, c_2 be as in Lemma 6.3.7. Let c be the constant in Theorem 6.3.5, such that for any $n \in \mathbb{N}$ and $t \in \mathbb{N}$, there exists an $x \in \{0,1\}^*$ of length n such that

$$\mathrm{depth}_{n-K(n)-2K(t)-c}(x) > t. \tag{6.17}$$

Then let $c_3 = c + c_1$ and let $c_4 = c_2$.

We continue to prove the lemma. Let n, $t \in \mathbb{N}$ be as in the statement.

From equation 6.17, we get that there exists a $z \in \{0,1\}^*$ of length n such that $\mathrm{depth}_{n-K(n)-2K(t)-c}(z) \geq t$.

Then from Lemma 6.3.7, we get that

$$\mathrm{depth}_{n-K(n)-2K(t)-c-c_1}(db(z)) \geq t - c_2 n. \tag{6.18}$$

By assumption, $K(t) \leq K(n) + d$, and thus

$$\begin{aligned}
n - K(n) - 2K(t) - c - c_1 &\geq n - 3K(n) - 2d - c - c_1\\
&= n - 3K(n) - c_3 - 2d.
\end{aligned}$$

Then $\mathrm{depth}_{n-3K(n)-c_3-2d}(db(z)) > t - c_2 n$ follows from equation 6.18. □

Specifying y

We have all the necessary intermediate results for one string. Now we will state some lemmas that are necessary for ensuring that the other string exists.

Lemma 6.3.9. *There exists a $d \in \mathbb{N}$ such that for all $n \in \mathbb{N}$,*

$$K(n^3) \leq K(n) + d$$

and

$$K(2n) \leq K(n) + d.$$

Proof. Straightforward. □

Lemma 6.3.10. *There exists a d_1, $d_2 \in \mathbb{N}$ such that for all $n \in \mathbb{N}$, there exists a string y of length $2n$, such that*

$$\mathrm{depth}_{2n-3K(n)-d_1}(y) > n^3$$

and

$$K(y) \leq K(n) + d_2.$$

The proof is similar to the proof of Theorem 6.3.5.

Proof. Before we can give d_1, d_2, we need to define a self-delimiting Turing machine M.

On an input $p \in \{0,1\}^*$, the machine M computes V on input p. If $V(p) \uparrow$ or V keeps computing forever on input p, then so does M. If $V(p) \downarrow$, then let $bin(n) = V(p)$. Let x_0^{2n}, $x_1^{2n}, \ldots$ be the lexicographical ordering of strings of length $2n$. For $i = 0, 1, \ldots$, M computes $Q_V^{n^3}(x_i^{2n})$. If $Q_V^{n^3}(x_i^{2n}) \leq 2^{-2n}$ for some i, then M outputs x_i^{2n} and the computation halts. Otherwise M keeps computing forever.

Let d be as in Lemma 6.3.9 such that

$$K(n^3) \leq K(n) + d \tag{6.19}$$

and

$$K(2n) \leq K(n) + d \tag{6.20}$$

for all $n \in \mathbb{N}$.

Let $e \in \mathbb{N}$ be as in the Coding Theorem 5.2.2, such that

$$Q_V(x) < 2^{-(K(x)-e)} \tag{6.21}$$

for all $x \in \{0,1\}^*$. Let $c_1 \in \mathbb{N}$ be as in Theorem 6.2.11, such that for all $x \in \{0,1\}^*$, $s \in \mathbb{N}$ and $t \in \mathbb{N}$

$$\mathrm{ldepth}_{s+K(t)+c_1}(x) > t \Rightarrow \mathrm{depth}_s(x) > t. \tag{6.22}$$

Let α_M be the binary representation of M as in Definition 5.1.4.2. Then $d_2 = 2|\alpha_M| + 2$ and $d_1 = d + d_2 + c_1 - e$.

Let n, $c \in \mathbb{N}$. Let $p_n \in \{0,1\}^*$ be the witness of $K(n)$. That is, $K(n) = |p_n|$, $V(p_n) \downarrow$ and $V(p_n) = bin(n)$. Similarly to how it is shown in the proof of Theorem 6.3.5, we have that there exists an $y \in \{0,1\}^*$ of length $2n$ with $V(\langle \alpha_M, p_n \rangle) \downarrow$

and $V(\langle \alpha_M, p_n \rangle) = y$. By equation 5.1, we have that $K(y) \leq K(n) + 2|\alpha_M| + 2 = K(n) + d_2$.

In order to prove the lemma, we now need to show that $\text{depth}_{2n-3K(n)-d_1}(y) > n^3$. This also follows similarly to the proof of Theorem 6.3.5. $Q_V^{n^3}(y) \leq 2^{-2n}$ by definition of M. By equation 6.21,

$$\frac{Q_V^{n^3}(y)}{Q_V(y)} < 2^{-(2n-K(y)+e)}.$$

Thus by Definition 6.1.5,

$$\text{ldepth}_{2n-K(y)+e}(y) > n^3.$$

And from equation 6.22,

$$\text{depth}_{2n-K(y)+e-K(n^3)-c_1}(y) > n^3.$$

Since $K(y) \leq K(n) + d_2$,

$$\begin{aligned} 2n - K(y) + e - K(n^3) - c_1 &\geq 2n - K(n) - d_2 + e - K(n^3) - c_1 \\ &\geq 2n - 2K(n) + e - d - d_2 - c_1 \qquad \text{by equation 6.19} \\ &= 2n - 2K(n) - d_1. \end{aligned}$$

□

For the following lemma, recall Definition 6.1.1.

Lemma 6.3.11. *For all $c \in \mathbb{N}$ there exists a $d_3 \in \mathbb{N}$, such that for all $y \in \{0,1\}^* \backslash \epsilon$ of length $2n$,*

$$K(\mathrm{d}_1(y) + cn) \leq K(y) + d_3.$$

Proof. Let $c \in \mathbb{N}$. Before we can define d_3, we define a self-delimiting Turing machine M.

On an input $p \in \{0,1\}^*$, the machine M computes V on input p, and it counts the steps for this computation. If $V(p)\downarrow$, let $m \in \mathbb{N}$ be the number of steps counted. Then M outputs $m + c|V(p)|$. If $V(p)\uparrow$ or keeps computing forever, then so does M.

We let $d_3 = 2|\alpha_M| + 2$.

We continue to prove the lemma. Let $y \in \{0,1\}^* \backslash \epsilon$. Let $T = \mathrm{d}_1(y)$ and let $p_y \in \{0,1\}^*$ be the witness of $\mathrm{d}_1(y)$. That is, $K(y) = |p_y|$, $V(p_y)\downarrow$, $V(p_y) = y$ and $T_V(p_y) = \mathrm{d}_1(y) = T$.

Let α_M be the binary representation of M as in Definition 5.1.4.2. By definition of M, we have $V(\alpha_M, p_y)\downarrow$ and $V(\alpha_M, p_y) = T + c|y|$. Thus by equation 5.1, $K(T + c|y|) \leq 2|\alpha_M| + 2 + K(y) = K(y) + d_3$. □

The final theorem

We continue to the proof of Theorem 6.3.4.

Theorem 6.3.4. *There exist infinitely many $n \in \mathbb{N}$, for which there are $x, y \in \{0,1\}^*$ with $|x| = |y| = n$, and $s_1, s_2 \in \mathbb{N}$, such that $s_1 \leq s_2$ and $\infty > \text{depth}_{s_1}(x) > \text{depth}_{s_1}(y)$ and $\text{depth}_{s_2}(x) < \text{depth}_{s_2}(y)$.*

Proof of Theorem 6.3.4. We first lay out the necessary constants from the previous lemmas. Let c_1, $c_2 \in \mathbb{N}$ be the constants from Lemma 6.3.6, such that for any $z \in \{0,1\}^* \backslash \epsilon$ of length n,

$$\text{depth}_{n+2\log n+c_1}(db(z)) \leq c_2 n^2. \tag{6.23}$$

Let c_3, $c_4 \in \mathbb{N}$ be the constants from Lemma 6.3.8, such that for any n, t, $d \in \mathbb{N}_{\geq 1}$ with $K(t) \leq K(n) + d$, there exists a $z \in \{0,1\}^*$ of length n such that

$$\text{depth}_{n-3K(n)-c_3-2d}(db(z)) > t - c_4 n. \tag{6.24}$$

Let d_1, $d_2 \in \mathbb{N}$ be the constants from Lemma 6.3.10 such that for all $n, \in \mathbb{N}$, there exists a string y of length $2n$, such that

$$\text{depth}_{2n-3K(n)-d_1}(y) > n^3 \tag{6.25}$$

and

$$K(y) \leq K(n) + d_2. \tag{6.26}$$

Let $d_3 \in \mathbb{N}$ be the constant from Lemma 6.3.11 for c_4, such that for all $y \in \{0,1\}^* \backslash \epsilon$ of length $2n$,

$$K(\mathrm{d}_1(y) + c_4 n) \leq K(y) + d_3. \tag{6.27}$$

Let $e \in \mathbb{N}$ be the constant from Proposition 6.2.6, such that

$$depth_{s+e} \leq d_s(x) < \infty \tag{6.28}$$

for all $s \in \mathbb{N}_{\geq 1}$ and $x \in \{0,1\}^*$.

We now continue to prove the theorem. Let $n \in \mathbb{N}_{\geq 1}$ be such that the following three equations hold:

$$n - 3K(n) - c_3 - 2(d_3 + d_2) \geq e + 1. \tag{6.29}$$

$$2n - 3K(n) - d_1 \geq n + 2\log n + c_1. \tag{6.30}$$

$$n \geq c_2. \tag{6.31}$$

Note that there are infinitely many n for which this holds.

Let $y \in \{0,1\}^*$ be the string of length $2n$ such that

$$\text{depth}_{2n-3K(n)-d_1}(y) > n^3. \tag{6.32}$$

From equation 6.26 and 6.27, we obtain that

$$K(d_1(y) + c_4 n) \leq K(n) + d_2 + d_3. \tag{6.33}$$

From equation 6.24 we then obtain that there exists a $z \in \{0,1\}^*$ of length n such that

$$\text{depth}_{n-3K(n)-c_3-2(d_2+d_3)}(db(z)) > d_1(y). \tag{6.34}$$

We let $x = db(z)$. Let $s_1 = n - 3K(n) - c_3 - 2(d_2 + d_3)$ and $s_2 = n + 2\log n + c_1$.

We now have the following.

$$
\begin{aligned}
\mathrm{depth}_{s_1}(y) &= \mathrm{depth}_{n-3K(n)-c_3-2(d_2+d_3)}(y) \\
&\leq \mathrm{depth}_{e+1}(y) && \text{by eq. 6.29} \\
&\leq \mathrm{d}_1(y) && \text{by eq. 6.28} \\
&< \mathrm{depth}_{n-3K(n)-c_3-2(d_2+d_3)}(x) && \text{by eq. 6.34} \\
&= \mathrm{depth}_{s_1}(x) \\
&< \infty && \text{by eq. 6.29 and 6.28}
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{depth}_{s_2}(x) &= \mathrm{depth}_{n+2\log n+c_1}(x) \\
&\leq c_2 n^2 && \text{by eq. 6.23} \\
&\leq n^3 && \text{by eq. 6.31} \\
&< \mathrm{depth}_{2n-3K(n)-d_1}(y) && \text{by eq. 6.32} \\
&\leq \mathrm{depth}_{n+2\log n+c_1}(y) && \text{by eq. 6.32} \\
&= \mathrm{depth}_{s_2}(y).
\end{aligned}
$$

□

Chapter 7

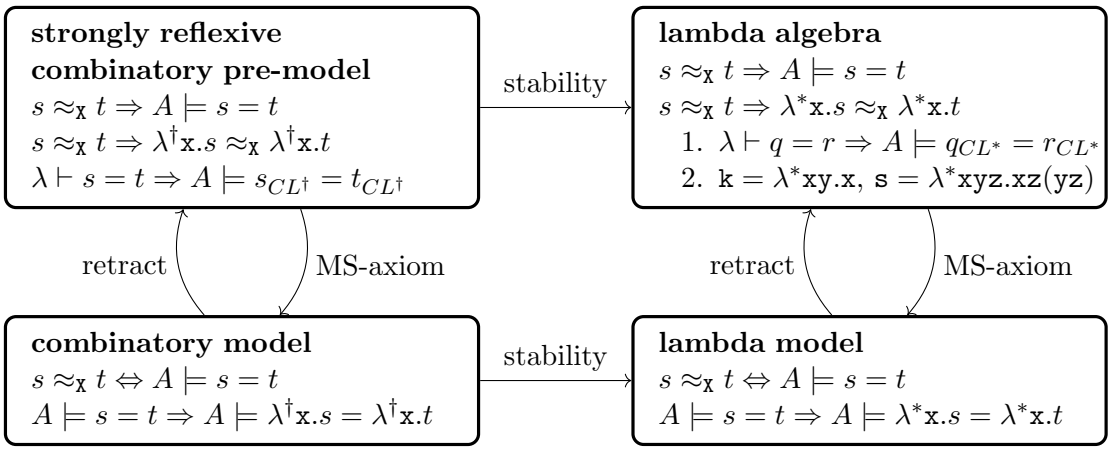
Concluding remarks

7.1 Computability

In Chapter 4 we reconsidered different structures for the lambda calculus. Starting from a combinatory algebra, we first introduced a combinatory pre-model by adding two combinators. We then defined the notion of reflexivity (an algebraic analogue of the Meyer-Scott axiom). Reflexivity was shown to be important for interpreting the lambda calculus. However, it is not sufficient. Strong reflexivity, which corresponds to the polynomial algebra being reflexive, is sufficient for interpreting the lambda calculus. We also compared strongly reflexive combinatory pre-models with other known structures of the lambda calculus.

With the results in this dissertation, the relations and properties of the known structures for the lambda calculus can now be displayed in a more complete manner. It is also useful to distinguish between two different lambda abstractions for combinatory pre-models. In addition to the known λ^*-abstraction, we introduced the $\lambda^\dagger$-abstraction. It is similar to the λ^*-abstraction, but makes use of the $\mathtt{e}$-combinator. This lambda abstraction is particularly useful for showing properties of structures of the lambda calculus that do not satisfy stability.

The results can be summarized in the diagram below. The statements hold for all q, $r \in \mathsf{T}_\Lambda(A)$ and for all s, $t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$, where A is a structure as denoted by the box.



For a strongly reflexive combinatory pre-model or for a lambda algebra A, we have that when s, $t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$ are equal as polynomials (equal using the absolute interpretation), then the corresponding algebra also satisfies their equality. That is, $s \approx_{\mathtt{X}} t \Rightarrow A \models s = t$. The other direction does not hold necessarily. It does, when the Meyer-Scott axiom holds, and thus it is true for combinatory models and lambda models. There, polynomials are determined by their behaviour as a function.

This can also be observed in the following way. For a strongly reflexive combinatory pre-model or for a lambda algebra A, the lambda abstractions only respect equality between terms in the absolute sense: $s \approx_{\mathtt{X}} t \Rightarrow \lambda^\dagger \mathtt{x}.s \approx_{\mathtt{X}} \lambda^\dagger \mathtt{x}.t$ or $s \approx_{\mathtt{X}} t \Rightarrow \lambda^* \mathtt{x}.s \approx_{\mathtt{X}} \lambda^* \mathtt{x}.t$ for all s, $t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$. For a combinatory model or for a lambda model A, we have that $A \models s = t \Rightarrow A \models \lambda^\dagger \mathtt{x}.s = \lambda^\dagger \mathtt{x}.t$ or $A \models s = t \Rightarrow A \models \lambda^* \mathtt{x}.s = \lambda^* \mathtt{x}.t$ for all s, $t \in \mathcal{T}(\mathtt{X} \cup \underline{A})$. This is thus also because of the Meyer-Scott axiom.

The differences between the structures on the left and on the right side are characterized by stability. With stability, we can replace the $\lambda^\dagger$-abstraction by the λ^*-abstraction.

We have only considered the β-equality of lambda calculus. In [7, Def 7.3.13] it was shown that in order for a lambda algebra to respect the η-equality, the axiom $\lambda^* \mathtt{x}.\mathtt{s}(\mathtt{kx})\mathtt{i} = \mathtt{i}$ can be introduced. For future work, it is worthwhile to consider how this axiom fits in the diagram with the four structures. More specifically, we should consider the difference between stable and non-stable structures, and see whether there is an alternative of the axiom using the $\lambda^\dagger$-abstraction.

Another possible direction of research is reflexive combinatory algebras. We have defined reflexivity and identified it as important, but not sufficient, for interpreting the lambda calculus. However, reflexive combinatory algebras possibly have other properties that still can be found.

We have also shown how to construct a cartesian closed category with a reflexive object from a strongly reflexive combinatory pre-model in Section 3.4. This was analogous to the known construction of a cartesian closed category from a lambda algebra. There is also a construction of lambda algebras from cartesian closed categories with a reflexive object. Possibly, there is a construction of a strongly reflexive combinatory pre-model from such categories as well, however, this seems unlikely. More research should be done in order to obtain a definite answer to this question.

7.2 Complexity

In Chapter 6, we gave an overview of several related definitions for the notion of logical depth (or physical complexity). We also outlined results that state the correspondence between different definitions. By doing this, we fixed mistakes that were found in the literature. Moreover, all these results are gathered in a single document for the first time.

It has not been made very clear how to compare the logical depth of two different strings. For example, it is not certain whether the definition results in a total order; whether we can compare the logical depth of any two strings. Perhaps this is not the case. The definition uses a significance level, which has as a result that it is not directly apparent which value of logical depth represents the "right" value for the

string, if there is such a thing. The other results in this dissertation indicate that it is important to first make clear what requirements there are for using logical depth, and specifically how to compare two different strings.

Namely, we showed that logical depth is unstable with respect to the significance level. This was already shown for sophistication, a similar definition. It was then concluded that sophistication should be considered as a function in the significance level. It is reasonable to then also assume this for logical depth. The problem with this, is that we do not know yet how to compare the logical depth of different strings, when we interpret logical depth as a function in the significance level. This has also not been written out for sophistication. The most straightforward way would be to check whether one string has a higher logical depth than the other string for every significance level. But this is not always possible. With Theorem 6.3.4, we showed that it is not always the case that one string has a higher logical depth than the other for every significance level.

With these results, it becomes uncertain how the definition of logical depth should be used. This illustrates that it is important to reconsider the basis for the definition, and make clear what assumptions and requirements are behind it. Some of these requirements are already clear: random strings and the most simple strings (such as a sequence of ones) should have the lowest depth. Other presumptions, such as those specifying which strings can be compared to each other and which not, are still needed.

Publication

Marlou M. Gijzen, Hajime Ishihara, Tatsuji Kawai, Reflexive combinatory algebras, Journal of Logic and Computation, 2022; https://doi.org/10.1093/logcom/exac049

Bibliography

[1] Leonard M. Adleman. *Time, space and randomness*. Tech. rep. MIT/LCS/TM-131. Massachusetts Institute of Technology, Laboratory for Computer Science, 1979.

[2] Luís Antunes, Bruno Bauwens, André Souto, and Andreia Teixeira. "Sophistication vs Logical Depth". In: *Theory of Computing Systems* 60.2 (2017), pp. 280–298.

[3] Luís Antunes and Lance Fortnow. "Sophistication revisited". In: *Theory of Computing Systems* 45.1 (2009), pp. 150–161.

[4] Luís Antunes, Lance Fortnow, Dieter van Melkebeek, and N. V. Vinodchandran. "Computational Depth: Concept and Applications". In: *Theoretical Computer Science* 354.3 (2006), pp. 391–404.

[5] Luís Antunes, Armando Matos, André Souto, and Paul. M. B. Vitányi. "Depth as randomness deficiency". In: *Theory of Computing Systems* 45.4 (2009), pp. 724–739.

[6] Luís Antunes, André Souto, and Paul. M. B. Vitányi. "On the rate of decrease in logical depth". In: *Theoretical Computer Science* 702 (2017), pp. 60–64.

[7] Hendrik Pieter Barendregt. *The Lambda Calculus: its Syntax and Semantics*. Vol. 103. Studies in Logic and the Foundations of Mathematics. Amsterdam: Elsevier, 1984.

[8] Charles H. Bennett. "Dissipation, information, computational complexity and the definition of organization". In: *Emerging syntheses in science*. 1985, pp. 215–233.

[9] Charles H. Bennett. "Logical Depth and Physical Complexity". In: *The Universal Turing Machine–a Half-Century Survey*. Ed. by R.Herken. Oxford University Press, 1988, pp. 227–257.

[10] Felice Cardone and J Roger Hindley. "History of lambda-calculus and combinatory logic". In: *Handbook of the History of Logic* 5 (2006), pp. 723–817.

[11] Gregory J. Chaitin. "A theory of program size formally identical to information theory". In: *Journal of the ACM (JACM)* 22.3 (1975), pp. 329–340.

[12] Gregory. J. Chaitin. "Toward a mathematical definition of "life"". In: *The Maximum Entropy Formalism*. Ed. by R. D. Levine and M. Tribus. MIT Press, 1979, pp. 477–498.

[13] Alonzo Church. “An unsolvable problem of elementary number theory.” In: *American journal of mathematics* 58 (1936), pp. 345–363.

[14] Alonzo Church. *The calculi of lambda-conversion*. Annals of Mathematics Studies 6. Princeton University Press, 1941.

[15] H. B. Curry. “An Analysis of Logical Substitution”. In: *American Journal of Mathematics* 51.3 (1929), pp. 363–384. ISSN: 00029327, 10806377. URL: http://www.jstor.org/stable/2370728.

[16] Haskell B Curry. “A simplification of the theory of combinators”. In: *Synthese* (1948), pp. 391–399.

[17] Peter Freyd. “Combinators”. In: *Categories in computer science and logic (Boulder, CO, 1987)*. Vol. 92. Contemp. Math. Amer. Math. Soc., Providence, RI, 1989, pp. 63–66. DOI: 10.1090/conm/092/1003195. URL: https://doi.org/10.1090/conm/092/1003195.

[18] Cédric Gaucherel. “Ecosystem complexity through the lens of logical depth: Capturing ecosystem individuality”. In: *Biological Theory* 9.4 (2014), pp. 440–451.

[19] Marlou M Gijzen, Hajime Ishihara, and Tatsuji Kawai. “Reflexive combinatory algebras”. In: *Journal of Logic and Computation* (July 2022). exac049. ISSN: 0955-792X. DOI: 10.1093/logcom/exac049. eprint: https://academic.oup.com/logcom/advance-article-pdf/doi/10.1093/logcom/exac049/45193554/exac049.pdf. URL: https://doi.org/10.1093/logcom/exac049.

[20] David Hilbert. “Über das Unendliche”. In: *Mathematische Annalen* 95 (1926). English translation in Van Heijenoort, From Frege to Gödel. A Source Book in Mathematical Logic (1967, 367–392), 161—190.

[21] David W. Juedes and Jack H. Lutz. “Modeling time-bounded prefix Kolmogorov complexity”. In: *Theory of Computing Systems* 33.2 (2000), pp. 111–123.

[22] Stephen C Kleene and J Barkley Rosser. “The inconsistency of certain formal logics”. In: *Annals of Mathematics* (1935), pp. 630–636.

[23] Andrei N. Kolmogorov. “Three approaches to the quantitative definition of information”. In: *Problems of Information Transmission* 1.1 (1065), pp. 4–11.

[24] Moshe Koppel. “Structure”. In: *The Universal Turing Machine: A Half-Century Survey*. Ed. by R. Herken. Oxford University Press, 1988, pp. 435–452.

[25] Moshe Koppel and Henri Atlan. “An almost machine-independent theory of program-length complexity, sophistication, and induction”. In: *Information Sciences* 56.1-3 (1991), pp. 23–33.

[26] C.P.J. Koymans. “Models of the lambda calculus”. In: *Information and Control* 52.3 (1982), pp. 306–332. ISSN: 0019-9958. DOI: https://doi.org/10.1016/S0019-9958(82)90796-3. URL: http://www.sciencedirect.com/science/article/pii/S0019995882907963.

[27] J.-L. Krivine. *Lambda-calculus, types and models*. Ellis Horwood Series in Computers and Their Applications. Translated from the 1990 French original by René Cori. Ellis Horwood, New York; Masson, Paris, 1993, pp. viii+180. ISBN: 0-13-062407-1.

[28] Joachim Lambek. "From λ-calculus to cartesian closed categories". In: *To HB Curry: essays on combinatory logic, lambda calculus and formalism* (1980), pp. 375–402.

[29] Joachim. Lambek and Philip J. Scott. *Introduction to higher order categorical logic*. Vol. 7. Cambridge Studies in Advanced Mathematics. Cambridge University Press, Cambridge, 1986, pp. x+293. ISBN: 0-521-24665-2.

[30] Leonid A. Levin. "Laws of Information Conservation (Nongrowth) and Aspects of the Foundation of Probability Theory". In: *Problemy Peredachi Informatsii* 10.3 (1974), pp. 30–35.

[31] Ming Li and Paul Vitányi. *An introduction to Kolmogorov complexity and its applications*. 3rd ed. Springer, 2008.

[32] Albert R. Meyer. "What Is a Model of the Lambda Calculus?" In: *Information and Control* 52.3 (1982), pp. 87–122.

[33] Li Ming and Paul M. B. Vitányi. *An introduction to Kolmogorov complexity and its applications (third edition)*. Springer, 2008.

[34] Gordon D. Plotkin. "The λ-calculus is ω-incomplete". In: *The Journal of Symbolic Logic* 39.2 (1974), pp. 313–317.

[35] Barkley Rosser. "New Sets of Postulates for Combinatory Logics". In: *The Journal of Symbolic Logic* 7.1 (1942), pp. 18–27. ISSN: 00224812.

[36] Moses Schönfinkel. "Über die Bausteine der mathematischen Logik". In: *Mathematische annalen* 92.3-4 (1924), pp. 305–316.

[37] Dana S. Scott. "Relating theories of the λ-calculus". In: *To H. B. Curry: essays on combinatory logic, lambda calculus and formalism*. Academic Press, London-New York, 1980, pp. 403–450.

[38] Peter Selinger. "The lambda calculus is algebraic". In: *Journal of Functional Programming* 12.6 (2002), pp. 549–566. DOI: 10.1017/S0956796801004294.

[39] Ray J. Solomonoff. "A formal theory of inductive inference. Part I and II". In: *Information and control* 7.1 (1964), pp. 1–22, 224–254.

[40] Alan M. Turing. "On computable numbers, with an application to the Entscheidungsproblem". In: *Proceedings of the London Mathematical Society* 42.2 (1936), pp. 230–265.

[41] Nikolay Vereshchagin and Alexander Shen. "Algorithmic statistics: forty years later". In: *Computability and Complexity*. Springer, 2017, pp. 669–737.

[42] Richard Zach. *"Hilbert's Program", The Stanford Encyclopedia of Philosophy (Fall 2019 Edition), Edward N. Zalta (ed.)* https://plato.stanford.edu/archives/fall2019/entries/hilbert-program/.