Characterising Computational Devices with Logical Systems

Richard Arthur James Whyman

Submitted in accordance with the requirements for the degree of

Doctor of Philosophy

University of Leeds

School of Mathematics

September 2018

The candidate confirms that the work submitted is his own and that appropriate credit has been given where reference has been made to the work of others.

Some of the work in Chapters 4, 5, 7, and 8 of this thesis has appeared in publication as: "Physical Computation and First-Order Logic," in the proceedings of the Machines Computations and Universality 2018 conference [75] (this won the best student paper award). Some of the work in Chapters 4, 5, 8 and 9 has appeared in publication as: "An Atemporal Model of Physical Complexity," in the proceedings of the Physics and Computation 2018 workshop [74]. I was responsible for the entirety of both of these publications.

Acknowledgements

Firstly I would like to thank my family for raising me and encouraging me to study mathematics. I would also like to thank Vladimir Kisil for supervising me. The Leeds logic group also deserves a thank you for inspiring me to pursue a more logic focused PhD.

Most of all I would like to thank all the friends I have made at Leeds and especially Sarah Sigley for their love and support throughout my PhD.

Abstract

In this thesis we shall present and develop the concept of a theory machine. Theory machines describe computation via logical systems, providing an overarching formalism for characterising computational systems such as Turing machines, type-2 machines, quantum computers, infinite time Turing machines, and various physical computation devices.

Notably we prove that the class of finite problems that are computable by a finite theory machine acting in first-order logic is equal to the class Turing machine computable problems. Whereas the class infinite problems that are computable by a finite first-order theory machine is equal to the class type-2 machine computable problems.

A key property of a theory machine computation is that it does not have to occur in a causally ordered manner. A consequence of this fact is that the class of problems that are computable by finite first-order theory machine in polynomial resources is equal to $\mathcal{NP} \cap \text{co-}\mathcal{NP}$. Since there are problems which appear to lie in $\mathcal{NP} \cap \text{co-}\mathcal{NP} \setminus \mathcal{P}$ that are efficiently solvable by a quantum computer (such as the factorisation problem), this gives weight to the argument that there is an atemporal/non-causal component to the apparent speed-up offered by quantum computers.

Contents

List of Figures

Chapter 1

Introduction

"Computation is a kind of magic, a mathematician can write a few a lines on a piece of paper and point to a patch of sky and say; *there* is a new plant." -Vladimir Kisil

In 1936 Turing published his famous paper [67] in which he defined what he called automatic machines but what we now call the Turing machine [27], which mathematically formulated the process of a person or machine algorithmically deciding a problem. However Turing was not the first to describe a mathematical formulation of computation, in the 1930's Church described his lambda calculus [23, 24], and prior to this the theory of general recursive functions had been developed [20, 27].

Notably, the class of problems that can be computed by a Turing machine is equal to the class of problems that are effectively calculable via Lambda calculus, which is in turn equal to the class of problems that are recursive [27, 67]. Therefore, despite being quite different from each other these three models of computation are in a sense equivalent to one another.

This equivalence lead to what is now known as the *Church-Turing thesis* [27, 29, 67], which is often rendered as:

"Every effectively calculable function is computable by a Turing machine."

The Church-Turing thesis seems to suggest that any computational system that we can compute with is at most as computationally powerful as a Turing machine. Indeed modern digital computers have the same computational power as Turing machines [27].

Suppose that we have arbitrary physical system $\mathcal{S}$, the Church-Turing thesis seems to imply that if we were to use $\mathcal{S}$ to carry out a computation then we should only be able to compute Turing machine computable problems with $\mathcal{S}$. However a Turing machine computation is defined is in a very particular way, whereas since $\mathcal{S}$ is arbitrary it is not immediately apparent what a computation on $\mathcal{S}$ actually *is*. It is therefore not entirely clear why such a computation should be bounded by what is Turing machine computable.

Remark 1.0.1 By physical system, we mean a system that exists or could be made to exist in the real world. Computing with a physical system should then involve being able to input data into the system (by adjusting the locations or properties of the objects within said system) so that we may reliably observe an output from the system that provides a solution to some problem of ours. Examples of physical computation systems include a table of ball bearings and grooves [13], a screen diffracting a ray of light [17], a slide rule [7], and indeed a normal digital computer.

Computation is typically thought of as a discrete sequential process that is governed by an algorithm [42], but for such a process to make sense in $\mathcal{S}$ the relevant evolution in $\mathcal{S}$ would have to happen in accordance to a discrete, ordered, and acyclic notion of time. Not only does this effectively preclude computers which make use of closed time loops [5] (see Subsection 1.1.1 for details), it also does not accurately reflect our understanding of many real-world physical systems. For example, if we want to describe the action of a fluid-mechanical system we use a set of differential equations such as the Navier-Stokes equations [55]. However, such a description does not tell us directly how the system evolves at each moment in time, and since there remains no general solution to the Navier-Stokes equations, we are in general unable to extract an algorithm detailing its evolution.

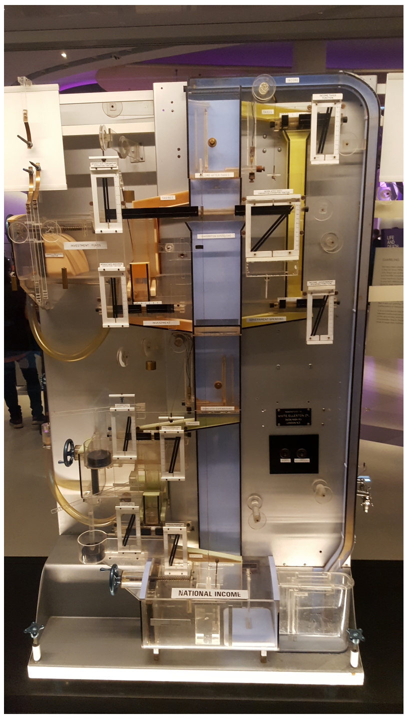


Figure 1.1: MONIAC, located in the science museum, London (Picture copyright: Richard Whyman)

An example of a fluid-based computer is the monetary national income analogue computer (MONIAC) [14] (Figure 1) which utilises water and a collection of containers and valves to simulate the UK economy.

Now the quantum computer [50, 59, 63] is an example of a physical computation system which does *appear* to act in accordance to a discrete, ordered, and acyclic notion of time. However the manner in which quantum computers compute is quite different to how a Turing machine computes (see Sections 2.2 and 2.4 for details), and though a Turing machine is able to simulate any quantum computation [50, 59], the simulation appears to be far more complex than the original computation. Indeed, whilst the class of problems that are computable by a quantum computer is equal to the class of Turing machine computable problems, the class of problems that are *feasibly* computable by a quantum computer appears to be larger that of a Turing machine [63].

Going beyond systems that are clearly physically realisable, there are many examples of computational systems that are more powerful than a Turing machine. Perhaps the most reasonable of these is the type-2 machine of computable analysis [71, 72]. A type-2 machine is essentially a Turing machine that has been modified to have infinite inputs and outputs (see Section 2.3 for details). Hence type-2 machines can clearly compute problems that are not Turing machine computable.

Type-2 machines are also capable of computing with real numbers, as each real number can be encoded as an infinite word via its binary expansion. However, a key aspect of type-2 machines is that at every finite stage the output of a computation is computable by a regular Turing machine. So in this sense a type-2 machine can be simulated by a Turing machine, and so in some respects a type-2 machine is not more powerful than a Turing machine and the existence of a such a device in the real world would not violate the Church-Turing thesis. However this also means that a type-2 machine is in general unable to decide whether two real numbers are equal to one another, as whilst $0.999\ldots = 1$, if $a = 0.999\ldots$ and $b = 1.000\ldots$ then a type-2 machine cannot know that $a = b$ by looking at only a finite number of the symbols in a and b.

Another form of super-Turing computer is the Blum-Shub-Smale (BSS) machine [22, 52], which performs algebraic computations over an arbitrary (ordered) ring. The BSS machine was devised by Blum, Shub, and Smale in 1989 [18, 19]. Unlike a type-2 machine, a BSS machine that acts over the real numbers $\mathbb{R}$ is capable of deciding whether two arbitrary real numbers are equal to one another in a single step. Conversely, type-2 machines are able to compute the real exponential function, but a real BSS machine cannot [22], so in some sense they are incomparable.

In 2000 Hamkins and Lewis presented the infinite time Turing (ITT) machine [43, 44] which acts like a Turing machine but is able to compute in an ordinal number of time steps (see Section 2.5 for details). ITT machines are capable of computing any type-2 machine computable problem, as well as *any* arithmetical relation (Theorem 7.2.5). Which means that ITT machines are even more powerful than type-2 machines, and BSS machines, and are able to decide problems that a Turing machine has no way of computing.

Some infinite computations that occur in the manner of an ITT machine may in fact be physically realisable. As it is has been proposed [57] that a computer passing through the event horizon of a black hole and into another universe could in theory compute a problem in finite time (from the appropriate observer's perspective) that is not Turing machine computable.

In recent work Beggs and Tucker have proposed various physical experiments that could, in theory, be used to compute problems which are not Turing computable [8, 9, 10, 11, 12]. Arguing that with unbounded precision such experiments should be able to efficiently decide any problem in $\mathcal{P}/\log\star$. A mathematical formalisation of these experimental processes was given by us in [73][1].

Beyond this, in 2012 Koepke and Seyfferth [52] combined BSS and ITT machines to obtain infinite time Blum-Shub-Smale machines, which are able to compute any sets of real numbers in the $\mathbf{L}_{\omega^\omega}$ level of Gödel's constructible universe. Going even further Koepke and Koerwien in [51] presented the concept of an ordinal computer, which acts like a Turing machine but has tapes of ordinal length and acts in an ordinal number of time steps. Ordinal computers are capable of deciding any set of ordinals in the whole of the constructible universe $\mathbf{L}$ [51].

So given everything we have mentioned above, it can be seen that there exist multiple different mathematical formalisms which claim to describe some form of computation. Some of these formalisms are inequivalent to one another, some of these formalisms claim to describe some form of "physical" computation, and some of these "physical" computation systems are inequivalent to each other. However what does not appear to defined is an overarching computational formalism that would allow one to characterise any of these forms of computation, as well as enable one to compare different computational systems to one another.

In this document we present the concept of a *theory machine*[2], which is intended to provide this overarching computational formalism, and allow us to characterise

[1]In should be noted that the mathematical formalism presented in [73] will not be appearing in this document.

[2]Despite the similar name, theory machines are not related to the concept of a "logic theory machine" found in [58].

any form of computational system. Theory machines are inspired by (and serve as a generalisation of) Gurevich's concept of a sequential algorithm [42], and Hosman et al.'s reasoning on physical computation [48] (both of which we describe below).

We originally presented theory machines in our paper *Physical Computation and First-Order Logic* [75] and further developed the concept in our paper *An Atemporal Model of Physical Complexity* [74]. The formal definition of a theory machine will be given in Chapter 4.

1.1 Sequential and Non-sequential Algorithms

In attempt to characterise the computational processes required to implement any deterministic sequential algorithm Gurevich introduced the concept of a sequential abstract state machine in *Logic and the Challenge of Computer Science* [41]. Gurevich further demonstrated in *Sequential abstract-state machines capture sequential algorithms* [42] that every sequential algorithm can be step-for-step simulated by an appropriate sequential abstract state machine.

Gurevich's proof of this result depended on the assertion that a sequential algorithm should satisfy the following 3 postulates [42]:

1. Sequential time: Each sequential algorithm A is associated with a set of states $\mathcal{S}(A)$, a set of initial states $\mathcal{I}(A) \subseteq \mathcal{S}(A)$, and a one-step transformation $\tau_A : \mathcal{S}(A) \to \mathcal{S}(A)$, which evolves A in discrete time steps.

2. Abstract state: $\mathcal{S}(A)$ and $\mathcal{I}(A)$ are each sets of first-order structures with the same finite vocabulary Υ that are closed under isomorphism. The one-step transformation τ_A does not change the domain of any state, and any isomorphism from a state $X \in \mathcal{S}(A)$ to a state $Y \in \mathcal{S}(A)$ is also an isomorphism from $\tau_A(X)$ to $\tau_A(Y)$.

3. Bounded exploration: There exists a finite set of Υ-ground terms[1] T such that

[1]Definition 3.1.8

if $X, Y \in \mathcal{S}(A)$ coincide over T then the differences between X and $\tau_A(X)$ are the same as the differences between Y and $\tau_A(Y)$.

A computation of a sequential algorithm A is then a finite or infinite sequence of states $X_0, X_1, X_2, \ldots$ from $\mathcal{S}(A)$, where $X_0 \in \mathcal{I}(A)$ and $\tau(X_i) = X_{i+1}$ for all $i \in \mathbb{N}$.

Gurevich justified the first part of the second postulate by arguing that "The huge experience of mathematical logic and its applications indicates that any static mathematical situation can be faithfully described as a first-order structure." Before noting that second-order and higher-order structures can be described in terms of many-sorted first-order structures. Indeed sequential algorithms can still be described using higher order logic, it is just the states that use the semantics of first-order logic.

Notably, the first postulate was not directly justified by Gurevich in [42], instead it serves as more of a definition than an assertion. Now it does make sense that a sequential algorithm should have an associated collection of states/configurations, with some of them corresponding to input states. However the association with a discrete one-step transformation only appears to follow from the fact that only sequential algorithms are considered.

We believe that whilst Gurevich's algorithmic postulates all make sense for describing a classical machine-based computational process, the sequential time postulate and its one-step transformation is not sufficiently general for describing all forms of physical computation. This is because crucially the transformation expects that a physical system's evolution is discrete. Not only does this not fit with our continuous experience of the world, but it also assumes that a classical notion of time is present and acts independently of the physical system in an unchanging, acyclic manner.

Our belief is further justified by the fact that algorithms that are not constrained by a discrete unchanging notion of time have been defined. In *Axiomatising analog algorithms* [21] Bournez, Dershowitz, and Néron defined the analog algorithm, which is a generalisation Gurevich's sequential algorithms to contexts where the states (which are still first-order structures) evolve over a totally ordered monoid (for example the monoid $\mathbb{R}^{\geqslant 0}$). Thereby enabling a characterisation of continuously evolving computational scenarios.

Though Bournez et al.'s analog algorithms do provide a characterisation of continuous computation, they are still limited. As since totally ordered monoids are necessarily acyclic (that is, time is not allowed to loop), Bournez et al.'s analog algorithms are unable to characterise a computation on a closed time-like curve. This is a problem as closed time-like curves exist as solutions to Einstein's theory of general relativity [38], which suggests that we should also be able to implement a physical computation system on such a curve [32].

1.1.1 Non-causal Circuit Algorithms

Now computation on a closed time-like curve may, at first, appear to be logically impossible as a closed time-like computer could output in such a way so as to prevent its computation from ever being initiated, leading to a paradox. A second issue is that computing on a closed time-like curve could lead to an "ex nihilo"[1] [5] appearance of some uncomputed piece of information. For example, a closed time-like computer could output a proof of whether $\mathcal{P} = \mathcal{NP}$, a user might then check that the proof is correct, before inputting it back into the computer, and sending the proof back in time to be outputted. The proof is given, but there is no clear answer as to where it actually came from.

In *Computational tameness of classical non-causal models* Baumeler and Wolf [5], described a model of computation on closed time-like curves in which these issues are overcome by assuming that any such computation is uniquely determined and logically consistent. Uniqueness ensures that any uncomputed information cannot appear without being a necessary result of the process, whereas logical consistency necessarily prevents a paradox from occurring.

Specifically, they defined a deterministic non-causal circuit algorithm A to be a

[1]Out of nothing.

deterministic algorithm which on any input $x \in \{0,1\}^*$ produces a Boolean circuit C_x in which there exists a unique $y \in \{0,1\}^*$ that is a fixed point of C_x. That is, $C_x(y) = y$. If $y = 1z$ for some $z \in \{0,1\}^*$ then A accepts x, otherwise A rejects x.

Baumeler and Wolf proved that the computational power of polynomially time deterministic non-causal circuit algorithms is equal to $\mathcal{UP} \cap$ co-$\mathcal{UP}$ (see Definitions 2.2.12 and 2.2.13 for details), and every problem that is decidable by a deterministic non-causal circuit algorithm is Turing machine computable. So not only is their non-causal computation non-paradoxical, it also does not violate the Church-Turing thesis.

Notably, there exist problems in $\mathcal{UP} \cap$ co-$\mathcal{UP}$ that are efficiently solvable by a quantum computer but have no known polynomial time Turing machine algorithm, such as the factorisation problem [63] (see Section 2.2 for details). This suggests that there may be a non-causal component to the quantum computational speed-up.

Therefore, given all of this, following the arguments of Gurevich and Bournez et al. theory machine computations will occur on logical structures. However, rather than describing each computation as a discrete ordered sequence of structures, in a theory machine the whole of a computation is described via a single consistent structure. Hence any temporal evolution of the machine is described within this structure, and may take on a variety of forms. Indeed, as we shall demonstrate, the inclusion of the evolution within the structure itself allows a theory machine to compute in a consistent non-causal or atemporal manner.

1.2 Constraints on Physical Computation

The algorithmic notions described above each provide abstract descriptions of *potential* computational processes. But what they do not make clear is when a given algorithm can be actually realised by a physical system. Indeed, there might exist physical computational processes that are not describable (or at least not faithfully describable) by such algorithms.

A framework for discussing physically realisable computation was put forward in *When does a physical system compute?* [48] by Horsman, Stepney, Wagner, and Kendon. Horsman et al.'s central assertion was that for any physical system to be computationally usable, a user must possess a representation relation $\mathcal{R}$ from the system to some numerical/logical/linguistic abstraction, reasoning that "without a way of describing objects abstractly, we cannot do science." To then compute with this physical system the user must have a theory $\mathcal{T}$ that describes how the system behaves within the abstraction.

The representation relation and the theory model the physical system, and together they are required to provide a "good" model of the system. So suppose that $\mathbf{p}$ is an initial state of the system and $\mathcal{R}(\mathbf{p})$ is the abstract representation of the system in state $\mathbf{p}$. Let $\mathbf{H}$ be some physical dynamics and $\mathbf{H}_{\mathcal{T}}$ be the representation of $\mathbf{H}$ within the abstract theory. The representation $\mathcal{R}$ and theory $\mathcal{T}$ then provide a "good" model of the dynamics of the system if in general $\mathcal{R}(\mathbf{H}(\mathbf{p})) \approx \mathbf{H}_{\mathcal{T}}(\mathcal{R}(\mathbf{p}))$, so in a sense, if $\mathcal{R}$ and $\mathbf{H}$ commute. Knowledge that this commutation relation is true can be obtained by carrying out multiple experimental tests. However, a user is naturally limited to being able to carry out only a finite number of these tests. Hence for unbounded systems they can only ever be partially certain about the correctness of their theory.

Further, the abstract representation and theory should allow the user to predict the output of the physical computer from any given input. Horsman et al. stressed that "Without this predictive element, a physical system is not a computer... If a computational description of a physical evolution can only be applied post-hoc, then the system has not acted as a computer." [48] So a user cannot carry out a physical process on a system and *then* pick a computational representation for the system.

For example, let $f : \mathbb{N} \to \mathbb{N}$ be an arbitrary function that is not Turing machine computable, and let M be a simple Turing machine that on any input $x \in \{0,1\}^*$ outputs $x0$. Through a post-hoc description we would be able to make M compute f. As for any $n \in \mathbb{N}$ we can take $x \in \{0,1\}^*$ to represent n and interpret $x0$ as representing $f(n)$. It would then by the case that on input n the Turing machine

M outputs $f(n)$ despite M not really doing anything.

We believe that Horsman et al.'s approach is, for the most part, the "correct" approach to physical computation. As it is surely the case that a user is unable to compute with a physical system without some prior theoretical understanding of it. They cannot simply push a button and just hope that their physical computer gives them the correct output.

However, Horsman et al.'s formalism is not particularly explicit, they do not specify any general conditions about how the abstraction of a physical system may be presented. This was of course a deliberate choice, humanity as a whole does not have a complete understanding of physics, so there was no sense in Horsman et al. proposing unjustifiable constraints on their representations. But beyond clearly defining what is *not* physical computation, Horsman et al.'s generality does unfortunately prevent further conclusions from being drawn about what is achievable (both feasibly and in principal) with physical computation.

Theory machines are intended to be able characterise any possible physical computation system. To ensure that this is the case we shall try to make theory machines be as general as possible, so as to not preclude any forms of physical computation. However this does mean that some of the computational systems that are characterisable may well be "unphysical". This is less of a bug and more of a feature of our formalism, as it allows us to also consider hypercomputational frameworks such as infinite time Turing machines and Blum-Shub-Smale machines. Thereby enabling us to see where the boundary between the physical computation and unphysical computation lies.

The generality of theory machines does not conflict with Horsman et al.'s key ideas. Indeed they do note that the inverse of the representation relation, an instantiation from the abstract to physical, is unlikely to always exist, stating that "there is no a priori reason to suppose that there is a physical system corresponding to every model." However in Chapter 8 we do consider a restricted version of theory machines, which we call finite first-order theory machines. We prove that computational systems characterisable by finite first-order theory machines are computationally equivalent to Turing machines. Therefore if we assume that the Church-Turing thesis is true and also applies to physical computation, then any physical computational device may be characterised by a finite first-order theory machine.

1.2.1 Other Uses for our Formalism

Another benefit of our theory machine formalism is that it provides a general way of looking at physical complexity.

Though it has been argued [70] that the class of problems which are *feasibly* computable by a physical system should be equal to the class of problems that are efficiently computable by a Turing machine (denoted by $\mathcal{P}$). The fact that quantum computers appear to be able to feasibly decide problems that are not believed to lie in $\mathcal{P}$, such as the factorisation problem [63], appears to suggest this equality is false.

Further, in [16, 17] Blakey described a collection of *classical* physical devices that are capable of solving the factorisation problem in polynomially bounded space and time. However, Blakey argued that, unlike quantum factorisers, his factorisation systems are not feasibly realisable, as the *precision* needed to implement his systems is required to grow exponentially with the size of the input.

Indeed Blakey argued that "unconventional computers warrant unconventional complexity analysis" and that the general resource usage of physical computation should be measured in more than just time and space alone. For example the energy or precision required by a computation should also be considered.

As we shall see in Chapter 9 theory machines provide us with a natural way of describing the general resource usage of a computation. We then argue that since finite first-order theory machines appear to be able to characterise any physical computational process, a problem is feasibly computable by such a process only if it may be characterised by a theory machine with polynomial resource usage. Notably, we prove in Chapter 9 that the class of problems that can be computed by a finite first-order theory machine with polynomial resources is exactly $\mathcal{NP} \cap$co-$\mathcal{NP}$. Theory machines are able to achieve this feat by efficiently computing in an atemporal/non-causal manner. Since problems such as the factorisation problem appear to lie in $\mathcal{NP} \cap \text{co-}\mathcal{NP} \setminus \mathcal{P}$, this gives additional weight to the argument that there is an atemporal/non-causal component to the quantum computational speed-up.

Besides complexity, our formalism also serves as an actualisation of the concept of model-based computation put forward by Beebe and Ulmann in *Model-based computation* [7, 68]. The idea behind model-based computation is that in order to compute the solution to a given problem we use a a system which acts in a manner that is *analogous* to the problem. For example a scale model of a physical process acts in analogous manner to that process and we can use the model to answer questions about the physical process (see Remark 5.2.1 for details).

1.3 Outline of this Document

The remainder of this document is structured as follows.

- In Chapter 2 we define various standard computational frameworks that are considered in this document.
- In Chapter 3 we give the definition of a logical system. We also give the specific definitions of each of the logical systems used in this document. Further, we define what it means for a logical system to be complete.
- In Chapter 4 we define our principal concept, the theory machine. We also define what it means for a theory machine to compute decision and function problems.
- In Chapter 5 we give examples of theory machines. In particular we explain how theory machines may be used to characterise Turing machines, type-2 machines, and physical systems which are defined via a collection of differential equations.
- In Chapter 6 we discuss the properties of theory machines, as well as how we may combine two theory machines. We also prove in Theorems 6.1.9 and

6.1.12 that if $\mathfrak{LS}$ is a logical system which contains first-order logic, then the class of problems computable by a finite $\mathfrak{LS}$-theory machine is closed under functional concatenation as well as union and intersection.

- In Chapter 7 we give further examples of theory machines, detailing how theory machines may be used to characterise quantum computers, and infinite time Turing machines.
- In Chapter 8 we discuss computation with finite first-order theory machines, and finite $\mathfrak{LS}$-theory machines in which $\mathfrak{LS}$ is a complete logical system. We prove in Theorem 8.2.1 that, for finite problems, the computational power of such theory machines is equivalent to that of a Turing machine. Whereas in Theorem 8.3.3 we prove that for general problems their power is equivalent to that of a type-2 machine. This leads us to argue that these theory machines are able to characterise an arbitrary physical computation system.
- In Chapter 9 we define a concept of complexity for theory machines. We prove in Theorem 9.3.1 that the class of problems that are computable by a finite first-order theory machine with polynomial resources is $\mathcal{NP} \cap \text{co-}\mathcal{NP}$.
- In Chapter 10 we conclude and discuss further work.
- In the Appendix A we define the various sets of axioms, such as the integer successor axioms, that are used repeatedly in this document.
- The end of this document features the references followed by the index.

Chapter 2

Computation Preliminaries

The following definitions are presented for the purposes of clarity and completeness, and may be mostly skim-read by a familiar reader.

Definitions 2.0.1 Let $\mathcal{A}$ be a finite set of symbols, we call such a set an *alphabet*. The set of *words of $\mathcal{A}$ length $n \in \mathbb{N}$* is:

$$\mathcal{A}^n = \{a_1 a_2 \dots a_n \mid a_1, a_2, \dots, a_n \in \mathcal{A}\}.$$

The *set of finite words of $\mathcal{A}$* is then:

$$\mathcal{A}^* = \bigcup_{n \in \mathbb{N}} \mathcal{A}^n.$$

The *set of infinite words of $\mathcal{A}$* is the set:

$$\mathcal{A}^\omega = \{a_1 a_2 \dots \mid a_1, a_2, \dots \in \mathcal{A}\}.$$

The *length* of a finite word is the number of symbols it contains, if $w = b_1 b_2 \dots b_m$ then we denote the length of w by $|w| = m$.

Let $v = c_1 c_2 \dots \in \mathcal{A}^* \cup \mathcal{A}^\omega$, the *concatenation* of w and v is the word $wv = b_1 b_2 \dots b_m c_1 c_2 \dots$.

For any $a \in \mathcal{A}$ we denote the word of length n which consists entirely of a's by a^n, and the infinite word consisting entirely of a's by a^ω.

Definitions 2.0.2 A *total function* $f : A \to B$ is a map that is defined for every element of A.

A *partial function* f from A to B is a total function from some subset $C \subseteq A$ to B. That is, $f(x)$ may not be defined for some $x \in A$. We then write $f :\subseteq A \to B$ for this partial function and denote the set of elements of A for which f is defined by $\mathrm{dom}(f)$, which we refer to as the *domain of* f.

2.1 Computability Theory

Definitions 2.1.1 [4, 27, 65, 67] A *Turing machine* M has a single two-way infinite tape (Figure 2.1) divided into squares (also referred to as *cells*). At each moment in time each cell contains a single symbol from the tape's alphabet. The machine has a *head* which at any moment in time points to exactly one of the tape cells. If the cell pointed to by the head at a given moment in time contains the symbol a then we say that the head is *reading* a at this moment in time. The machine also has a finite set of *internal states* and at each moment in time the Turing machine is in one of these states.

A *configuration* of a Turing machine then consists of the contents of the tape, the current internal state of the machine and the position of the head. The inputs of a Turing machine are finite words taken from specified *input alphabet*. At time 0 the machine begins in a specified *initial state* with the input written on the tape from left to right, and the head pointing to the cell containing the leftmost symbol of the input. The remaining tape cells at time 0 are taken to be *blank*, however for clarity it useful to refer to such cells as containing the *blank symbol* of the tape alphabet. The blank symbol is not part of the input alphabet, hence from looking at the tape it is clear where the input word starts and ends.

To evolve the configuration from one time step to the next the machine follows a finite set of *rules*. At each time step the machine checks its internal state, reads the contents of the cell pointed to by the head, and implements one of its rules accordingly.

Some of the Turing machine's internal states are specified to be *halting states*, and if the machine enters one of these states then it *halts*, meaning that it produces an output and ceases to evolve.

A Turing machine is represented by a 7-tuple $M = (\Lambda, \Pi, \mathbf{b}, \mathcal{A}, s_0, H, \mathcal{R})$ where:

- Λ is a finite alphabet of tape symbols,
- Π is a finite set of internal states,
- $\mathbf{b} \in \Lambda$ is the blank tape symbol,
- $\mathcal{A} \subseteq (\Lambda \setminus \mathbf{b})$ is a finite alphabet of input symbols,
- $s_0 \in \Pi$ is the initial state,
- $H = \langle s \rangle$ or $\langle s, t \rangle$ for halting states $s, t \in \Pi$,
- $\mathcal{R}$ is a set of Turing machine rules.

Each rule in $\mathcal{R}$ is of the form:

$$(t, b; u, c, p) \in (\Pi \setminus H) \times \Lambda \times \Pi \times \Lambda \times \{LEFT, PAUSE, RIGHT\},$$

which is read as "if the machine is in internal state t reading b then go to state u, replace the symbol being pointed to with c, and have the tape head perform the operation p." With $p = LEFT$ standing for "move one square left", $p = PAUSE$ for "stay in the same tape square" and $p = RIGHT$ for "move one square right".

We call a Turing machine M a *deterministic Turing machine* if for any two rules $(t, b; u, c, p), (t', b'; u', c', p') \in \mathcal{R}$ we have:

$$((t, b) = (t', b')) \Rightarrow ((u, c, p) = (u', c', p')).$$

So at each moment in time at most one rule may be implemented by M. In this document, unless otherwise mentioned, all our Turing machines will be deterministic.

If $H = \langle s_a, s_r \rangle$ for some $s_a, s_r \in \Pi$ then M carries out a decision process, and we call s_a the *accepting state* and s_r the *rejecting state*. Suppose that M is a

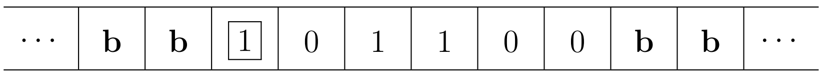


Figure 2.1: An input configuration of a single-tape Turing machine.

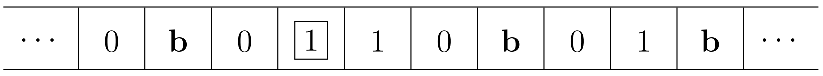


Figure 2.2: If a Turing machine carrying out the computation of a function halted in the above configuration then its output would be 110.

deterministic Turing machine. If on a given input $w \in \mathcal{A}^*$ the machine eventually halts in the accepting state then the output of M is that it "accepts" w, whereas if it eventually halts in the rejecting state then the output of M is that it "rejects" w.

If $H = \langle s_1 \rangle$ for some $s_1 \in \Pi$ then M carries out the computation of a function. If on a given input $w \in \mathcal{A}^*$ the machine reaches s_1 then the output of M consists of the word written on the tape from the right of the tape cell pointed to by the head up to but not including the first blank symbol (See Figure 2.1). This output word is then denoted by $M(w)$. If M never reaches a halting state from input w then this output is undefined.

Definition 2.1.2 Let $a \in \{*, \omega\}$. A *decision problem* is a subset $A \subseteq \mathcal{A}^a$. If $a = *$ then A is a *finite* decision problem, otherwise if $a = \omega$ then A is an *infinite* decision problem.

Definition 2.1.3 [65] Let M be a Turing machine with input alphabet $\mathcal{A}$. We say that M *computes* a finite decision problem $A \subseteq \mathcal{A}^*$ if for any $w \in \mathcal{A}^*$ we have:

$$(w \in A \iff M \text{ accepts } w) \quad \text{and} \quad (w \notin A \iff M \text{ rejects } w).$$

If there exists a Turing machine that computes A then we say that A is *Turing machine computable*.

Definition 2.1.4 Let $a, b \in \{*, \omega\}$. A *function problem* is a partial function $f :\subseteq \mathcal{A}^a \to \mathcal{B}^b$ from a set of words $\mathcal{A}^a$ to a set of words $\mathcal{B}^b$. If $a, b = *$ then f is a *finite* function problem.

Definition 2.1.5 [65] Let M be a Turing machine with input alphabet $\mathcal{A}$ and blank symbol $\mathbf{b} \notin \mathcal{B}$. We say that M *computes* a finite function problem $f :\subseteq \mathcal{A}^* \to \mathcal{B}^*$ if for any $w \in \mathrm{dom}(f)$ we have:

$$M(w) = f(w).$$

If there exists a Turing machine that computes f then we say that f is *Turing machine computable*.

The condition above that the blank symbol of M is not part of $\mathcal{B}$ is a necessity as it is not possible for a Turing machine to output a word that contains its blank symbol.

Remark 2.1.6 Since a Turing machine program is finite, in a given alphabet there is a countable number of possible Turing machine programs. However the sets of possible decision problems and possible function problems are both uncountable. Which means that there must exist problems that are not Turing machine-computable.

Definition 2.1.7 A *finite problem* is either a finite decision problem or a finite function problem. A *general problem* is either a decision problem or a function problem.

Definition 2.1.8 [65] A *multi-tape Turing machine* M_m is a Turing machine with $m \geqslant 2$ tapes (Figure 2.1), each with their own head. At time 0 the input is written on only the first tape, with every other cell starting off blank.

A multi-tape Turing machine is represented by the 7-tuple $M_m = (\Lambda, \Pi, \mathbf{b}, \mathcal{A}, s_0, H, \mathcal{P})$ where as in Definition 2.1.1; Λ is a finite alphabet of tape symbols, Π is a finite set of internal states, $\mathbf{b} \in \Lambda$ is the blank tape symbol, $\mathcal{A} \subseteq (\Lambda \setminus \mathbf{b})$ is a finite alphabet of input symbols, $s_0 \in \Pi$ is the initial state, and H is a 1 or 2-tuple of halting states. Though unlike a Turing machine, $\mathcal{P}$ is a set of multi-tape Turing machine rules.

Each multi-tape Turing machine rule in $\mathcal{P}$ is of the form:

$$(t, \vec{b}; u, \vec{c}, \vec{p}) \in (\Pi \setminus H) \times \Lambda^m \times \Pi \times \Lambda^m \times \{LEFT, PAUSE, RIGHT\}^m,$$

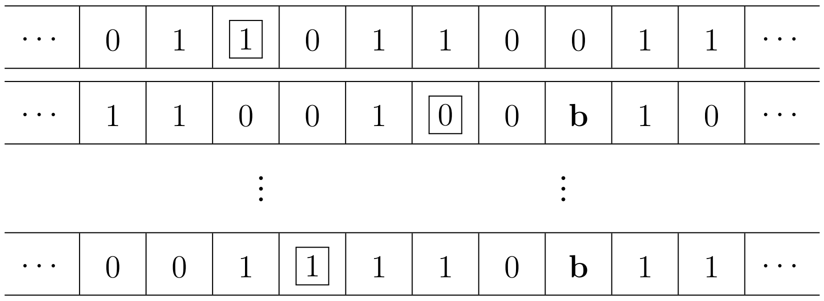


Figure 2.3: A multi-tape Turing machine configuration.

which is read as "if the machine is in internal state t reading $\vec{b}$ from the tapes then go to state u, replace the symbols being pointed to with $\vec{c}$, and have the tape head perform the operation $\vec{p}$ on the tapes."

If M_m is computing a function then the output is written on the mth tape. Slightly unusually, here we shall take the output to be the word written rightwards from the cell where mth tape head is at time 0[1]. The output word of M_m on input $w \in \mathcal{A}^*$ is denoted by $M_m(w)$.

Like with Turing machines, the multi-tape Turing machines in this document will be deterministic, so for any $(t, \vec{b}; u, \vec{c}, \vec{p}), (t', \vec{b}'; u', \vec{c}', \vec{p}') \in \mathcal{P}$ we have:

$$((t, \vec{b}) = (t', \vec{b}')) \Rightarrow ((u, \vec{c}, \vec{p}) = (u', \vec{c}', \vec{p}')).$$

Remark 2.1.9 For any alphabet $\mathcal{A}$, if $|\mathcal{A}| \geqslant 2$ then for any $n \in \mathbb{N}$ there exists a computable bijective function $\underbrace{\langle \cdot, \ldots, \cdot \rangle}_{\times n} : \prod_{i=1}^n \mathcal{A}^* \to \mathcal{A}^*$ which encodes elements of $\prod_{i=1}^n \mathcal{A}^*$ as elements of $\mathcal{A}^*$.

We can therefore represent a multi-input decision problem $C \subseteq \prod_{i=1}^n \mathcal{A}^*$ as single-input decision problem $D \subseteq \mathcal{A}^*$ such that for any $(w_1, \ldots, w_n) \in \prod_{i=1}^n \mathcal{A}^*$:

$$(w_1, \ldots, w_n) \in C \iff \langle w_1, \ldots, w_n \rangle \in D.$$

[1]We define the output of a multi-tape Turing machine in this somewhat unusual way as later it will allow us to more easily describe type-2 machines (Definition 2.3.1) and infinite time Turing machines (Definition 2.5.4).

Similarly we may represent a multi-input and output function problem $g :\subseteq \prod_{i=1}^{n} \mathcal{A}^* \to \prod_{j=1}^{m} \mathcal{B}^*$ as single-input and output function problem $h :\subseteq \mathcal{A}^* \to \mathcal{B}^*$ such that for any $(w_1, \ldots, w_n) \in \mathrm{dom}(h)$:

$$g(w_1, \ldots, w_n) = (v_1, \ldots, v_m) \iff h(\langle w_1, \ldots, w_n \rangle) = \langle v_1, \ldots, v_m \rangle.$$

Definitions 2.1.10 A *relation problem* is a subset $R \subseteq \prod_{i=1}^{n} \mathcal{A}^*$.

R is *computable* if there exists some Turing machine M with input alphabet $\mathcal{A}$ such that for any $(w_1, \ldots, w_n) \in \prod_{i=1}^{n} \mathcal{A}^*$:

$$((w_1, \ldots, w_n) \in R \iff M \text{ accepts } \langle w_1, \ldots, w_n \rangle),$$

and:

$$((w_1, \ldots, w_n) \notin R \iff M \text{ rejects } \langle w_1, \ldots, w_n \rangle).$$

Definition 2.1.11 [27] Let $\Sigma_0^0 = \Pi_0^0 = \Delta_0^0$ be the class of computable relations. For each $n \in \mathbb{N}$ let:

- Σ_{n+1}^0 the class of relations of the form $R(\vec{x}) \equiv \exists \vec{y} P(\vec{x}, \vec{y})$ for some $P \in \Pi_n^0$.
- Π_{n+1}^0 the class of relations of the form $R(\vec{x}) \equiv \forall \vec{y} P(\vec{x}, \vec{y})$ for some $P \in \Sigma_n^0$.
- $\Delta_n^0 = \Sigma_n^0 \cap \Pi_n^0$.

We then say that a relation R is *arithmetical* if $R \in \bigcup_{n \in \mathbb{N}} (\Sigma_n^0 \cup \Pi_n^0)$.

Theorem 2.1.12 *[27] For all $n \in \mathbb{N}$:*

$$\Sigma_n^0, \Pi_n^0 \subset \Delta_{n+1}^0 \subset \Sigma_{n+1}^0, \Pi_{n+1}^0.$$

Also:

$$\Sigma_n^0 \neq \Pi_n^0.$$

Hence every relation problem in $\bigcup_{n \in \mathbb{N}} (\Sigma_n^0 \cup \Pi_n^0) \setminus \Delta_0^0$ is not Turing machine-computable.

2.2 Complexity Theory

The definitions appearing in this section, unless otherwise mentioned, can also be found in [4].

Definition 2.2.1 Let $t : \mathbb{N} \to \mathbb{N}$ be an increasing function[1]. A deterministic Turing machine M with input alphabet $\mathcal{A}$ *computes in time* t if for any $w \in \mathcal{A}^*$ the number of time steps between time 0 and the time at which M on input w reaches a halting state is at most $t(|w|)$.

Definition 2.2.2 Let $u : \mathbb{N} \to \mathbb{N}$ be an increasing function. A deterministic Turing machine M with input alphabet $\mathcal{A}$ *computes in space* u if for any $w \in \mathcal{A}^*$ the number of tape cells that are used in a computation[2] with input w is at most $u(|w|)$.

Remark 2.2.3 Clearly if M computes in time t then on any input w the number of tape cells that are used in the computation is at most $t(|w|) + |w|$ as the machine cannot visit more than one tape cell at each time step.

Notation 2.2.4 Let $f, g : \mathbb{N} \to \mathbb{N}$ be functions, we write:

$$f(n) = O(g(n)),$$

if there exists $c, N \in \mathbb{N}$ such that for any $n \geqslant N$ we have $f(n) \leqslant c \times g(n)$.

Definition 2.2.5 A function $p : \mathbb{N} \to \mathbb{N}$ is a *polynomial function* if there is some $k \in \mathbb{N}$ such that $p(n) = O(n^k)$.

Definition 2.2.6 A finite word problem $A \subseteq \mathcal{A}^*$ is *polynomial time computable* if there exist a Turing machine M and polynomial function p such that M computes A in time p.

We denote the class of polynomial time computable word problems by $\mathcal{P}$.

[1]That is, for any $m, n \in \mathbb{N}$, if $m \leqslant n$ then $t(m) \leqslant t(n)$.

[2]We say that a tape cell is used if it either contains part of the input or at some point in time during the computation the head of the machine points to that cell

Definition 2.2.7 Let $M = (\Lambda, \Pi, \mathbf{b}, \mathcal{A}, s_0, H, \mathcal{R})$ be such that for some rules $(s, a; k, b, p), (s, a; k', b', p') \in \mathcal{R}$ we have $((k, b, p) \neq (k', b', p'))$. Clearly M is not a deterministic Turing machine and so we refer to M as a *non-deterministic Turing machine*.

So if at a given moment in time M is in state s and its head is reading a then either the rule $(s, a; k, b, p)$ or the rule $(s, a; k', b', p')$ may be implemented by the machine. Hence at the next moment in time there are at least two possible configurations that M may be in. It is therefore possible that M may carry out multiple different computations on the same input.

Let $H = \langle s_a, s_r \rangle$ then M carries out a non-deterministic decision process, with accepting state s_a and rejecting state s_r. If for a given computation path of M on input $w \in \mathcal{A}^*$ the machine eventually halts in state s_a then the output of this computation path is that M "accepts" w. Whereas if on this computation path the machine eventually halts in the rejecting state s_r then the output of the computation path is that M "rejects" w.

Definition 2.2.8 Let M be a non-deterministic Turing machine with input alphabet $\mathcal{A}$. We say that M *computes* a finite decision problem $A \subseteq \mathcal{A}^*$ if for any $w \in \mathcal{A}^*$ we have:

$$w \in A \iff \text{There exists a computation path of } M \text{ on input } w \text{ which accepts } w,$$

and:

$$w \notin A \iff \text{Every computation path of } M \text{ on input } w \text{ rejects } w.$$

So every computation of M on any input halts.

Definition 2.2.9 Let $t : \mathbb{N} \to \mathbb{N}$ be an increasing function. A non-deterministic Turing machine N with input alphabet $\mathcal{A}$ *computes in time* t if for any $w \in \mathcal{A}^*$ and any computation of M on input w, the number of time steps between time 0 and the time at which the computation reaches a halting step is at most $t(|w|)$.

Definition 2.2.10 A word problem $B \subseteq \mathcal{A}^*$ is *non-deterministic polynomial time computable* if there exists a non-deterministic Turing machine N and polynomial function p such that N computes B in time p.

We denote the class of non-deterministic polynomial time computable word problems by $\mathcal{NP}$.

Definition 2.2.11 A word problem $C \subseteq \mathcal{A}^*$ is *co-non-deterministic polynomial time computable* if there exists a non-deterministic Turing machine N and polynomial function p such that N computes $\mathcal{A}^* \setminus C$ in time p.

We denote the class of co-non-deterministic polynomial time computable word problems by co-$\mathcal{NP}$.

Definition 2.2.12 [5, 69] A word problem $D \subseteq \mathcal{A}^*$ is *unambiguous non-deterministic polynomial time computable* if there exists a non-deterministic Turing machine N and polynomial function p such that N computes D in time p, and for any $w \in D$ the accepting computation for M on input w is unique.

We denote the class of unambiguous non-deterministic polynomial time computable word problems by $\mathcal{UP}$.

Definition 2.2.13 A word problem $E \subseteq \mathcal{A}^*$ is *co-unambiguous non-deterministic polynomial time computable* if there exists a non-deterministic Turing machine N and polynomial function p such that N computes $\mathcal{A}^* \setminus E$ in time p, and for any $w \in E$ the accepting computation for M on input w is unique.

We denote the class of co-unambiguous non-deterministic polynomial time computable word problems by co-$\mathcal{UP}$.

An important problem in complexity theory is the *factorisation problem*. The factorisation problem is typically rendered as "given a number $N \in \mathbb{N}$ find a prime factor of N". As for any natural number N there exists a unique prime factorisation, that is a collection of prime numbers $p_1, \ldots, p_l$ such that $p_1 \times \cdots \times p_l = N$.

So the factorisation problem may be rendered as a function problem $f : \{0,1\}^* \to \{0,1\}^*$ such that if $w \in \{0,1\}^*$ is a binary expansion of $N \in \mathbb{N}$ then $f(w) = v$, where v is the binary expansion of one of $p_1, \ldots, p_l$.

However the factorisation problem may also be rendered as a decision problem $F \subseteq \{0,1\}^*$ such that for each $\langle w_1, w_2 \rangle \in \{0,1\}^*$[1], if w_1 and w_2 are binary expansions of $N \in \mathbb{N}$ and $K \in \mathbb{N}$ respectively then $\langle w_1, w_2 \rangle \in F$ if and only if there exists a prime factor p_i of N such that $p_i \leqslant K$.

It is also known that $F \in \mathcal{UP} \cap \text{co-}\mathcal{UP} \subseteq \mathcal{NP} \cap \text{co-}\mathcal{NP}$ [5].

2.3 Type-2 Machines

The type-2 machines of computable analysis [72] (Figure 2.3) generalise the concept of a multi-tape Turing machine by enabling it to compute with infinite input words and produce infinite output words.

Definition 2.3.1 [72] A *type-2 machine* T is a multi-tape machine (Definition 2.1.8) with $m \geqslant 3$ tapes, where the input and output word sets are of two possible types $*$ or ω. Like with multi-tape Turing machines, each input word of a type-2 machine is placed on the first tape rightwards from the head position. If T's input set is of type $*$ then every input word must be finite and followed by an infinite sequence of blank cells. Whereas if T's input set is of type ω then every input word must be infinite and fill the entirety of the first tape to the right of the head.

Rule-wise a type-2 machine behaves exactly like a multi-tape Turing machine with $m \geqslant 3$ tapes. That it, the machine sequentially applies its multi-tape rules in accordance to the contents of its tapes and its internal state.

If T's output set is of type $*$ then every output of T must be a finite word. So like with a multi-tape Turing machine that computes a function, if T given input w reaches the halting state then the output of T is the finite word written on tape m to the right of the cell where the tape head started to the last non-blank symbol. We denote the output of T given input w by $T(w)$ and if T on input w never halts then $T(w)$ is undefined.

[1]Where as in Remark 2.1.9, $\langle \cdot, \cdot \rangle$ is an encoding of $\{0,1\}^* \times \{0,1\}^*$ in $\{0,1\}^*$.

Conversely if T's output set is of type ω then every output of T must be an infinite word. To output an infinite word a type-2 machine must compute forever without halting, eventually filling every cell of tape m to the right of where the head started with a non-blank output symbol. So $T(w)$ is the infinite word of non-blank tape symbols that are eventually written on tape m to the right of where the tape head started. If T on input w eventually halts or does not eventually write a non-blank symbol on every relevant tape cell, then $T(w)$ is undefined.

The mth tape of T is write-only, and after each symbol is written the machine moves right and never changes that symbol[1]. Whereas the input tape is read-only, with the machine never moving leftwards on tape 1[2]. The computation is then carried out on the remaining tape(s).

A type-2 machine is then represented by the 9-tuple $T = (\Lambda, \Pi, \mathbf{b}, \mathcal{A}, s_0, s_1, \mathcal{P}, a, b)$ where $a, b \in \{*, \omega\}$ describe the types of the input and output words sets respectively. Whereas $\Lambda, \Pi, \mathbf{b}, \mathcal{A}, s_0, s_1$ and $\mathcal{P}$ are as they are in Definition 2.1.8 so; Λ is a finite alphabet of tape symbols, Π is a finite set of internal states, $\mathbf{b} \in \Lambda$ is the blank tape symbol, $\mathcal{A} \subseteq (\Lambda \setminus \mathbf{b})$ is a finite alphabet of input symbols, $s_0 \in \Pi$ is the initial state, $s_1 \in \Pi$ is the halting state, and $\mathcal{P}$ is a set of multi-tape Turing machine rules.

Remark 2.3.2 A key aspect of a type-2 machine is that we should be able to stop the computation at any time and still obtain a portion of the output. However this aspect would not make sense if we tried to carrying out a never-ending decision process. As in such a process the machine should only ever output a clear "accept" or "reject". This is why here a type-2 machine is defined to only ever have one halting state, as type-2 machines are typically used to carry out the computation of functions and not the computation of decision problems.

Definition 2.3.3 Let T be a type-2 machine with input alphabet $\mathcal{A}$, blank symbol

[1]This means that we could in theory stop a type-2 machine at any time step of its computation and know that whatever T has already written on tape M must be a correct initial segment of the output word.

[2]Such a restriction allows for the output tape of one type-2 machine to used as the input tape of another.

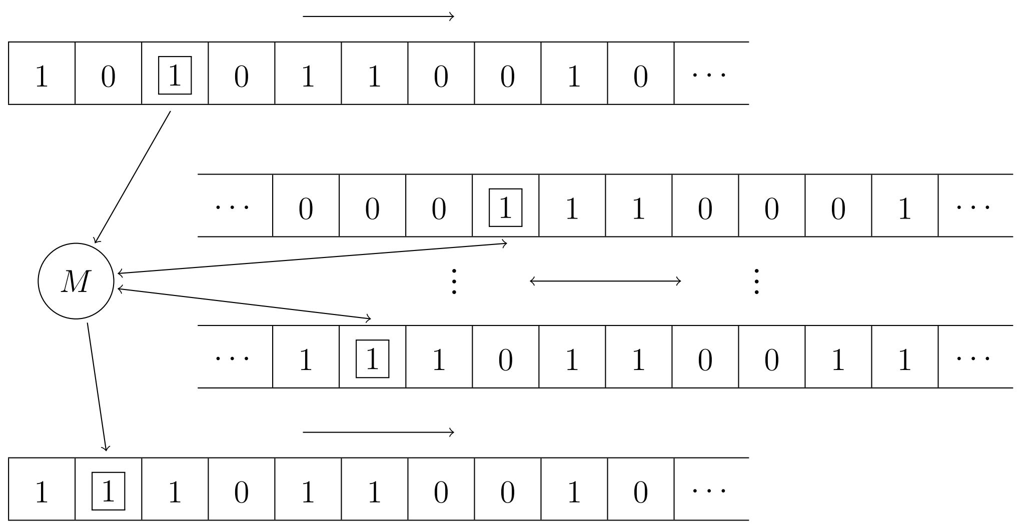


Figure 2.4: A type-2 machine.

$\mathbf{b} \notin \mathcal{B}$, input type a, and output type b. We say that T *computes* a word function problem $f :\subseteq \mathcal{A}^a \to \mathcal{B}^b$ if for any $w \in \mathrm{dom}(f)$ we have:

$$T(w) = f(w).$$

If there exists a type-2 machine that computes f then we say that f is *type-2 computable*.

Proposition 2.3.4 *[72] Let $a, b, c \in \{*, \omega\}$. If the word function problems $f :\subseteq \mathcal{A}^a \to \mathcal{B}^b$ and $g :\subseteq \mathcal{B}^b \to \mathcal{C}^c$ are type-2 computable and $b = *$ or $c = \omega$, then $g \circ f :\subseteq \mathcal{A}^a \to \mathcal{C}^c$ is type-2 computable.*

Otherwise if $b = \omega$ and $c = *$ then for both $a = *$ and $a = \omega$, there exists a type-2 computable function $f :\subseteq \mathcal{A}^a \to \mathcal{B}^\omega$ and a type-2 computable function $g :\subseteq \mathcal{B}^\omega \to \mathcal{C}^*$ such that $g \circ f :\subseteq \mathcal{A}^a \to \mathcal{C}^*$ is not type-2 computable.

2.4 Quantum Computation

Quantum computers utilise quantum objects and quantum transformations to efficiently perform calculations that do not appear to be feasibly implementable by a Turing machine in polynomial time.

A well-studied model of quantum computation is the quantum circuit model, the necessary details of which we will explain below. A full explanation of the model is given in [50, 59], whilst a full introduction to quantum mechanics may be found in [6]. The quantum object that a quantum circuits operate on is called a qubit.

Definitions 2.4.1 A *qubit* is a quantum object whose *state* is an element of the set:

$$\mathcal{Q} = \{\alpha|0\rangle + \beta|1\rangle \mid \alpha, \beta \in \mathbb{C} \text{ and } |\alpha|^2 + |\beta|^2 = 1\}.$$

$\mathcal{Q}$ is a Hilbert space [76] with basis elements $|0\rangle$ and $|1\rangle$, we refer to these elements as the *basis states* for $\mathcal{Q}$.

When multiple qubits are considered they can be entangled, which means that the set of possible states for $N \in \mathbb{N}$ qubits is the tensor product [47]:

$$\underbrace{\mathcal{Q} \otimes \cdots \otimes \mathcal{Q}}_{N \text{ times}} = \mathcal{Q}^{\otimes N} = \left\{ \sum_{k=0}^{2^N-1} \alpha_k |k\rangle \;\middle|\; \alpha_0, \ldots, \alpha_{2^N-1} \in \mathbb{C} \text{ and } \sum_{k=0}^{2^N-1} |\alpha_k|^2 = 1 \right\}.$$

Where we write $|b_{N-1}\rangle|b_{N-2}\rangle \cdots |b_0\rangle = |b_{N-1}b_{N-2} \cdots b_0\rangle = |k\rangle$ if the binary expansion of k is $b_N b_{N-1} \cdots b_0$, so $\mathcal{Q}^{\otimes N}$ has 2^N basis states.

If we consider a qubit with state $\alpha_1|0\rangle + \beta_1|1\rangle$ together with a qubit with state $\alpha_2|0\rangle + \beta_2|1\rangle$ the combined state is:

$$(\alpha_1|0\rangle + \beta_1|1\rangle)(\alpha_2|0\rangle + \beta_2|1\rangle) = \alpha_1\alpha_2|00\rangle + \alpha_1\beta_2|01\rangle + \beta_1\alpha_2|10\rangle + \beta_1\beta_2|11\rangle$$

Definition 2.4.2 A *quantum transformation* $U : \mathcal{Q}^{\otimes N} \to \mathcal{Q}^{\otimes N}$ is a map from the space of N qubits to the space of N qubits. Which means that U must be a unitary transformation. We also refer to a quantum transformation as a quantum *gate*.

Notably, a quantum transformation can be applied to two qubits in such a way that the two qubits cannot be decomposed, for example the joint state may become $\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle \neq (\alpha_1|0\rangle + \beta_1|1\rangle)(\alpha_2|0\rangle + \beta_2|1\rangle)$ for any $\alpha_1, \beta_1, \alpha_2, \beta_2 \in \mathbb{C}$.

Any quantum transformation on $\mathcal{Q}^{\otimes N}$ can be written as a $2^N \times 2^N$ complex-valued matrix, with the kth row and columns corresponding to the kth basis state of $\mathcal{Q}^{\otimes N}$.

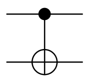
An example of a single qubit quantum gate is the *Hadamard gate*:

$$-\boxed{H}- = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}.$$

The Hadamard gate maps $\alpha|0\rangle + \beta|1\rangle$ to $\frac{\alpha+\beta}{\sqrt{2}}|0\rangle + \frac{\alpha-\beta}{\sqrt{2}}|1\rangle$. Another example of a single qubit quantum gate is the $\frac{\pi}{4}$ *gate*[1]:

$$-\boxed{T}- = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{pmatrix}.$$

The $\frac{\pi}{4}$ gate maps $\alpha|0\rangle + \beta|1\rangle$ to $\alpha|0\rangle + e^{i\frac{\pi}{4}}\beta|1\rangle$. An example of a two qubit quantum gate is the *controlled-not gate*:

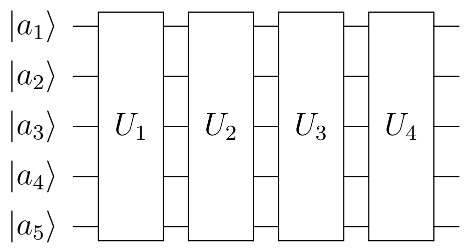


$$= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

The controlled-not gate maps $\alpha|00\rangle + \beta|01\rangle + \gamma|10\rangle + \delta|11\rangle$ to $\alpha|00\rangle + \beta|01\rangle + \delta|10\rangle + \gamma|11\rangle$.

Definition 2.4.3 A *quantum circuit* on N qubits consists of a finite sequence of quantum transformations applied to $\mathcal{Q}^{\otimes N}$. Typically, the qubits inputted into a quantum circuit are either in state $|0\rangle$ or in state $|1\rangle$.

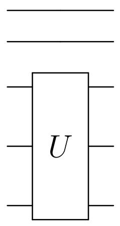
A circuit U_1, U_2, U_3, U_4 applied to 5 qubits in states $|a_1\rangle, |a_2\rangle, |a_3\rangle, |a_4\rangle, |a_5\rangle \in \{|0\rangle, |1\rangle\}$ respectively can be written as:



The output of the above circuit is then $U_4U_3U_2U_1(|a_1a_2a_3a_4a_5\rangle)$.

[1]For historical reasons, sometimes $\frac{\pi}{4}$ gate is often referred to as the $\frac{\pi}{8}$ gate [50, 59]

In many cases a quantum gate U acting on $\mathcal{Q}^{\otimes N}$ will just carry out the identity mapping I on some (but not necessarily all) of the qubits. Which means that it can be decomposed as $U = V \otimes I^{\otimes(2^N - l)}$, where $V : \mathcal{Q}^{\otimes l} \to \mathcal{Q}^{\otimes l}$ is also a unitary transformation. Which means that if for U acting on $\mathcal{Q}^{\otimes 5}$ we have $U = V \otimes I^{\otimes 2}$ then U can be written as:



Naturally if we can perform the transformation U then we can perform $U \otimes I$. We may also permute the qubits in $\mathcal{Q}^{\otimes N}$ before and after applying each gate. Through permutations and tensor products we may construct a large gate set from a small number of gates.

Indeed it so happens that any N-qubit quantum circuit can be efficiently approximated by an N-qubit circuit constructed from just the Hadamard gate, the $\frac{\pi}{4}$ gate and the controlled-not gate [50, 59]. In this document we will therefore assume that any quantum circuit will be constructed from this gate set.

The exact values of the complex state of a qubit cannot typically be measured. Instead a *quantum measurement* of a qubit can only have two possible outputs, these two outputs each correspond to a basis state of $\mathcal{Q}$, and after measuring the state of the qubit will become one of these states. This change in the state due to the measurement means that a quantum measurement cannot be repeated.

In this document we will always take these basis states to be $|0\rangle$ and $|1\rangle$. If we apply a measurement with basis states $|0\rangle, |1\rangle$ to a qubit in state $\alpha|0\rangle + \beta|1\rangle$ then the probability of measuring a $|0\rangle$ is $|\alpha|^2$ and the probability of measuring a $|1\rangle$ is $|\beta|^2$.

Applying a measurement with basis states $|0\rangle, |1\rangle$ to the first qubit of a collection of N qubits in the collective state of $\sum_{k=0}^{2^N-1} \alpha_k |k\rangle$ results in $|0\rangle$ with a probability of $\sum_{j=0}^{2^{N-1}-1} |\alpha_{2j}|^2$ and in $|1\rangle$ with a probability of $\sum_{j=0}^{2^{N-1}-1} |\alpha_{2j+1}|^2$.

Definition 2.4.4 In the quantum circuit model, a *quantum computer* consists of a sequence of quantum circuits $Q = \{Q_N\}_{N\in\mathbb{N}}$ that can be computably constructed [1], and for each $N \in \mathbb{N}$ the circuit Q_N has N inputs.

We input a word $w \in \{0,1\}^*$ into the quantum computer Q as the state $|w\rangle$, to which we apply the quantum circuit $Q_{|w|}$. We then apply a measurement with basis states $|0\rangle, |1\rangle$ to $Q_{|w|}|w\rangle$, the *output* of Q on input w is then the result of this measurement.

Definition 2.4.5 Let Q be a quantum computer and $A \subseteq \{0,1\}^*$ be a decision problem. We say that Q *computes* A if for any input $w \in \{0,1\}^*$:

$$w \in A \iff Q \text{ outputs } |1\rangle \text{ with probability } P > \frac{2}{3},$$

and:

$$w \notin A \iff Q \text{ outputs } |1\rangle \text{ with probability } P \leqslant \frac{1}{3}.$$

Definition 2.4.6 A word problem A is *bounded quantum polynomial time computable* if there exists a quantum circuit Q and a polynomial function $p : \mathbb{N} \to \mathbb{N}$ such that Q computes A, and for any $w \in \{0,1\}^*$ the number of gates in $Q_{|w|}$ is at most $p(|w|)$.

We denote the class of bounded quantum polynomial time computable word problems by $\mathcal{BQP}$.

Let F be the factorisation problem rendered as a decision problem, it is known [63] that $F \in \mathcal{BQP}$.

[1]That is there exists some Turing machine which for any $N \in \mathbb{N}$ is able to compute an exact description of Q_N

2.5 Infinite Time Turing Machines

The concept of an infinite time Turing (ITT) machine [44] was devised by Hamkins and Lewis in 2000. An ITT machine generalises standard Turing machine computation by allowing it to take an ordinal number of time steps. The ordinal numbers are defined as follows

Definition 2.5.1 [25, 64] A *well-ordered set* is a set W with a strict total ordering relation $<$ that is well-ordered. That is in W the relation $<$ satisfies the properties of:

- For any $x, y \in W$, either $(x < y)$, $(x = y)$ or $(x > y)$ is true and no two of these relations are simultaneously true.
- For any $x, y, z \in W$ if $(x < y)$ and $(y < z)$ then $(x < z)$.
- For any non-empty subset $T \subseteq W$ there exists an $x \in T$ such that for any $y \in T \setminus \{x\}$ we have $(x < y)$.

Definition 2.5.2 [25, 64] Two ordered sets W_1, W_2 with respective orderings $<_1$ and $<_2$ have the same *order type* if there exists an order-preserving bijection between W_1 and W_2. That is there exists a total function $f : W_1 \to W_2$ such that for any $x, y \in W_1$, we have $x <_1 y$ in W_1 if and only if $f(x) <_2 f(y)$ is true in W_2.

Definition 2.5.3 [25, 64] An *ordinal number* is an order type of a well-ordered set. The set of *ordinals* ORD is then the set of all ordinal numbers.

Axioms for the ordinals are detailed in Definition A.1.14.

The order type of the empty set is clearly an ordinal, as is the order type of any finite set of the form $\{0, 1, \ldots, n\}$. The set of natural numbers $\mathbb{N}$ is well-ordered and its corresponding ordinal is denoted by ω. Generally, given any ordinal α there exists a successor ordinal $\alpha + 1$. If a well-ordered set W with ordering $<$ has order type α, then the set $W \cup \{a\}$ with ordering $<$, where $x < a$ for any $x \in W$, has order type $\alpha + 1$. The class of ordinals is of the form:

$$ORD = 0, 1, 2, \ldots \omega, \omega + 1, \ldots 2\omega, 2\omega + 1, \ldots$$

So in a sense, the class of ordinals generalises the set of natural numbers. For $\alpha \neq 0$, if there does not exist an ordinal β such that $\beta + 1 = \alpha$ then we refer to α as a limit ordinal, otherwise α is a successor ordinal. ω is an example of a limit ordinal.

Definition 2.5.4 [43, 44] An *infinite time Turing machine* (IIT machine) V is a multi-tape machine with $m \geqslant 3$ tapes. Like a multi-tape Turing machine (Definition 2.1.8) an ITT machine has $m \geqslant 3$ tapes each with their own tape head. Each tape consists of a well-ordered collection of ω cells that contain either a 0 or a 1 (an ITT machine has no blank symbol). Unlike a usual multi-tape Turing machine the tapes of an ITT machine are only infinite in the rightwards direction, and each tape has a left-most cell at position 0 (which allows the tape cells to be indexed by $\mathbb{N}$).

An ITT computation is allowed to take an ordinal number [54] of time steps. At time 0 and successor time steps an ITT machine behaves just like a normal multi-tape Turing machine with tape alphabet $\{0, 1\}$, implementing a multi-tape Turing machine rule depending on what the tape heads see and what the machine's internal state is. Whereas at limit ordinal time steps we make the contents of each tape cell equal to the limit supremum of its previous contents[1]. At limit ordinal times the heads of the machine are placed back at cell 0, and the internal state becomes the special *limit state*.

Like a type-2 machine, the inputs of an ITT machine are infinite words from $\{0, 1\}^\omega$, which are placed on the first tape at time 0. Every cell on every other tape contains 0 at time 0 (see Figure 2.5). Unlike a type-2 machine an infinite output word in $\{0, 1\}^\omega$ can be written (and re-written) in its entirety on the mth tape before the machine halts at some (possibly transfinite) time.

[1]So if δ is a limit time step, and there exists an ordinal $\alpha < \delta$ such that the symbol in cell x is 0 at every time step $\beta \in (\alpha, \delta)$, then at time δ the symbol in cell x must be a 0. Otherwise if for every $\alpha < \delta$ there exists a $\beta \in (\alpha, \delta)$ such that the symbol in cell x is a 1, then at time δ the symbol in cell x must be a 1.

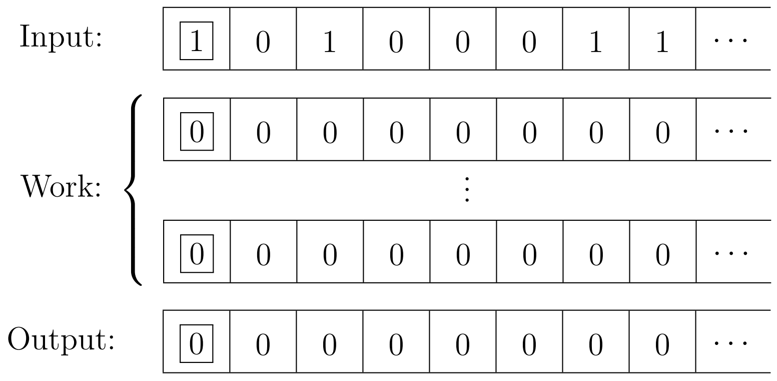


Figure 2.5: The initial configuration of an infinite time Turing machine with an input beginning with 10100011....

We denote an ITT machine by the 5-tuple[1] $V = (\Pi, s_0, s_\lambda, s_1, \mathcal{S})$, where Π is a finite set of internal states, $s_0 \in \Pi$ is the input state, $s_\lambda \in \Pi$ is the limit state, $s_1 \in \Pi$ is the halting state, and $\mathcal{S}$ is the finite set of rules. Each rule of V is an element of $(\Pi \setminus \{s_0\}) \times \{0,1\}^m \times \Pi \times \{0,1\}^m \times \{LEFT, PAUSE, RIGHT\}^m$.

If an ITT machine V given input $w \in \{0,1\}^\omega$ eventually halts, then we denote the infinite word written of the mth tape from the left-most cell by $V(w)$.

Definitions 2.5.5 Let $A \subseteq \{0,1\}^\omega$ be a decision problem. We say that an ITT machine V *computes* A if for any $w \in \{0,1\}^\omega$:

$$(w \in A \iff V(w) = 10^\omega) \text{ and } (w \notin A \iff V(w) = 0^\omega).$$

Let $f :\subseteq \{0,1\}^\omega \to \{0,1\}^\omega$ be a function problem. We say that an ITT machine V *computes* f if for any $w \in \mathrm{dom}(f)$:

$$V(w) = f(w).$$

Proposition 2.5.6 *[44] Every halting ITT machine computation must halt at a countable ordinal time step.*

[1]We do not need to mention the tape or input alphabet of an ITT machine as it is always $\{0,1\}$.

Chapter 3

Logical System Preliminaries

"Rome wasn't built in a day" - Anonymous

In this chapter we formally define what we mean by a logical system, and also define its related notions.

Informally a logical system $\mathfrak{LS}$ is a mathematical formalism with which we may define $\mathfrak{LS}_\mathcal{V}$-structures, $\mathfrak{LS}_\mathcal{V}$-sentences, and a semantic consequence relation ($\models_{\mathfrak{LS}_\mathcal{V}}$). We will use these concepts to define a theory machine and the manner in which it computes.

Examples of logical systems include first-order logic, second-order logic, and modal logic [15, 31].

Typically the concept of a logical system is not formally defined, hence whilst the definitions in this chapter are based on what is generally taken to be true when working with any logical system, they are to some extent our own concepts.

3.1 Vocabularies and Structures

Here, for any logical system $\mathfrak{LS}$, every $\mathfrak{LS}$-structure will be based upon a first-order logical structure [46]. Every such structure includes a vocabulary of symbols, which is sometimes referred to as a similarity-type.

Definition 3.1.1 A *vocabulary* $\mathcal{V}$ is a set of symbols. Each symbol in $\mathcal{V}$ is either:

- An m-ary *relation* symbol, for some $m \in \mathbb{N}$.
- An n-ary *function* symbol, for some $n \in \mathbb{N} \setminus \{0\}$.
- A *constant* symbol.

Definition 3.1.2 Let $\mathcal{V}$ be a vocabulary. A $\mathcal{V}$-*structure* $\mathfrak{M}$ consists of the following:

- A set M, which is referred to as the *domain* of $\mathfrak{M}$ and denoted by $\text{dom}(\mathfrak{M}) = M$.
- For each m-ary relation symbol $R \in \mathcal{V}$, there is an m-ary relation $R : M^m \to \{true, false\}$ in $\mathfrak{M}$. So for any $a_1, \ldots, a_m \in M$ the statement $R(a_1, \ldots, a_m)$ is either true or false in $\mathfrak{M}$.
- For each n-ary function symbol $f \in \mathcal{V}$, there is an n-ary function $f : M^n \to M$ in $\mathfrak{M}$. So for any $a_1, \ldots, a_n \in M$ there is a $b \in M$ such that $f(a_1, \ldots, a_n) = b$ in $\mathfrak{M}$.
- For each constant symbol $c \in \mathcal{V}$, there is a constant c in $\mathfrak{M}$. The constant c is assigned to some element of M.

If $\mathcal{V} = \{R_1, \ldots, R_I\} \cup \{f_1, \ldots, f_J\} \cup \{c_1, \ldots, c_K\}$ where each R_i is a relation, each f_j is a function and each c_k is a constant, then we write:

$$\mathfrak{M} = \langle M; R_1, \ldots, R_I, f_1, \ldots, f_J, c_1, \ldots, c_K \rangle.$$

Notation 3.1.3 When required for the sake of clarity, we denote the relation in a structure $\mathfrak{M}$ which corresponds to the relation symbol R by $R^{\mathfrak{M}}$. Similarly we denote the function in $\mathfrak{M}$ which corresponds to the function symbol f by $f^{\mathfrak{M}}$, and the constant in $\mathfrak{M}$ which corresponds to the constant symbol c by $c^{\mathfrak{M}}$.

We will define the formulas of a logical system as combinations of other formulas. In order for this construction to make sense the variables of all of these formulas will be taken from a fixed set.

Definition 3.1.4 The *set of variables* Ξ is a fixed vocabulary of the form:

$$\Xi = \Xi_{\mathfrak{C}} \cup \Xi_{\mathfrak{R}} \cup \Xi_{\mathfrak{F}}.$$

Where:

$$\begin{aligned}
\Xi_{\mathfrak{C}} &= \{x_1, x_2, \dots \mid \text{Each } x_i \text{ is a constant symbol}\},\\
\Xi_{\mathfrak{R}} &= \{P_1, P_2, \dots \mid \text{Each } P_j \text{ is an } \alpha(j)\text{-ary relation symbol}\},\\
\Xi_{\mathfrak{F}} &= \{g_1, g_2, \dots \mid \text{Each } g_k \text{ is a } \beta(k)\text{-ary function symbol}\},
\end{aligned}$$

and $\alpha : \mathbb{N} \setminus \{0\} \to \mathbb{N}$ and $\beta : \mathbb{N} \setminus \{0\} \to \mathbb{N} \setminus \{0\}$ are functions that denote the arity of each relation and function symbol in Ξ. Also for any $k \in \mathbb{N}$ and any $l \in \mathbb{N} \setminus \{0\}$ the sets $\{z \in \mathbb{N} \mid \alpha(z) = k\}$ and $\{z \in \mathbb{N} \mid \beta(z) = l\}$ are infinite.

We refer to the symbols in Ξ as *variables*.

So Ξ contains an unbounded number of distinct constant symbols. Similarly for any $m \in \mathbb{N}$ and any $n \in \mathbb{N} \setminus \{0\}$ there is an unbounded number of distinct m-ary relation symbols in Ξ, and an unbounded number of distinct n-ary function symbols in Ξ.

Remark 3.1.5 Whenever we refer to a vocabulary $\mathcal{V}$ in this document which is not Ξ we will assume that $\mathcal{V} \cap \Xi = \emptyset$.

Notation 3.1.6 We denote the set of possible relations (of any arity) on a set M by $M_{\mathfrak{R}}$ and the set of possible functions (of any arity) on M by $M_{\mathfrak{F}}$.

Remark 3.1.7 For a finite set $\{1, \dots, L\}$ there are 2^{L^m} possible m-ary relations and $L^{L^{n+1}}$ possible n-ary functions. For an infinite set I the situation is less clear, and contents of $I_{\mathfrak{R}}$ and $I_{\mathfrak{F}}$ depend on the set-theoretic axioms one assumes. For simplicity, in this document we will assume that the standard axioms of Zermelo-Fraenkel set theory ZFC [25] hold.

Definition 3.1.8 Inductively, a $\mathcal{V}$-*term* is a word such that:

- If $c \in \mathcal{V}$ is a constant symbol, then c is a $\mathcal{V}$-term.

- If $x \in \Xi_{\mathcal{C}}$ is a constant variable, then x is a $\mathcal{V}$-term.
- If $f \in \mathcal{V}$ is an n-ary function symbol and $\gamma_1, \ldots, \gamma_n$ are $\mathcal{V}$-terms, then $f(\gamma_1, \ldots, \gamma_n)$ is a $\mathcal{V}$-term.
- If $g \in \Xi_{\mathcal{F}}$ is an n-ary function variable and $\gamma_1, \ldots, \gamma_n$ are $\mathcal{V}$-terms, then $g(\gamma_1, \ldots, \gamma_n)$ is a $\mathcal{V}$-term.
- Nothing else is a $\mathcal{V}$-term.

A $\mathcal{V}$-term τ is a *first-order term* if it does not contain any function variables. τ is a *ground term* if it is a first-order term which does not contain any constant variables.

Notation 3.1.9 We write a term τ which contains constant variables $\vec{x} = (x_1, \ldots, x_k)$ from $\Xi_{\mathcal{C}}$ and function variables $\vec{g} = (g_1, \ldots, g_l)$ from $\Xi_{\mathcal{F}}$ as $\tau(x_1, \ldots, x_k; g_1, \ldots, g_l) = \tau(\vec{x}; \vec{g})$.

We can assign each variable in a $\mathcal{V}$-term τ to an element of M or $M_{\mathcal{F}}$. Under such an assignment τ itself becomes assigned to an element of M when considered in a $\mathcal{V}$-structure with domain M. Formally, this assignment is defined as follows.

Definition 3.1.10 Let $\mathfrak{M}$ be a $\mathcal{V}$-structure with domain M and let $\tau(\vec{x}; \vec{g})$ be a $\mathcal{V}$-term. Let $\vec{a} = a_1 a_2 \ldots \in M^\omega$ an infinite word of elements of M such that each a_i in $\vec{a}$ corresponds to x_i in $\vec{x} = (x_1, \ldots, x_k)$. Let $\vec{h} = h_1 h_2 \ldots \in M_{\mathcal{F}}^\omega$ be an infinite word where each h_j is a $\beta(j)$-ary function on M which corresponds to g_j in $\vec{g} = (g_1, \ldots, g_l)$. $\tau^{\mathfrak{M}}[\vec{a}; \vec{h}]$ is an element of M such that:

- If $\tau = c$, where c is a constant symbol in $\mathcal{V}$, then $\tau^{\mathfrak{M}}[\vec{a}; \vec{h}] = c^{\mathfrak{M}}$.
- If $\tau = x_i$, where x_i is a constant variable, then $\tau^{\mathfrak{M}}[\vec{a}; \vec{h}] = a_i$.
- If $\tau = f(\gamma_1, \ldots, \gamma_n)$, where f is a function symbol in $\mathcal{V}$, and for each $j \in \{1, \ldots, n\}$ γ_j is $\mathcal{V}$-term such that $\gamma_j^{\mathfrak{M}}[\vec{a}; \vec{h}] = b_j$, then $\tau^{\mathfrak{M}}[\vec{a}; \vec{h}] = f^{\mathfrak{M}}(b_1, \ldots, b_n)$.
- If $\tau = g_i(\gamma_1, \ldots, \gamma_n)$, where g_i is a function variable, and for each $j \in \{1, \ldots, n\}$ γ_j is $\mathcal{V}$-term such that $\gamma_j^{\mathfrak{M}}[\vec{a}; \vec{h}] = b_j$, then $\tau^{\mathfrak{M}}[\vec{a}; \vec{h}] = h_i(b_1, \ldots, b_n)$.

We refer to $\tau^{\mathfrak{M}}[\vec{a};\vec{h}]$ as the *assignment of* τ *to* $(\vec{a};\vec{h})$ *in* $\mathfrak{M}$.

Notation 3.1.11 Let $\tau, \delta(x)$ be terms, where δ contains a single constant variable x and no function variables. The word $\delta(\tau)$ denotes that x has been replaced by τ in δ. Let $n \in \mathbb{N}$, the word $\delta^n(\tau)$ then denotes that δ has been placed around τ exactly n-times. So $\delta^0(\tau) = \tau$ and $\delta^{k+1}(\tau) = \delta(\delta^k(\tau))$ for any $k \in \mathbb{N}$.

Definition 3.1.12 An *atomic $\mathcal{V}$-formula* is a word of the form $R(\tau_1, \ldots, \tau_m)$, where $R \in \mathcal{V} \cup \Xi$ is either an m-ary relation symbol or an m-ary relation variable, and $\tau_1, \ldots, \tau_m$ are $\mathcal{V}$-terms.

Let $\mathfrak{M}$ be a $\mathcal{V}$-structure with domain M, and let $\vec{a} \in M^\omega$ and $\vec{h} \in M^\omega_{\mathcal{F}}$.

We write $R(\tau_1, \ldots, \tau_n)[\vec{a};\vec{h}]$ for the *assignment of* $R(\tau_1, \ldots, \tau_n)$ *to* $(\vec{a};\vec{h})$ *in* $\mathfrak{M}$. $R(\tau_1, \ldots, \tau_n)[\vec{a};\vec{h}]$ is *true in* $\mathfrak{M}$ if the relation $R^{\mathfrak{M}}(\tau_1^{\mathfrak{M}}[\vec{a};\vec{h}], \ldots, \tau_n^{\mathfrak{M}}[\vec{a};\vec{h}])$ is true in $\mathfrak{M}$. Otherwise we say that $R(\tau_1, \ldots, \tau_n)[\vec{a};\vec{h}]$ is *false in* $\mathfrak{M}$.

If $\tau_1, \ldots, \tau_m$ are first-order terms then $R(\tau_1, \ldots, \tau_m)$ is a *first-order atomic formula*.

3.2 The Definition of a Logical System

We now come to our definition of a logical system. It should be noted that in this document we only ever make use of logical systems which contain first-order logic (Definition 3.2.4). So whilst the following definition may at first appear to be somewhat vague, every logical system that is used in the following chapters will be well-defined.

Definition 3.2.1 A logical system $\mathfrak{LS}$ consists of the following:

- A set of *logical symbols* $\mathcal{L}$.
- A set of *non-logical symbols* $\mathcal{N}$ which is a vocabulary.
- A collection of conditions for defining *$\mathfrak{LS}$-structures*.

- A collection of rules for constructing $\mathfrak{LG}$-*formulas*.
- A collection of rules for defining the truth of each $\mathfrak{LG}$-formula in each $\mathfrak{LG}$-structure.

Each $\mathfrak{LG}$-structure and each $\mathfrak{LG}$-formula is defined using an additional vocabulary $\mathcal{V}$, where $\mathcal{V} \cap \mathcal{N} = \emptyset$. When such a vocabulary is specified we refer to an $\mathfrak{LG}$-structure as an $\mathfrak{LG}_{\mathcal{V}}$-structure and an $\mathfrak{LG}$-formula as an $\mathfrak{LG}_{\mathcal{V}}$-formula.

Each $\mathfrak{LG}_{\mathcal{V}}$-structure is an $\mathcal{N} \cup \mathcal{V}$-structure in which the elements of $\mathcal{N}$ satisfy the conditions specified by $\mathfrak{LG}$.

Each $\mathfrak{LG}_{\mathcal{V}}$-formula is constructed from elements of $\mathcal{L}$ and $\mathcal{N} \cup \mathcal{V}$-atomic formulas.

Each rule for constructing the $\mathfrak{LG}_{\mathcal{V}}$-formulas is of the form; "if $\phi_1, \ldots, \phi_l$ are $\mathfrak{LG}_{\mathcal{V}}$-formulas and $\phi_1, \ldots, \phi_l$ satisfy conditions $\mathcal{C}$, then the word $\theta_{(1,\ldots,l)}$ in $(\{\phi_1, \ldots, \phi_l\} \cup \mathcal{L})^*$ is an $\mathfrak{LG}_{\mathcal{V}}$-formula." Words that cannot be constructed in this manner are not $\mathfrak{LG}_{\mathcal{V}}$-formulas.

Let $\mathfrak{M}$ be an $\mathfrak{LG}_{\mathcal{V}}$-structure with domain M, and let $\vec{a} \in M^{\omega}$, $\vec{Q} \in M^{\omega}_{\mathcal{R}}$, and $\vec{h} \in M^{\omega}_{\mathcal{F}}$. For every $\mathfrak{LG}_{\mathcal{V}}$-formula ψ, we either have $\mathfrak{M} \models_{\mathfrak{LG}_{\mathcal{V}}} \psi\big[\vec{a}; \vec{Q}; \vec{h}\big]$, in which case we say that ψ *is true in* $\mathfrak{M}$, or $\mathfrak{M} \not\models_{\mathfrak{LG}_{\mathcal{V}}} \psi\big[\vec{a}; \vec{Q}; \vec{h}\big]$ in which case we say that ψ *is false in* $\mathfrak{M}$.

If an atomic $\mathfrak{LG}_{\mathcal{V}}$-formula ϕ is true in $\mathfrak{M}$ under assignment $[\vec{a}; \vec{h}]$, then we write:

$$\mathfrak{M} \models_{\mathfrak{LG}_{\mathcal{V}}} \phi\big[\vec{a}; \vec{Q}; \vec{h}\big],$$

for any $\vec{Q} \in M^{\omega}_{\mathcal{R}}$. Otherwise if $\phi[\vec{a}; \vec{h}]$ is false in $\mathfrak{M}$, we then write:

$$\mathfrak{M} \not\models_{\mathfrak{LG}_{\mathcal{V}}} \phi[\vec{a}; \vec{Q}; \vec{h}].$$

For each non-atomic $\mathfrak{LG}_{\mathcal{V}}$-formula $\theta_{(1,\ldots,l)}$ constructed from $\phi_1, \ldots, \phi_l$ there is a rule of $\mathfrak{LG}$ of the form:

$$\mathfrak{M} \models_{\mathfrak{LG}_{\mathcal{V}}} \theta_{(1,\ldots,l)}[\vec{a}; \vec{Q}; \vec{h}] \iff \mathcal{S}(\phi_1, \ldots, \phi_l; \big[\vec{a}; \vec{Q}; \vec{h}\big]).$$

Where $\mathcal{S}$ is some statement, the truth of which depends on the truth of $\mathfrak{M} \models_{\mathfrak{LG}_{\mathcal{V}}} \phi_i\big[\vec{a}'; \vec{Q}'; \vec{h}'\big]$ for each $i \in \{1, \ldots, l\}$ and some collection of assignments related to $\big(\vec{a}; \vec{Q}; \vec{h}\big)$.

Notation 3.2.2 If ϕ is a first-order formula which does not contain any relation or function variables, then for any $\mathfrak{LS}_{\mathcal{V}}$-structure $\mathfrak{M}$, we may write $\mathfrak{M} \models_{\mathfrak{LS}_{\mathcal{V}}} \phi[\vec{a}]$ instead of $\mathfrak{M} \models_{\mathfrak{LS}_{\mathcal{V}}} \phi[\vec{a}; \vec{Q}; \vec{h}]$.

3.2.1 Logical Systems that we will use

In this subsection we will define each of the logical systems that will we use in this document. For simplicity, all of these logical systems we will be based on first-order logic [46].

Definition 3.2.3 The *logical symbol set of first-order logic* is $\mathcal{L}_{FO} = \{\neg, \wedge, \vee, \rightarrow, \leftrightarrow, \forall, \exists\}$.

Definition 3.2.4 A logical system $\mathfrak{LS}$ with non-logical symbol set $\mathcal{N}$ *contains first-order logic*, if the following is true for $\mathfrak{LS}$ and a vocabulary $\mathcal{V}$:

- $\mathcal{L}_{FO} \subseteq \mathcal{N}$.
- Every first-order atomic $\mathcal{V} \cup \mathcal{N}$-formula is an $\mathfrak{LS}_{\mathcal{V}}$-formula.
- If ϕ and θ are $\mathfrak{LS}_{\mathcal{V}}$-formulas, then $\neg\phi$, $\phi \wedge \theta$, $\phi \vee \theta$, $\phi \rightarrow \theta$, and $\phi \leftrightarrow \theta$ are $\mathfrak{LS}_{\mathcal{V}}$-formulas. Also for any $x_i \in \Xi_{\mathfrak{C}}$ if ϕ does not contain $\forall x_i$ or $\exists x_i$ then $\forall x_i \phi$ and $\exists x_i \phi$ are $\mathfrak{LS}_{\mathcal{V}}$-formulas. We refer to $\mathfrak{LS}_{\mathcal{V}}$-formulas constructed in this manner from first-order atomic $\mathcal{V} \cup \mathcal{N}$-formulas as first-order $\mathcal{V} \cup \mathcal{N}$-formulas.
- For each $\mathfrak{LS}_{\mathcal{V}}$-structure $\mathfrak{A}$ with domain A, each $\vec{a} \in A^{\omega}$, and any $\mathfrak{LS}_{\mathcal{V}}$-formulas ϕ and θ we have:

$$\begin{aligned}
(\mathfrak{A} \models_{\mathfrak{LS}_{\mathcal{V}}} \neg\phi[\vec{a}]) &\iff (\mathfrak{A} \not\models_{\mathfrak{LS}_{\mathcal{V}}} \phi[\vec{a}]),\\
(\mathfrak{A} \models_{\mathfrak{LS}_{\mathcal{V}}} (\phi \wedge \theta)[\vec{a}]) &\iff (\mathfrak{A} \models_{\mathfrak{LS}_{\mathcal{V}}} \phi[\vec{a}] \text{ and } \mathfrak{A} \models_{\mathfrak{LS}_{\mathcal{V}}} \theta[\vec{a}]),\\
(\mathfrak{A} \models_{\mathfrak{LS}_{\mathcal{V}}} (\phi \vee \theta)[\vec{a}]) &\iff (\mathfrak{A} \models_{\mathfrak{LS}_{\mathcal{V}}} \phi[\vec{a}] \text{ or } \mathfrak{A} \models_{\mathfrak{LS}_{\mathcal{V}}} \theta[\vec{a}]),\\
(\mathfrak{A} \models_{\mathfrak{LS}_{\mathcal{V}}} (\phi \rightarrow \theta)[\vec{a}]) &\iff (\mathfrak{A} \models_{\mathfrak{LS}_{\mathcal{V}}} \neg\phi[\vec{a}] \text{ or } \mathfrak{A} \models_{\mathfrak{LS}_{\mathcal{V}}} \theta[\vec{a}]),\\
(\mathfrak{A} \models_{\mathfrak{LS}_{\mathcal{V}}} (\phi \leftrightarrow \theta)[\vec{a}]) &\iff (\mathfrak{A} \models_{\mathfrak{LS}_{\mathcal{V}}} (\phi \rightarrow \theta)[\vec{a}] \text{ and } \mathfrak{A} \models_{\mathfrak{LS}_{\mathcal{V}}} (\theta \rightarrow \phi)[\vec{a}]),\\
(\mathfrak{A} \models_{\mathfrak{LS}_{\mathcal{V}}} \forall x_i \phi[\vec{a}]) &\iff (\text{For every } b_i \in A \text{ we have } \mathfrak{A} \models_{\mathfrak{LS}_{\mathcal{V}}} \phi[\vec{a} \setminus b_i]),\\
(\mathfrak{A} \models_{\mathfrak{LS}_{\mathcal{V}}} \exists x_i \phi[\vec{a}]) &\iff (\text{There exists } b_i \in A \text{ such that } \mathfrak{A} \models_{\mathfrak{LS}_{\mathcal{V}}} \phi[\vec{a} \setminus b_i]).
\end{aligned}$$

Where $\vec{a} \setminus b_i$ denotes the word $\vec{a}$ in which the ith symbol a_i is replaced by b_i.

Definition 3.2.5 *First-order logic* (denoted by FO) is a logical system which contains first-order logic, has logical symbol set $\mathcal{L}_{FO}$, and non-logical symbol set $\emptyset$.

Every $\mathcal{V}$-structure is an $FO_{\mathcal{V}}$-structure. Every $FO_{\mathcal{V}}$-formula is a first-order $\mathcal{V}$-formula.

Definition 3.2.6 *First-order logic with equality* (denoted by $FO^{=}$) is a logical system which contains first-order logic, has logical symbol set $\mathcal{L}_{FO}$, and non-logical symbol set $\{=\}$, where $=$ is a binary relation.

A $\mathcal{V} \cup \{=\}$-structure $\mathfrak{A}$ is an $FO^{=}_{\mathcal{V}}$-structure if for every $a, b \in \text{dom}(\mathfrak{A})$ we have $a =^{\mathfrak{A}} b$ iff a and b are the same element of $\text{dom}(\mathfrak{A})$. Every $FO^{=}_{\mathcal{V}}$-formula is a first-order $\mathcal{V} \cup \{=\}$-formula.

Definition 3.2.7 *First-order real logic* (denoted by $FO\mathbb{R}$) is a logical system which contains first-order logic, has logical symbol set $\mathcal{L}_{FO}$, and non-logical symbol set $\mathcal{N}_{FO\mathbb{R}} = \{=, <, +, \times, 0, 1\}$. Where $=, <$ are binary relation symbols, $+, \times$ are binary function symbols, and 0,1 are constant symbols.

A $\mathcal{V} \cup \mathcal{N}_{FO\mathbb{R}}$-structure $\mathfrak{A}$ is an $FO\mathbb{R}$-structure if it has domain $\mathbb{R}$ and the symbols $=, <, +, \times, 0, 1$ all have their usual meanings in $\mathbb{R}$[1]. Every $FO\mathbb{R}_{\mathcal{V}}$-formula is a first-order $\mathcal{V} \cup \mathcal{N}_{FO\mathbb{R}}$-formula.

Definition 3.2.8 *First-order complex logic* (denoted by $FO\mathbb{C}$) is a logical system which contains first-order logic, has logical symbol set $\mathcal{L}_{FO}$, and non-logical symbol set $\mathcal{N}_{FO\mathbb{C}} = \mathcal{N}_{FO\mathbb{R}} \cup \{\mathbb{R}, i\}$. Where $\mathcal{N}_{FO\mathbb{R}}$ is as in Definition 3.2.7, $\mathbb{R}$ is a unary relation symbol, and i is a constant symbol.

A $\mathcal{V} \cup \mathcal{N}_{FO\mathbb{C}}$-structure $\mathfrak{A}$ is an $FO\mathbb{C}$-structure if it has domain $\mathbb{C}$, the symbols $=, <, +, \times, 0, 1, i$ all have their usual meanings in $\mathbb{C}$, with $\mathbb{R}(a)$ being true iff $a \in \mathbb{R} \subset \mathbb{C}$, and $<$ giving the usual real ordering on this subset [2]. Every $FO\mathbb{C}_{\mathcal{V}}$-formula

[1]Which means that every $FO\mathbb{R}$-structure satisfies the real arithmetic axioms in Definition A.1.7.

[2]Which means that every $FO\mathbb{C}$-structure satisfies the complex arithmetic axioms in Definition

is a first-order $\mathcal{V} \cup \mathcal{N}_{FO\mathbb{C}}$-formula.

Definition 3.2.9 *Second-order logic* (denoted by SO) is a logical system which contains first-order logic, has logical symbol set $\mathcal{L}_{SO} = \mathcal{L}_{FO} \cup \bigcup_{N \in \mathbb{N}} \{\underline{\forall}^N, \underline{\exists}^N\} \cup \bigcup_{L \in \mathbb{N} \setminus \{0\}} \{\overline{\forall}^L, \overline{\exists}^L\}$, and non-logical symbol set $\emptyset$.

Every $\mathcal{V}$-structure is an $SO_{\mathcal{V}}$-structure.

For any $R_j \in \Xi_{\mathfrak{R}}$, if ϕ is an $SO_{\mathcal{V}}$-formula which does not contain $\underline{\forall}^N R_j$ or $\underline{\exists}^N R_j$ then $\underline{\forall}^N R_j \phi$ and $\underline{\exists}^N R_j \phi$ are $SO_{\mathcal{V}}$-formulas. For any $f_k \in \Xi_{\mathfrak{F}}$, if ϕ does not contain $\overline{\forall}^N f_k$ or $\overline{\exists}^N f_k$ then $\overline{\forall}^N f_k \phi$ and $\overline{\exists}^N f_k \phi$ are $SO_{\mathcal{V}}$-formulas.

For each $SO_{\mathcal{V}}$-structure $\mathfrak{A}$ with domain A, each $\vec{a} \in A^\omega$, $\vec{Q} \in A^\omega_{\mathfrak{R}}$, $\vec{h} \in A^\omega_{\mathfrak{F}}$, and any $SO_{\mathcal{V}}$-formulas ϕ and θ we have:

$$
\begin{array}{lcl}
(\mathfrak{A} \models_{SO} \underline{\forall}^N R_j \phi[\vec{a}; \vec{Q}; \vec{h}]) & \iff & \text{For every } N\text{-ary relation } P_j \in A_{\mathfrak{R}} \\
 & & \text{we have } \mathfrak{A} \models_{SO} \phi[\vec{a}; \vec{Q} \setminus P_j; \vec{h}], \\
(\mathfrak{A} \models_{SO} \underline{\exists}^N R_j \phi[\vec{a}; \vec{Q}; \vec{h}]) & \iff & \text{There exists an } N\text{-ary relation } P_j \in A_{\mathfrak{R}} \\
 & & \text{such that } \mathfrak{A} \models_{SO} \phi[\vec{a}; \vec{Q} \setminus P_j; \vec{h}], \\
(\mathfrak{A} \models_{SO} \overline{\forall}^L f_k \phi[\vec{a}; \vec{Q}; \vec{h}]) & \iff & \text{For every } L\text{-ary function } g_k \in A_{\mathfrak{F}} \\
 & & \text{we have } \mathfrak{A} \models_{SO} \phi[\vec{a}; \vec{Q}; \vec{h} \setminus g_k], \\
(\mathfrak{A} \models_{SO} \overline{\exists}^L f_k \phi[\vec{a}; \vec{Q}; \vec{h}]) & \iff & \text{There exists an } L\text{-ary function } g_k \in A_{\mathfrak{F}} \\
 & & \text{such that } \mathfrak{A} \models_{SO} \phi[\vec{a}; \vec{Q}; \vec{h} \setminus g_k].
\end{array}
$$

Definition 3.2.10 *Second-order logic with equality* (denoted by $SO^=$) is a logical system which is the same as second-order logic, except that $SO^=$'s non-logical symbol set is $\{=\}$, and a $\mathcal{V} \cup \{=\}$-structure $\mathfrak{A}$ is an $SO^=_{\mathcal{V}}$-structure if for every $a, b \in \mathrm{dom}(\mathfrak{A})$ we have $a =^{\mathfrak{A}} b$ iff a and b are the same element of $\mathrm{dom}(\mathfrak{A})$.

3.3 Further Logic Definitions

In this section we define some commonly used logical notions [46] in the context of an arbitrary logical system $\mathfrak{L}\mathfrak{S}$ with vocabulary $\mathcal{V}$.

A.1.9.

Definition 3.3.1 We refer to the logical symbols $\forall, \exists \in \mathcal{L}_{FO}$ as *first-order quantifiers* and the logical symbols $\underline{\forall}^N, \underline{\exists}^N, \overline{\forall}^L, \overline{\exists}^L \in \mathcal{L}_{SO}$ for $N \in \mathbb{N}$ and $L \in \mathbb{N} \setminus \{0\}$ as *second-order quantifiers*. Hence a first-order and second-order quantifier is referred to as *quantifier*.

Definition 3.3.2 Let ϕ be an $\mathfrak{LS}_{\mathcal{V}}$-formula. A variable $v \in \Xi$ is *free* in ϕ if ϕ does not contain $\mathcal{Q}v$ for any quantifier $\mathcal{Q}$.

Definition 3.3.3 An $\mathfrak{LS}_{\mathcal{V}}$*-sentence* is an $\mathfrak{LS}_{\mathcal{V}}$-formula that does not contain any free variables.

Notation 3.3.4 If ϕ is an $\mathfrak{LS}_{\mathcal{V}}$-sentence then for any $\mathfrak{LS}_{\mathcal{V}}$-structure $\mathfrak{M}$, we may write $\mathfrak{M} \models_{\mathfrak{LS}_{\mathcal{V}}} \phi$ if $\mathfrak{M} \models_{\mathfrak{LS}_{\mathcal{V}}} \phi[\vec{a}; \vec{Q}; \vec{h}]$, in which case we say that ϕ is true in $\mathfrak{M}$.

Definition 3.3.5 Let $\mathfrak{B}$ be an $\mathfrak{LS}_{\mathcal{V}}$-structure, and Φ, Θ be sets of $\mathfrak{LS}_{\mathcal{V}}$-sentences. We say that $\mathfrak{B}$ is an $\mathfrak{LS}_{\mathcal{V}}$*-model* of Φ if every sentence in Φ is true in $\mathfrak{B}$, and we denote this by $\mathfrak{B} \models_{\mathfrak{LS}_{\mathcal{V}}} \Phi$.

Definition 3.3.6 We say that Φ *semantically implies* Θ *in* $\mathfrak{LS}_{\mathcal{V}}$ if every $\mathfrak{LS}_{\mathcal{V}}$-structure that is an $\mathfrak{LS}_{\mathcal{V}}$-model of Φ is also an $\mathfrak{LS}_{\mathcal{V}}$-model of Θ. We denote this by $\Phi \models_{\mathfrak{LS}_{\mathcal{V}}} \Theta$.

Definition 3.3.7 If for a given set of sentences Θ there is an $\mathfrak{LS}_{\mathcal{V}}$-structure in which Θ is true, then we say that Θ is $\mathfrak{LS}_{\mathcal{V}}$*-satisfiable*.

Definitions 3.3.8 Let $\mathcal{V}_1, \mathcal{V}_2, \mathcal{V}_3$ be vocabularies such that $\mathcal{V}_1 \subset \mathcal{V}_2 \subset \mathcal{V}_3$, and let $\mathfrak{A}_2$ be an $\mathfrak{LS}_{\mathcal{V}_2}$-structure.

A $\mathcal{V}_1$*-reduct* of $\mathfrak{A}_2$ is an $\mathfrak{LS}_{\mathcal{V}_1}$-structure $\mathfrak{A}_1$ such that $\mathrm{dom}(\mathfrak{A}_1) = \mathrm{dom}(\mathfrak{A}_2)$ and for every m-ary relation $R \in \mathcal{V}_1$, n-ary function $f \in \mathcal{V}_1$, and constant $c \in \mathcal{V}_1$ we have that $R^{\mathfrak{A}_1}(a_1, \ldots, a_m)$ is true iff $R^{\mathfrak{A}_2}(a_1, \ldots, a_m)$ is true, $f^{\mathfrak{A}_1}(b_1, \ldots, b_n) = f^{\mathfrak{A}_2}(b_1, \ldots, b_n)$ and $c^{\mathfrak{A}_1} = c^{\mathfrak{A}_2}$.

We say that an $\mathfrak{LS}_{\mathcal{V}_3}$-structure $\mathfrak{A}_3$ is a $\mathcal{V}_3$*-expansion* of $\mathfrak{A}_2$ if $\mathfrak{A}_2$ is a $\mathcal{V}_2$-reduct of $\mathfrak{A}_3$.

Clearly if $\mathfrak{A}_1$ is a $\mathcal{V}_1$-reduct of $\mathfrak{A}_2$ and $\mathrm{dom}(\mathfrak{A}) = A$ then for any $\mathfrak{LS}_{\mathcal{V}_1}$-formula ϕ and each $\vec{a} \in A^\omega$, $\vec{Q} \in A^\omega_{\mathfrak{R}}$, and $\vec{h} \in A^\omega_{\mathfrak{F}}$ we have:

$$(\mathfrak{A}_1 \models_{\mathfrak{LS}_{\mathcal{V}_1}} \phi[\vec{a};\vec{Q};\vec{h}]) \iff (\mathfrak{A}_2 \models_{\mathfrak{LS}_{\mathcal{V}_2}} \phi[\vec{a};\vec{Q};\vec{h}]).$$

Definition 3.3.9 Let $\mathfrak{A}$ and $\mathfrak{B}$ be $\mathfrak{LS}_{\mathcal{V}}$-structures, and let $\mathfrak{LS}$ have non-logical symbol set $\mathcal{N}$. An *embedding* from $\mathfrak{A}$ to $\mathfrak{B}$ is an injective map $\mu : \mathrm{dom}(\mathfrak{A}) \to \mathrm{dom}(\mathfrak{B})$ such that:

- For any m-ary relation $R \in \mathcal{V} \cup \mathcal{N}$ and any $a_1, \ldots, a_m \in \mathrm{dom}(\mathfrak{A})$ we have:

$$R^{\mathfrak{A}}(a_1, \ldots, a_m) \text{ is true in } \mathfrak{A} \iff R^{\mathfrak{B}}(\mu(a_1), \ldots, \mu(a_m)) \text{ is true in } \mathfrak{B}.$$

- For any n-ary function $f \in \mathcal{V} \cup \mathcal{N}$ and any $b_1, \ldots, b_n \in \mathrm{dom}(\mathfrak{A})$ we have:

$$\mu(f^{\mathfrak{A}}(b_1, \ldots, b_n)) = f^{\mathfrak{B}}(\mu(b_1), \ldots, \mu(b_n)).$$

- For any constant $c \in \mathcal{V} \cup \mathcal{N}$ we have $\mu(c^{\mathfrak{A}}) = c^{\mathfrak{B}}$.

Definition 3.3.10 We say that two $\mathfrak{LS}_{\mathcal{V}}$-structures $\mathfrak{A}$ and $\mathfrak{B}$ are *isomorphic* if there exists an embedding μ from $\mathfrak{A}$ to $\mathfrak{B}$ such that its inverse μ^{-1} is an embedding from $\mathfrak{B}$ to $\mathfrak{A}$. We then refer to μ as an *isomorphism*.

For the most part theory machines we will only be able to specify structures up to isomorphism. However the following theorem demonstrates that this is not a problem as the output of a theory machine will depend only on the sentences that are true in its structures.

Theorem 3.3.11 *Let $\mathfrak{LS}$ be one of the logical systems defined in Section 3.2.1 and let $\mathcal{V}$ be a vocabulary. If two $\mathfrak{LS}_{\mathcal{V}}$-structures $\mathfrak{A}$ and $\mathfrak{B}$ are isomorphic, then for any $\mathfrak{LS}_{\mathcal{V}}$-sentence ϕ:*

$$(\mathfrak{A} \models_{\mathfrak{LS}_{\mathcal{V}}} \phi) \iff (\mathfrak{B} \models_{\mathfrak{LS}_{\mathcal{V}}} \phi).$$

Proof: Let $\mathfrak{A}$ and $\mathfrak{B}$ have domains A and B respectively, and $\mu : A \to B$ be an isomorphism from $\mathfrak{A}$ to $\mathfrak{B}$.

If $\mathfrak{LG} \in \{FO, FO^{=}, FO\mathbb{R}, FO\mathbb{C}\}$ then the set of logical symbols of $\mathfrak{LG}$ does not contain any second-order quantifiers, and the $\mathfrak{LG}$-formulas are constructed from first-order terms.

Let $\mathfrak{LG}$ have non-logical symbol set $\mathcal{N}$, and let τ be a first-order $\mathcal{V} \cup \mathcal{N}$-term. We wish to show that $\mu(\tau^{\mathfrak{A}}[\vec{a}]) = \tau^{\mathfrak{B}}[\mu(\vec{a})]$, where if $\vec{a} = a_1 a_2 \ldots \in A^{\omega}$ then $\mu(\vec{a}) = \mu(a_1)\mu(a_2)\ldots \in B^{\omega}$. There are 3 possibilities.

If $\tau = c$, where c is a constant symbol in $\mathcal{V} \cup \mathcal{N}$, then:

$$\mu(\tau^{\mathfrak{A}}[\vec{a}]) = \mu(c^{\mathfrak{A}}) = c^{\mathfrak{B}} = \tau^{\mathfrak{B}}[\mu(\vec{a})].$$

If $\tau = x_i$, where $x_i \in \Xi_{\mathfrak{C}}$, then:

$$\mu(\tau^{\mathfrak{A}}[\vec{a}]) = \mu(a_i) = \tau^{\mathfrak{B}}[\mu(\vec{a})].$$

If $\tau = f(\gamma_1, \ldots, \gamma_n)$, where f is a function symbol in $\mathcal{V} \cup \mathcal{N}$, and for each $j \in \{1, \ldots, n\}$ γ_j is $\mathcal{V}$-term such that $\gamma_j^{\mathfrak{A}}[\vec{a}] = b_j$, then:

$$\mu(\tau^{\mathfrak{A}}[\vec{a}]) = \mu(f^{\mathfrak{A}}(b_1, \ldots, b_n)) = \mu(f)^{\mathfrak{B}}(\mu(b_1), \ldots, \mu(b_n)) = \tau^{\mathfrak{B}}[\mu(\vec{a})].$$

Hence $\mu(\tau^{\mathfrak{A}}[\vec{a}]) = \tau^{\mathfrak{B}}[\mu(\vec{a})]$ follows by induction on the length of τ.

Now let ϕ be an $\mathfrak{LG}_{\mathcal{V}}$-formula of the form $\phi = R(\tau_1, \ldots, \tau_n)$ for some m-ary relation R and some first-order $\mathcal{V} \cup \mathcal{N}$-terms $\tau_1, \ldots, \tau_n$. It is then the case that:

$$\begin{aligned}
\phi[\vec{a}] \text{ is true in } \mathfrak{A} &\iff R^{\mathfrak{A}}(\tau_1^{\mathfrak{A}}[\vec{a}], \ldots, \tau_n^{\mathfrak{A}}[\vec{a}]) \text{ is true in } \mathfrak{A}, \\
&\iff R^{\mathfrak{B}}(\mu(\tau_1^{\mathfrak{A}}[\vec{a}]), \ldots, \mu(\tau_n^{\mathfrak{A}}[\vec{a}])) \text{ is true in } \mathfrak{B}, \\
&\iff R^{\mathfrak{B}}(\tau_1^{\mathfrak{B}}[\mu(\vec{a})], \ldots, \tau_n^{\mathfrak{B}}[\mu(\vec{a})]) \text{ is true in } \mathfrak{B}.
\end{aligned}$$

Therefore for any first-order atomic $\mathfrak{LG}_{\mathcal{V}}$-formula ϕ we have $\mathfrak{A} \models_{\mathfrak{LG}_{\mathcal{V}}} \phi[\vec{a}]$ iff $\mathfrak{B} \models_{\mathfrak{LG}_{\mathcal{V}}} \phi[\mu(\vec{a})]$. This must similarly be true for any $\mathfrak{LG}_{\mathcal{V}}$-formula constructed without quantifiers, as we can just decompose such a formula into its atomic parts, which must each have the same truth value in $\mathfrak{A}$ and $\mathfrak{B}$. Whereas if $\phi = \forall x_i \theta(x_i)$ we then have:

$$\begin{aligned}(\mathfrak{A} \models_{\mathfrak{LG}_{\mathcal{V}}} \forall x_i \theta(x_i)[\vec{a}]) &\iff (\text{For every } b_i \in A \text{ we have } \mathfrak{A} \models_{\mathfrak{LG}_{\mathcal{V}}} \theta(x_i)[\vec{a} \setminus b_i]),\\ &\iff (\text{For every } \mu(b_i) \in B \text{ we have } \mathfrak{B} \models_{\mathfrak{LG}_{\mathcal{V}}} \theta(x_i)[\mu(\vec{a}) \setminus \mu(b_i)])),\\ &\iff (\text{For every } d_i \in B \text{ we have } \mathfrak{B} \models_{\mathfrak{LG}_{\mathcal{V}}} \theta(x_i)[\mu(\vec{a}) \setminus d_i])),\\ &\iff (\mathfrak{B} \models_{\mathfrak{LG}_{\mathcal{V}}} \forall x_i \theta(x_i)[\mu(\vec{a})]).\end{aligned}$$

The same is similarly true for $\phi = \exists x_i \theta(x_i)$. Consequently, by induction on the length of ϕ, the result holds for $\mathfrak{LG} \in \{FO, FO^=, FO\mathbb{R}, FO\mathbb{C}\}$.

Now if $\mathfrak{LG} \in \{SO, SO^=\}$ then the set of logical symbols of $\mathfrak{LG}$ does contain second-order quantifiers, and the $\mathfrak{LG}$-formulas are constructed from general terms.

Let τ be a $\mathcal{V} \cup \mathcal{N}$-term. We now wish to show that $\mu(\tau^{\mathfrak{A}}[\vec{a}; \vec{h}]) = \tau^{\mathfrak{B}}[\mu(\vec{a}); \mu(\vec{h})]$, where if $\vec{h} = h_1 h_2 \ldots \in A^{\omega}_{\mathcal{F}}$ then $\mu(\vec{h}) = \mu(h_1)\mu(h_2) \ldots \in B^{\omega}_{\mathcal{F}}$ where each $\mu(h_j)$ is a $\beta(j)$-ary function such that $\mu(f_j)(\mu(d_1), \ldots, \mu(d_{\beta(j)})) = \mu(f_j(d_1, \ldots, d_{\beta(j)}))$ for any $d_1, \ldots, d_{\beta(j)} \in A$.

For $\tau = c$, x_i, or $f(\gamma_1, \ldots, \gamma_n)$, where c is a constant symbol in $\mathcal{V} \cup \mathcal{N}$, $x_i \in \Xi_{\mathfrak{C}}$, and f is a function symbol in $\mathcal{V} \cup \mathcal{N}$ then we have the required equality by our above reasoning.

Alternatively if $\tau = g_i(\gamma_1, \ldots, \gamma_n)$, where g_i is a function variable, and for each $j \in \{1, \ldots, n\}$ γ_j is $\mathcal{V} \cup \mathcal{N}$-term such that $\gamma_j^{\mathfrak{A}}[\vec{a}; \vec{h}] = b_j$, then:

$$\mu(\tau^{\mathfrak{A}}[\vec{a}; \vec{h}]) = \mu(h_i(b_1, \ldots, b_n)) = \mu(h_i)(\mu(b_1), \ldots, \mu(b_n)) = \tau^{\mathfrak{B}}[\mu(\vec{a}); \mu(\vec{h})].$$

Hence as before $\mu(\tau^{\mathfrak{A}}[\vec{a}; \vec{h}]) = \tau^{\mathfrak{B}}[\mu(\vec{a}); \mu(\vec{h})]$ follows by induction on the length of τ.

Let $\phi = R(\tau_1, \ldots, \tau_n)$ for some m-ary relation R and some $\mathcal{V} \cup \mathcal{N}$-terms $\tau_1, \ldots, \tau_n$. We can then follow the same reasoning as above to see that $\phi[\vec{a}, \vec{h}]$ is true in $\mathfrak{A}$ iff $R^{\mathfrak{B}}(\tau_1^{\mathfrak{B}}[\mu(\vec{a}), \mu(\vec{h})], \ldots, \tau_n^{\mathfrak{B}}[\mu(\vec{a}), \mu(\vec{h})])$ is true in $\mathfrak{B}$.

Therefore for any atomic $\mathfrak{LG}_{\mathcal{V}}$-formula ϕ we have $\mathfrak{A} \models_{\mathfrak{LG}_{\mathcal{V}}} \phi\big[\vec{a}; \vec{Q}; \vec{h}\big]$ iff $\mathfrak{B} \models_{\mathfrak{LG}_{\mathcal{V}}} \phi\big[\vec{a}; \vec{Q}; \vec{h}\big]$. By our above reasoning this must similarly be true for any $\mathfrak{LG}_{\mathcal{V}}$-formula constructed without second-quantifiers.

Suppose $\vec{Q} = Q_1 Q_2 \ldots \in A_{\mathfrak{R}}^{\omega}$ then $\mu(\vec{Q}) = \mu(Q_1)\mu(Q_2)\ldots \in B_{\mathfrak{R}}^{\omega}$ where each $\mu(Q_i)$ is an $\alpha(i)$-ary relation such that $\mu(Q_i)(\mu(b_1), \ldots, \mu(b_{\alpha(i)}))$ is true iff $Q_i(b_1, \ldots, b_{\alpha(i)})$ is true for any $b_1, \ldots, b_{\alpha(i)} \in A$. Now if $\phi = \underline{\forall}^N R_j \theta(R_j)$ then we have:

$$\begin{aligned}
\mathfrak{A} \models_{\mathfrak{LS}_{\mathcal{V}}} \underline{\forall}^N R_j \theta(R_j)\big[\vec{a}; \vec{Q}; \vec{h}\big] \iff & \text{For every } N\text{-ary relation } P_j \in A_{\mathfrak{R}} \\
& \text{we have } \mathfrak{A} \models_{\mathfrak{LS}_{\mathcal{V}}} \theta(R_j)\big[\vec{a}; \vec{Q} \setminus P_j; \vec{h}\big], \\
\iff & \text{For every } N\text{-ary relation } \mu(P_j) \in B_{\mathfrak{R}} \text{ we} \\
& \text{have } \mathfrak{B} \models_{\mathfrak{LS}_{\mathcal{V}}} \theta(R_j)\big[\mu(\vec{a}); \mu(\vec{Q}) \setminus \mu(P_j); \mu(\vec{h})\big], \\
\iff & \text{For every } N\text{-ary relation } T_j \in B_{\mathfrak{R}} \\
& \text{we have } \mathfrak{B} \models_{\mathfrak{LS}_{\mathcal{V}}} \theta(R_j)\big[\mu(\vec{a}); \mu(\vec{Q}) \setminus T_j; \mu(\vec{h})\big], \\
\iff & \mathfrak{B} \models_{\mathfrak{LS}_{\mathcal{V}}} \underline{\forall}^N R_j \theta(R_j)\big[\mu(\vec{a}); \mu(\vec{Q}; \mu(\vec{h})\big]
\end{aligned}$$

The same is similarly true for $\phi = \underline{\exists}^N R_j \theta(R_j)$. Whereas for $\phi = \overline{\forall}^L f_k \theta(f_k)$:

$$\begin{aligned}
\mathfrak{A} \models_{\mathfrak{LS}_{\mathcal{V}}} \overline{\forall}^L f_k \theta(f_k)\big[\vec{a}; \vec{Q}; \vec{h}\big] \iff & \text{For every } L\text{-ary function } g_k \in A_{\mathfrak{F}} \\
& \text{we have } \mathfrak{A} \models_{\mathfrak{LS}_{\mathcal{V}}} \theta(f_k)\big[\vec{a}; \vec{Q}; \vec{h} \setminus g_k\big], \\
\iff & \text{For every } L\text{-ary function } \mu(g_k) \in B_{\mathfrak{F}} \text{ we} \\
& \text{have } \mathfrak{B} \models_{\mathfrak{LS}_{\mathcal{V}}} \theta(f_k)\big[\mu(\vec{a}); \mu(\vec{Q}); \mu(\vec{h}) \setminus \mu(g_k)\big], \\
\iff & \text{For every } L\text{-ary function } e_k \in B_{\mathfrak{F}} \\
& \text{we have } \mathfrak{B} \models_{\mathfrak{LS}_{\mathcal{V}}} \theta(f_k)\big[\mu(\vec{a}); \mu(\vec{Q}); \mu(\vec{h}) \setminus e_k\big], \\
\iff & \mathfrak{B} \models_{\mathfrak{LS}_{\mathcal{V}}} \overline{\forall}^L f_k \theta(f_k)\big[\mu(\vec{a}); \mu(\vec{Q}; \mu(\vec{h})\big]
\end{aligned}$$

The same is similarly true for $\phi = \overline{\exists}^L f_k \theta(f_k)$. Consequently, by induction on the length of ϕ, the result also holds for $\mathfrak{LS} \in \{SO, SO^{=}\}$. ❑

3.4 Proofs in a Logical System

An important thing for us to be able to do with a logical system is prove statements from other statements [34]. We therefore define below what it means for a sentence to be proven in the context of a given logical system $\mathfrak{LS}$. Using this we define what it means for a logical system to be complete which we will be an important concept in Chapter 8.

Definition 3.4.1 An $\mathfrak{LS}$-*proof system* $\mathfrak{P}$ is a finite set of rules of the form:

$$\frac{\Gamma_1 \mid \phi_1, \ldots, \Gamma_K \mid \phi_K}{\Delta \mid \psi}.$$

Where $\Gamma_1, \ldots, \Gamma_K, \Delta$ are sets of $\mathfrak{LS}$-formulas, $\phi_1, \ldots, \phi_K, \psi$ are $\mathfrak{LS}$-formulas, and Δ and ψ depend on $\Gamma_1, \ldots, \Gamma_K$ and $\phi_1, \ldots, \phi_K$.

Example 3.4.2 Examples of FO-proof system rules include:

$$\frac{\Gamma_1 \mid \phi_1,\ \Gamma_2 \mid \phi_2}{\Gamma_1 \cup \Gamma_2 \mid \phi_1 \wedge \phi_2}, \qquad \frac{\Gamma \cup \{\theta\} \mid \phi}{\Gamma \mid \theta \to \phi},$$

$$\frac{\Gamma_1 \cup \{\theta_1\} \mid \phi,\ \Gamma_2 \cup \{\theta_2\} \mid \phi}{\Gamma_1 \cup \Gamma_2 \cup \{\theta_1 \vee \theta_2\} \mid \phi}, \qquad \frac{\Gamma \mid \theta}{\Gamma \cup \Delta \mid \theta}.$$

Where the in the last rule Δ is any set of FO-formulas.

Definition 3.4.3 Let $\mathfrak{P}$ be an $\mathfrak{LS}$-proof system. A $\mathfrak{P}$-*proof* is a finite sequence $\Gamma_1 \mid \phi_1, \ldots, \Gamma_N \mid \phi_N$ where for each $i \in \{1, \ldots, N\}$, Γ_i is a finite set of $\mathfrak{LS}$-formulas, ϕ_i is an $\mathfrak{LS}$-formula, and either $\phi_i \in \Gamma_i$ or there exist $i_1, \ldots, i_K < i$ such that:

$$\frac{\Gamma_{i_1} \mid \phi_{i_1}, \ldots, \Gamma_{i_K} \mid \phi_{i_K}}{\Gamma_i \mid \phi_i}.$$

For some rule of $\mathfrak{P}$.

Definition 3.4.4 Let $\mathfrak{P}$ be an $\mathfrak{LS}$-proof system, and Δ be a set of $\mathfrak{LS}$-sentences and θ be an $\mathfrak{LS}$-sentence. We say that there is a $\mathfrak{P}$-*proof of θ from Δ* if there exists a $\mathfrak{P}$-proof $\Gamma_1 \mid \phi_1, \ldots, \Gamma_N \mid \phi_N$ with $\Gamma_N \subseteq \Delta$ and $\theta = \phi_N$. We denote this by $\Delta \vdash_{\mathfrak{P}} \theta$.

Clearly, given any finite sequence $\Gamma_1 \mid \phi_1, \ldots, \Gamma_N \mid \phi_N$ written as an input word, a Turing machine may check whether the sequence is a valid $\mathfrak{P}$-proof θ from Δ.

Definition 3.4.5 A logical system $\mathfrak{LS}$ is *complete* if there exists an $\mathfrak{LS}$-proof system $\mathfrak{P}$ such that for any set of $\mathfrak{LS}$-sentences Γ and any $\mathfrak{LS}$-sentence ϕ we have:

$$\Gamma \models_{\mathfrak{LS}} \phi \iff \Gamma \vdash_{\mathfrak{P}} \phi.$$

Theorem 3.4.6 *[36, 45] First-order logic, and first-order logic with equality are complete logical systems.*

Chapter 4

Theory Machines

“It is my contention that these operations include all those which are used in the computation of a number. The defence of this contention will be easier when the theory of machines is familiar to the reader.” - Alan Turing [67]

As we discussed in Section 1.2, in *When does a physical system compute?* [48] Horsman, Stepney, Wagner, and Kendon put forward a minimal collection of requirements that a physical system must satisfy in order for it to be capable of computation. Crucially Horsman et al. asserted that in order for a person to be able to compute with a physical system they must be able to abstractly represent the necessary workings of the system, whilst also possessing a sufficiently correct theory of how the system behaves.

We assert that this representation and theory can be expressed in terms of the logical sentences of a *theory machine*. A theory machine is given by a triple $\mathcal{M} = (\mathcal{T}, \mathcal{I}, \mathcal{O})$ where $\mathcal{T}$ is a set of sentences, and $\mathcal{I}$ and $\mathcal{O}$ are sets of sets of sentences[1]. The theory of the system is given by $\mathcal{T}$, which describes the necessary aspects of the system we wish to compute with. The set of admissible inputs into the system is given by $\mathcal{I}$, and the set of measurable outputs from the system is given by $\mathcal{O}$.

[1]To clarify, each element of $\mathcal{I}$ and each element of $\mathcal{O}$ is a set of sentences.

Further we require that for any $\Phi \in \mathcal{I}$ there exists a structure $\mathfrak{P}$ which satisfies $\mathcal{T} \cup \Phi$, and for which there is at most one true output $\Theta \in \mathcal{O}$. We then take Θ to be the outcome of the computation by $\mathcal{M}$ on input Φ. This structure $\mathfrak{P}$ does not need to contain a clear notion of time, nor does Θ need to follow from $\mathcal{I}$ via a clear sequence of algorithmic steps. Hence the typical notion of a sequential causal computation does not necessarily occur within a theory machine. However, we shall insist that the only way Θ can be the output of $\mathcal{M}$ on input Φ is if Θ is true in *every* model of $\mathcal{T} \cup \Phi$. This constraint ensures that the computation cannot in general just happen in one undefined uncomputable step. Instead, as we shall see, the computation must typically have a non-trivial amount of structure to it in order to produce an output.

Suppose that in the real world a person has found/built a computational system $\mathcal{S}$. A key motivation behind the concept of a theory machine is that, for any reasonably correct theory $\mathcal{T}_{\mathcal{S}}$ of $\mathcal{S}$ and any valid input Φ of $\mathcal{S}$, they can expect that the real world will provide them with a structure that satisfies $\mathcal{T}_{\mathcal{S}} \cup \Phi$. If it did not then either $\mathcal{T}_{\mathcal{S}}$ would have to be incorrect or Φ would have to be invalid. So if $\mathcal{S}$ already exists as a real world physical system then we do not have to worry about constructing its models ourselves. What we need to concern ourselves with is constructing and understanding $\mathcal{S}$ in the first place.

The nature of a theory machine computation is intended to mimic what happens when we use a physical system[1] to carry out a decision process. For example, suppose we wish to compute with some kinematic system of billiard balls [13], to do this we can include the axioms of Newtonian mechanics as our theory $\mathcal{T}$ to predict the motions of the system (Newtonian mechanics may not be a perfect description of reality, but in many cases it is more than good enough for describing the necessary properties of a kinematic system). Each input $\Phi \in \mathcal{I}$ could be a non-contradictory description of the positions and velocities of the balls at some initial time t_0 [2]. Whereas each output $\Theta \in \mathcal{O}$ could be a position measurement at some final time t_1.

[1]See Remark 1.0.1 for an explanation of what we mean by a physical system.

[2]This description should state what we *know* to be true about the input configuration, such knowledge is likely to have a degree of uncertainty around it. For example, part of Φ could state that at time t_0 the x-coordinate of the 1st ball is located between rationals $\frac{a}{b}$ and $\frac{a+1}{b}$.

As it is a real physical situation we should always be able to create a kinematic scenario from t_0 to t_1 in which $\mathcal{T} \cup \Phi$ is satisfied. Due to imprecision in the input's description Φ and the inexactness of the theory $\mathcal{T}$ there are likely to be many scenarios that satisfy $\mathcal{T} \cup \Phi$. However, if we know that in each of them only the output Θ is true, then the exact scenario created does not actually matter, all that matters is which element of $\mathcal{O}$ is true given an input of Φ.

The fact that multiple models may exist and all point to the correct output is also key to the theory machine concept (and, notably, differs from Baumeler and Wolf's assertions in [5]). Accepting that our description of a system could have multiple inequivalent models rather than one "true" model allows us to focus on what properties are necessary to impose on the system in order for it to produce the correct output. This also helps us see what parts of the process must be shielded from outside influences for the computation to work.

For example, in the real world an implementation of a Turing machine will exist within a much larger ambient space. This space is likely to include complex objects that may lead to computational errors (such as a person). For the purpose of the computation such objects can be ignored provided we can prevent them from interfering.

Also, for defining an arbitrary decision process this focus on description rather than the individual structures is arguably a necessity. In many real world cases, to know the entirety of a physical system's structure would require an infinite number of tests, which a person would arguably never be able to do. However, as noted by Horseman et al. [48] after a finite number of tests we should be able to infer some of a system's properties and construct our own abstract theory about how it behaves.

In this document we will not dwell upon how one may infer/impose a system's properties, instead we shall take such properties to be known and implementable, and look into how assuming these properties effects what is computationally possible.

We believe that the computational aspects of many different kinds computational device can be described by a theory machine. Indeed in this document we will demonstrate how theory machines can be used to characterise:

- A Turing machine in Examples 5.1.1, 5.1.2, 8.1.1, 8.1.3 and 9.1.6.
- A type-2 machine in Examples 5.3.2, and 8.3.1.
- A quantum computer in Example 7.1.6.
- An infinite time Turing machine in Example 7.2.1.

4.1 The Definition of a Theory Machine

We shall now define our principal concept of a theory machine and its method of computation.

Definitions 4.1.1 Let $\mathfrak{LS}$ be a logical system and $\mathcal{V}$ a vocabulary of $\mathfrak{LS}$, an $\mathfrak{LS}_{\mathcal{V}}$-*theory machine* is a triple $\mathcal{M} = (\mathcal{T}, \mathcal{I}, \mathcal{O})$ where:

- $\mathcal{T}$ is a set of $\mathfrak{LS}_{\mathcal{V}}$-sentences,
- $\mathcal{I}$ and $\mathcal{O}$ are sets of sets of $\mathfrak{LS}_{\mathcal{V}}$-sentences,
- For every $\Phi \in \mathcal{I}$ the set $\mathcal{T} \cup \Phi$ is $\mathfrak{LS}_{\mathcal{V}}$-satisfiable,
- For every $\Phi \in \mathcal{I}$ and $\Theta, \Psi \in \mathcal{O}$ if $\Theta \neq \Psi$ then the set $\mathcal{T} \cup \Phi \cup \Theta \cup \Psi$ is not $\mathfrak{LS}_{\mathcal{V}}$-satisfiable.

We call $\mathcal{T}$ the *theory* of $\mathcal{M}$, call $\mathcal{I}$ the set of *inputs* of $\mathcal{M}$, and $\mathcal{O}$ the set of *outputs* from $\mathcal{M}$.

We say that $\mathcal{M}$ *computes* $\Theta \in \mathcal{O}$ from $\Phi \in \mathcal{I}$ if:

$$\mathcal{T} \cup \Phi \models_{\mathfrak{LS}_{\mathcal{V}}} \Theta.$$

We denote this by $\mathcal{M}(\Phi) = \Theta$. If for $\Theta, \Psi \in \mathcal{O}$ there exist $\mathfrak{LS}_{\mathcal{L}}$-models of $\mathcal{T} \cup \Phi$ where Θ is true and Ψ is true, and $\Theta \neq \Psi$ then $\mathcal{M}$ does not compute anything on input Φ and $\mathcal{M}(\Phi)$ is undefined.

If the logical system does not matter then we may refer to $\mathcal{M}$ as just a *theory machine* and if the logical system does matter but the vocabulary does not then we may refer to $\mathcal{M}$ as an $\mathfrak{LS}$-*theory machine*.

For a given physical computation system described by a theory machine $\mathcal{M}$, the theory $\mathcal{T}$ is intended to detail the laws that the system obeys.

Each element of the input set $\mathcal{I}$ is intended to be a description of some variable input configuration (e.g. the positions of a collection of dials), it could be finite and word-like, it could be an infinite real, it could be a function on reals, or any number of other possibilities. Whatever the case, if an object can be exactly defined by some set of properties then it can be inputted into a theory machine. The same is true for the outputs $\mathcal{O}$, allowing us to take the output from one theory machine and plug it in as an input to another[1].

Example 4.1.2 A set of first-order sentences that defines the real number $c \in [0, 1)$ with binary expansion $0.b_0 b_1 \ldots$ is:

$$\left\{ T^k(c) \bowtie_{b_k} \frac{1}{2} \right\}_{k \in \mathbb{N}}.$$

Where $T(x) = 2x - \lfloor 2x \rfloor$, and $\bowtie_0 \equiv <$ and $\bowtie_1 \equiv \geqslant$ (an explanation of why this works is given in Example 4.2.8). So for example we could have an input or output set of:

$$\left\{ \left\{ T^k(c) \bowtie_{b_k} \frac{1}{2} \right\}_{k \in \mathbb{N}} \;\middle|\; b_0 b_1 \ldots \in \{0, 1\}^{\omega} \right\}.$$

Example 4.1.3 A set of first-order sentences that defines a function from $f : \mathbb{N} \to \mathbb{N}$ is:

$$\{ f(S^n(0)) = S^{f(n)}(0) \}_{n \in \mathbb{N}},$$

where S is the successor function on $\mathbb{N}$.

In a theory machine we require that for any input $\Phi \in \mathcal{I}$ there exists a model of Φ together with the machine's theory. The purpose of this requirement is to ensure that the concept of inputting Φ into the machine actually makes sense. As if no

[1] In Section 6.1 and Theorem 6.1.9 we explain how and when theory machines can be concatenated together.

models existed of $\mathcal{T} \cup \Phi$ then the input configuration of Φ would have to be logically forbidden in such a way that inputting Φ would be physically impossible, and hence would itself not make sense as an input.

Note that this does not mean that an input is not allowed to cause a theory machine to "crash" or "break" in some way. If such a scenario is logically possible then such an input is allowed, and if necessary the machine breaking can be treated as a potential relevant output.

We also intend that for each input a theory machine produces at most one output. Hence we have the fourth condition of Definition 4.1.1, which means that not only can we not have $\mathcal{M}(\Phi) = \Theta$ and $\mathcal{M}(\Phi) = \Psi$ for $\Theta \neq \Psi$, but there do not even exist two separate $\mathfrak{LS}_{\mathcal{V}}$-models of $\mathcal{T} \cup \Phi$ in which Θ and Ψ are true. Consequently if we have a model $\mathfrak{A}$ of $\mathcal{T} \cup \Phi$ and we know that the output $\Theta \in \mathcal{O}$ is true in $\mathfrak{A}$, then we know for certain that $\mathcal{M}(\Phi) = \Theta$ as no other output can be true in $\mathfrak{A}$ if Θ is true in $\mathfrak{A}$.

Example 4.1.4 Let $\mathcal{V} = \{R, f, c\}$ where R is a unary relation, f a unary function, and c a constant. A simple example of a $FO_{\mathcal{V}}$-theory machine[1] is $\mathcal{M} = (\mathcal{T}, \mathcal{I}, \mathcal{O})$ where:

- $\mathcal{T} = \{\forall x(R(x) \leftrightarrow R(f(x)))\}$,
- $\mathcal{I} = \{\{R(c)\}, \{\neg R(c)\}\}$,
- $\mathcal{O} = \{\{R(f(c))\}, \{\neg R(f(f(c)))\}\}$.

We then have:

$$\mathcal{M}(\{R(c)\}) = \{R(f(c))\},$$

as in any model of $\mathcal{T}$, if $R(c)$ is true then $R(f(c))$ must also be true, so $\mathcal{T} \cup \{R(c)\} \models_{FO_{\mathcal{V}}} \{R(f(c))\}$. Whereas:

$$\mathcal{M}(\neg R(c)) = \{\neg R(f(f(c)))\},$$

[1]A FO stands for first-order logic (Definition 3.2.5)

as given $\neg R(c)$ is true in some model of $\mathcal{T}$ then we also have $\neg R(f(c))$ is true in this model and so $\neg R(f(f(c)))$ is true, hence $\mathcal{T} \cup \{\neg R(c)\} \models_{FO_{\mathcal{V}}} \{\neg R(f(f(c)))\}$.

4.2 Describing Words as Sets of Logical Sentences

As many examples of computation systems use words (or word-like objects) to represent their inputs and outputs we naturally require a standard manner in which to write words as sets of logical sentences. Informally, we do this by taking a set which defines each element in a well-behaved sequence of ground terms [31] to be equal to a symbol in the word.

Definition 4.2.1 We call a sequence of ground terms $\mathcal{X} = \{\chi_i\}_{i\in\mathbb{N}}$ a *simple sequence* if there exist a ground term δ and terms $\gamma(x)$ and $\sigma(x)$ with a single free variable x, such that every element of $\mathcal{X}$ is of the form $\chi_i = \gamma(\sigma^i(\delta))$.

The idea behind a simple sequence of terms is that it expresses the repeated application of a function, such as the "next symbol on the right" function. Hence it can be easily and simply constructed. For a unary function f and a constant c the sequence of terms $\{f^i(c)\}_{i\in\mathbb{N}}$ is a simple sequence, as is $\{h((g \circ f)^i(h(c,c)), c)\}_{i\in\mathbb{N}}$ for a unary function g and a binary function h.

Remark 4.2.2 When defining a word as a set of sentences we will make use of the equality relation "=", which satisfies the usual equality axioms (Definition A.1.1) of being an equivalence relation that preserves the functions and relations of the logical system. In an $FO_{\mathcal{V}}$-theory machine $\mathcal{M}$, we may add the set of equality axioms $EQ^{=}_{\mathcal{V}}$ to the theory of $\mathcal{M}$ to ensure that $= \in \mathcal{V}$ is an equivalence relation that preserves the functions and relations of $\mathcal{V}$.

$EQ^{=}_{\mathcal{V}}$ can also be defined in exactly the same manner in any logical system which contains first-order logic[1], and if the number of functions and relations in the vocabulary $\mathcal{V}$ is finite then $EQ^{=}_{\mathcal{V}}$ is also finite.

[1]Such as those logics mentioned in Subsection 3.2.1

However, for simplicity, we will often use logical systems that already contain true equality as part of their fixed set of symbols, in the structures of such a logical system must $EQ^{=}_{\mathcal{V}}$ always be satisfied.

4.2.1 Finite Word Sets

Definitions 4.2.3 Let $\mathcal{X} = \{\chi_i\}_{i\in\mathbb{N}}$ be a simple sequence. For a set of constants $\mathcal{A}$ with $\mathbf{b} \notin \mathcal{A}$, the **finite $\mathcal{X}$-word set** corresponding to the finite word $w = w_0 w_1 \cdots w_n \in \mathcal{A}^*$ is:

$$\Phi^*_{\mathcal{X}}(w) = \{\chi_i = w_i \mid i \in \{0, \ldots, n\}\} \cup \{\chi_{n+1} = \mathbf{b}\}.$$

The **set of finite $\mathcal{X}$-word sets** from an alphabet $\mathcal{A}$ is then:

$$\hat{\mathcal{A}}^*_{\mathcal{X}} = \{\Phi^*_{\mathcal{X}}(w) \mid w \in \mathcal{A}^*\}.$$

Hence a finite $\mathcal{X}$-word set $\Phi^*_{\mathcal{X}}(w)$ maps each term χ_i of $\mathcal{X}$ to the ith symbol in w. So whilst $\mathcal{X}$ may be "simple" the $\mathcal{X}$-word set may be arbitrarily complex.

Remark 4.2.4 The symbol $\mathbf{b}$ is intended to represent the "blank" symbol, hence $\chi_{n+1} = \mathbf{b}$ implies that this is the end of the word. Note that if $\chi_i = \gamma(\sigma^i(\delta))$ then by adding the sentence $\forall x((\gamma(x) = \mathbf{b}) \to (\gamma(\sigma(x)) = \mathbf{b}))$ to the theory of a machine with inputs from $\hat{\mathcal{A}}^*_{\mathcal{X}}$ we can ensure that $\chi_j = \mathbf{b}$ for each $j > n$.

We can describe the initial tape configuration of a Turing machine as a finite $\mathcal{X}$-word set as follows.

Example 4.2.5 Let C be a binary function, S be unary functions, and 0 be a constant. Suppose that we are describing a Turing machine-like scenario and for each time step x and cell y the value of $C(x, y)$ corresponds to the contents of the yth tape square at time x. To define the values of these tape squares at time 0 we may use the simple sequence:

$$\mathcal{X}_{TM} = \{C(0, S^n(0))\}_{n\in\mathbb{N}}.$$

As suppose that we have a second-order theory machine whose theory contains the Peano successor axioms (Definition A.1.3) with S as the successor function. The finite $\mathfrak{X}_{TM}$-word set:

$$\Phi^*_{\mathfrak{X}_{TM}}(w) = \{C(0, S^i(0)) = w_i \mid i \in \{0, \ldots, n\}\} \cup \{C(0, S^{n+1}(0)) = \mathbf{b}\},$$

then implies that:

$$C(0,0) = w_0,\ C(0,1) = w_1,\ C(0,2) = w_2,\ \ldots,\ C(0,n) = w_n,\ \text{and } C(0,n+1) = \mathbf{b},$$

where $C(0, n+1) = \mathbf{b}$ may be viewed as stating that the $n+1$th cell square is blank at time 0.

If the theory machine also contains the sentence:

$$\forall x((C(0,x) = \mathbf{b}) \to (C(0, S(x)) = \mathbf{b})),$$

then $\Phi^*_{\mathfrak{X}_{TM}}(w)$ also implies that $C(0,m) = \mathbf{b}$ for all $m > n$.

It is natural to expect that $\chi_i = a$ and $\chi_j = b$ means that $\chi_i \neq \chi_j$, however without specifying that $a \neq b$ it is entirely possible to have a model where two distinct constants are equal to one another. To avoid the issues that this would cause we will often use the following set of sentences.

Definition 4.2.6 Let $\mathcal{V}$ be a vocabulary and $= \in \mathcal{V}$ be a binary relation. The set of *distinct constant axioms for* $\mathcal{V}$ is:

$$CD^{=}_{\mathcal{V}} = \{\neg(c = d) \mid c, d \in \mathcal{V} \text{ are constants and } c \neq d\}.$$

So clearly if $\mathfrak{LS}$ contains first-order logic and $\mathfrak{A}$ is a $\mathfrak{LS}_{\mathcal{V}}$-structure which satisfies $EQ^{=}_{\mathcal{V}} \cup CD^{=}_{\mathcal{V}}$ then every constant in $\mathfrak{A}$ is distinct.

4.2.2 Infinite Word Sets

Definitions 4.2.7 Let $\mathfrak{X} = \{\chi_i\}_{i\in\mathbb{N}}$ be a simple sequence. For a set of constants $\mathcal{A}$, the **infinite $\mathfrak{X}$-word set** corresponding to the infinite word $u = u_0u_1\cdots \in \mathcal{A}^\omega$ is:

$$\Phi^\omega_{\mathfrak{X}}(u) = \{\chi_i = u_i \mid i \in \mathbb{N}\}.$$

The **set of infinite $\mathcal{X}$-word sets** from an alphabet $\mathcal{A}$ is then:

$$\hat{\mathcal{A}}^{\omega}_{\mathcal{X}} = \{\Phi^{\omega}_{\mathcal{X}}(u) \mid u \in \mathcal{A}^{\omega}\}.$$

Note how an infinite word has no end, so a blank symbol is not needed. We can describe a real number as an infinite $\mathcal{Y}$-word set as follows.

Example 4.2.8 Suppose we have a theory machine whose theory contains the real arithmetic axioms (Definition A.1.7). Let $-$, $\times$, and $\lfloor\cdot\rfloor$ be real functions that correspond to their usual meanings, and let 2 and c be constants, with 2 corresponding to its normal value $\mathbb{R}$.

Consider the terms:

$$\sigma(x) \equiv (2 \times x) - \lfloor(2 \times x)\rfloor, \text{ and } \gamma(x) \equiv \lfloor(2 \times x)\rfloor,$$

with free variable x. Using these we may construct the simple sequence:

$$\mathcal{Y} = \{\gamma(\sigma^n(c))\}_{n\in\mathbb{N}}.$$

Let $c \in [0,1]$ and $u \in \{0,1\}$. The infinite $\mathcal{Y}$-word set:

$$\Phi^{\omega}_{\mathcal{Y}}(u) = \{\gamma(\sigma^n(c)) = u_i \mid i \in \mathbb{N}\},$$

then corresponds to the statement that $c = 0.u_0u_1\ldots$ in binary.

To see why this is the case note that $\lfloor 2c\rfloor = 0$ if $c \in [0, \frac{1}{2})$ and $\lfloor 2c\rfloor = 1$ if $c \in [\frac{1}{2}, 1)$. Thus $\gamma(c) = u_0$ iff u_0 is the first digit of the binary expansion of c. Now if $c \in [0, \frac{1}{4}) \cup [\frac{1}{2}, \frac{3}{4})$ then $2c - \lfloor 2c\rfloor \in [0, \frac{1}{2})$, and if $c \in [\frac{1}{4}, \frac{1}{2}) \cup [\frac{3}{4}, 1)$ then $2c - \lfloor 2c\rfloor \in [\frac{1}{2}, 1)$. Hence $\gamma(\sigma(c)) = u_1$ iff u_1 is the second digit of the binary expansion of c. We can then see by induction that $\gamma(\sigma^n(c)) = u_i$ iff u_i is the ith digit of the binary expansion of c.

Lemma 4.2.9 *Let $\mathfrak{LS}$ be a logical system which contains first-order logic, let $\mathcal{X} = \{\chi_i\}_{i\in\mathbb{N}}$ be a simple sequence in a vocabulary $\mathcal{V}$ of $\mathfrak{LS}$, let $\mathcal{A} \subset \mathcal{V}$ be a set of constants, and let $a \in \{*, \omega\}$.*

For any $w, v \in \mathcal{A}^a$, if $w \neq v$ then: $\Phi^a_{\mathcal{X}}(w) \cup \Phi^a_{\mathcal{X}}(v) \cup EQ^{=}_{\mathcal{V}} \cup CD^{=}_{\mathcal{V}}$ is not $\mathfrak{LS}$-satisfiable.

Proof: If $a = *$ let $w = w_1 \cdots w_n$ and $v = v_1 \cdots v_m$. We then have $\Phi^*_{\mathcal{X}}(w) = \{\chi_i = w_i \mid i \in \{0, \ldots, n\}\} \cup \{\chi_{n+1} = \mathbf{b}\}$, and $\Phi^*_{\mathcal{X}}(v) = \{\chi_i = v_i \mid i \in \{0, \ldots, m\}\} \cup \{\chi_{m+1} = \mathbf{b}\}$. Since $w \neq v$ it must be the case that either $|w| \neq |v|$ or $w_j \neq v_j$ for some $j \in \mathbb{N}$.

If $|w| \neq |v|$ then without loss of generality let $|w| < |v|$, in which case we must have $\{\chi_{n+1} = \mathbf{b}\}$ and $\{\chi_{n+1} = v_{n+1}\}$ which together with $EQ^{=}_{\mathcal{V}} \cup CD^{=}_{\mathcal{V}}$ is a contradiction as $\mathbf{b} \notin \mathcal{A}$.

If $w_j \neq v_j$ for some $j \in \mathbb{N}$ then we have $\{\chi_j = w_j\}$ and $\{\chi_j = v_j\}$ which together with $EQ^{=}_{\mathcal{V}} \cup CD^{=}_{\mathcal{V}}$ must also be a contradiction.

If $a = \omega$ let $w = w_1 w_2 \cdots$ and $v = v_1 v_2 \cdots$. We then have $\Phi^{\omega}_{\mathcal{X}}(w) = \{\chi_i = w_i \mid i \in \mathbb{N}\}$, and $\Phi^{\omega}_{\mathcal{X}}(v) = \{\chi_i = v_i \mid i \in \mathbb{N}\}$. Since $w \neq v$ it must be the case that $w_j \neq v_j$ for some $j \in \mathbb{N}$.

We have $\{\chi_j = w_j\}$ and $\{\chi_j = v_j\}$ which together with $EQ^{=}_{\mathcal{V}} \cup CD^{=}_{\mathcal{V}}$ must also lead to a contradiction. ❏

4.2.3 Computing with Word Sets

We can now define what it means for a theory machine to compute a word problem.

Definition 4.2.10 Let $\mathcal{A}$ be a set of symbols and $A \subseteq \mathcal{A}^a$ for $a \in \{*, \omega\}$ be a decision problem. We say that an $\mathfrak{LS}_{\mathcal{V}}$-theory machine $\mathcal{M} = (\mathcal{T}, \mathcal{I}, \mathcal{O})$ in the vocabulary of $\mathcal{V}$ is **able to compute** A if there exists a set of constants $\mathcal{A} \subseteq \mathcal{V}$ and a simple sequence $\mathcal{X}$ such that $\hat{\mathcal{A}}^a_{\mathcal{X}} \subseteq \mathcal{I}$, and for two distinct finite output sets $\Theta, \Psi \in \mathcal{O}$ we have that for every $w \in \mathcal{A}^a$:

$$(w \in A \iff \mathcal{M}(\Phi^a_{\mathcal{X}}(w)) = \Theta) \text{ and } (w \notin A \iff \mathcal{M}(\Phi^a_{\mathcal{X}}(w)) = \Psi).$$

So a theory machine is able to compute a word problem if there exists a way in which we can configure each input word into the machine, such that the output of the function can clearly determined from the machine. Note that a problem can only be computed by a theory machine if every possible input word can be encoded into the machine, as we cannot just ignore troublesome inputs.

Definition 4.2.11 Let $\mathcal{A}$ and $\mathcal{B}$ be sets of symbols, and let $f :\subseteq \mathcal{A}^a \to \mathcal{B}^b$ for $a, b \in \{*, \omega\}$ be a word function problem.

We say that an $\mathfrak{LS}_\mathcal{V}$-theory machine $\mathcal{M} = (\mathcal{T}, \mathcal{I}, \mathcal{O})$ **is able to compute** f if there exist sets of constants $\mathcal{A}, \mathcal{B} \subseteq \mathcal{V}$ and simple sequences $\mathcal{X}, \mathcal{Y}$ such that $\hat{\mathcal{A}}^a_\mathcal{X} \subseteq \mathcal{I}$, $\hat{\mathcal{B}}^b_\mathcal{Y} \subseteq \mathcal{O}$, and for every $u \in \mathrm{dom}(f)$ we have:

$$\mathcal{M}(\Phi^a_\mathcal{X}(u)) = \Phi^b_\mathcal{Y}(f(u)).$$

So a theory machine is able to compute a word function problem if there exists a simple way for a user to configure each input word into the machine, such that the function's output can be simply read off from the machine.

As with a decision problem, a theory machine is only able to compute a function problem if every possible input and output word is contained within the input and the output sets. This is required, as we do not expect to know before hand exactly what words lie in f's domain and co-domain. Hence we cannot compute a partial function by just removing the undefined elements from the input or output sets, they must be undefined by the computation as well.

Also note the similarities between Definitions 4.2.10 and 4.2.11, with Definitions 2.2.8 and 2.1.5.

Definition 4.2.12 Let $\mathfrak{LS}$ be a logical system and $\mathcal{P}$ be either a word problem or a word function problem. We say that $\mathcal{P}$ is $\mathfrak{LS}$-*computable* if there exists an $\mathfrak{LS}$-theory machine that is able to compute $\mathcal{P}$.

Remark 4.2.13 Instead of using simple sequences as a basis for describing a theory machine's computation of a word problem, we could have just used computable mappings. That is, we could have just required that there existed Turing computable mapping from a set of input words to the set of input sets, and the set of output sets to a set of output words.

Such a requirement would have allowed for a much more general way of defining the inputs/outputs. However it would have also conflicted with our goal of devising a general overarching concept of computation. As our concept would have had to rely on another formulation of computation in order to make sense.

Another issue is that the computable mappings could themselves carry out a computation. Meaning that any form of computation defined in this way would, by default, be at least as powerful as a Turing machine.

Chapter 5

Examples of Theory Machines

Computational systems can take on many real world forms that are clearly not identical to one another. For example two different Turing machines may be built using completely different materials. However when we as observers look at these different realisations it is usually clear to us when they are exhibiting the same basic computational system. Similarly there may be multiple different inequivalent theory machines that characterise a given computational system, however it should be clear (or at least demonstrable) when such a characterisation occurs.

Definition 5.0.14 We say that a computational system $\mathcal{S}$ is *characterised* by a theory machine $\mathcal{M} = (\mathcal{T}, \mathcal{I}, \mathcal{O})$, if every input ι of $\mathcal{S}$ corresponds to an input Φ_ι in $\mathcal{I}$, in such a way that every model of $\mathcal{M}$ with input Φ_ι corresponds to the computation of $\mathcal{S}$ on input ι. Every output of $\mathcal{S}$ should also correspond to an output in $\mathcal{O}$, so that $\mathcal{M}$ is able to compute any problem that is computed by $\mathcal{S}$.

Correspondence is of course not quite a formal mathematical notion, however it should be clear (or at least be demonstrable) when a correspondence does occur. For example the input of a word w into a Turing machine may correspond to the finite word set $\Phi^*_{\mathcal{X}_TM}(w) = \{C(0, S^i(0)) = w_i \mid i \in \{0, \dots, n\}\} \cup \{C(0, S^{n+1}(0)) = \mathbf{b}\}$ in Example 4.2.5. Whereas the placement of an object at point $c = 0.u_0u_1 \ldots \in [0, 1)$ may correspond to the input set $\{\gamma(\sigma^n(c)) = u_i \mid i \in \mathbb{N}\}$ in Example 4.2.8.

In this chapter we will demonstrate how various well-known examples of computation can be characterised by theory machines.

5.1 Turing Machines

In the following example we will demonstrate how we can characterise a Turing machine which decides a word problem with a theory machine that uses second-order logic with equality (Definition 3.2.10).

Example 5.1.1 Let $M = (\Lambda, \Pi, \mathbf{b}, \mathcal{A}, s_0, \langle s_a, s_r\rangle, \mathcal{R})$ be a Turing machine[1] which computes the decision problem $A \subseteq \mathcal{A}^*$.

We can then characterise M by an $SO^{=}_{\mathcal{V}_M}$-theory machine[2]:

$$\mathcal{TM}_M = (\mathcal{TMT}_M, \hat{\mathcal{A}}^*_{\mathcal{X}_{TM}}, \{\{I(h) = s_a\}, \{I(h) = s_r\}\}),$$

with vocabulary:

$$\mathcal{V}_M = \{<, S, C, H, I, 0, h\} \cup \Lambda \cup \Pi.$$

Where $<$ is the usual binary ordering relation of $\mathbb{Z}$ and S is the unary successor function. C, H, I are functions such that for a cell y, the cell's content at time step x is given by $C(x, y)$, the cell pointed to by the head at time x is given by $H(x)$, and the machine's internal state at time x is given by $I(x)$.

The constant 0 represents both the centre of the tape and the starting time. Whereas the constant symbol h represents the halting time, which means that the value of h within the structure depends on when the machine reaches a halting state. Finally, Λ and Π are sets of constant symbols each corresponding to an element of Λ or Π.

To input $\mathcal{TM}_M$ uses the simple sequence:

$$\mathcal{X}_{TM} = \{C(0, S^n(0))\}_{n\in\mathbb{N}},$$

[1]As in Definition 2.1.1 Λ is M's tape alphabet, Π is the set of internal states, $\mathbf{b}$ is the blank symbol, $\mathcal{A}$ is the input alphabet, s_0 is the initial state s_a is the accepting state, s_r is the rejecting state, and $\mathcal{R}$ is M's set of rules.

[2]A $SO^{=}$ stands for second-order logic with equality (Definition 3.2.10).

which as in Example 4.2.5 ensures that each input provides a full description of how the input word is written on the tape at time 0. The theory of $\mathcal{TM}_M$ is:

$$\mathcal{TMT}_M = ISA \cup CD^{=}_{\mathcal{V}_M} \cup IT_{s_0} \cup ET_{\mathcal{R}} \cup HT_{(s_a,s_r)},$$

where ISA is the set of integer successor axioms (Definition A.1.5), and $CD^{=}_{\mathcal{V}_M}$ is the set of distinct constant axioms for $\mathcal{V}_M$ (Definition 4.2.6). The set of sentences IT_{s_0} defines the initial configuration of the machine, the set $ET_{\mathcal{R}}$ describes the evolution of the machine, and the set $HT_{(s_a,s_r)}$ ensures that the machine halts when it reaches s_a or s_r.

As $ISA \subset \mathcal{TMT}_M$, by Proposition A.1.6, any model of $\mathcal{TMT}_M$ must be isomorphic to an expansion of the usual ordered structure of the integers $\langle \mathbb{Z}; <, S, 0\rangle$.

Explicitly, the initial configuration is given by:

$$IT_{s_0} = \left\{ \begin{array}{l} (H(0) = 0) \wedge (I(0) = s_0), \\ \forall y(((C(0,y) = \mathbf{b}) \wedge (0 < y) \to (C(0,S(y)) = \mathbf{b})), \\ \forall y((y < 0) \to (C(0,y) = \mathbf{b})) \end{array} \right\}.$$

So by IT_{s_0}, in any model $\mathfrak{A}$ of $\mathcal{TMT}_M \cup \Phi^*_{\mathcal{X}_{TM}}(w)$ at time 0 the head points to cell 0 and the internal state is s_0. The input:

$$\Phi^*_{\mathcal{X}_{TM}}(w) = \{C(0, S^i(0)) = w_i \mid i \in \{0, \ldots, |w|-1\}\} \cup \{C(0, S^{|w|}(0)) = \mathbf{b}\},$$

specifies that at time 0 in $\mathfrak{A}$ the word $w \in \mathcal{A}^*$ is written on the tape starting from the 0th tape cell. $\Phi^*_{\mathcal{X}_{TM}}(w)$ also states that w is followed by a cell containing the blank symbol $\mathbf{b}$, so by the second sentence of IT_{s_0} and induction, the contents of every cell to the right of the input must be blank at time 0. Similarly by the last sentence of IT_{s_0}, every cell to the left of the input is blank. Hence the initial configuration of $\mathfrak{A}$ must be the same as it is for M with input w.

For evolving the configurations of the machine we have:

$$ET_{\mathcal{R}} = \left\{ \begin{array}{l} \forall x((0 < S(x)) \wedge \mu_{(t,b)}(x,x)) \to \\ \qquad\qquad (\mu_{(u,c)}(S(x),x) \wedge \pi_{(p)}(x) \wedge \nu(x))) \end{array} \;\middle|\; (t,b;u,c,p) \in \mathcal{R} \right\}.$$

After time step 0, each sentence of $ET_{\mathcal{R}}$ implements a rule of $\mathcal{R}$ via the following three sorts of terms. Firstly for each $s \in \Pi$, and $a \in \Lambda$ we have the term:

$$\mu_{(s,a)}(x_1,x_2) \equiv ((I(x_1) = s) \wedge (C(x_1, H(x_2)) = a)),$$

which indicates that at time x_1 the internal state is s, and the cell pointed to by the head at time x_2 contains an a at time x_1. Secondly for each $p \in \{LEFT, PAUSE, RIGHT\}$ we have the term:

$$\pi_{(p)}(x) \equiv \begin{cases} H(S(x)) = S(H(x)) & \text{if } p = \text{RIGHT}, \\ H(S(x)) = H(x) & \text{if } p = \text{PAUSE}, \\ S(H(S(x))) = H(x) & \text{if } p = \text{LEFT}, \end{cases}$$

that states that at the time step after x, if $p =$ RIGHT then the head is shifted to the succeeding tape position, if $p =$ LEFT then the head is shifted to the preceding tape position, and if $p =$ PAUSE then the head stays where it is. Finally we have:

$$\nu(x) \equiv \forall y(\neg(H(x) = y)) \to (C(x, y) = C(S(x), y))),$$

which ensures that the tape contents of any cell that is not being pointed to by a tape head, is preserved moving from time x to time $S(x)$.

Hence for each $(t, b; u, c, p) \in \mathfrak{R}$ the sentence $\forall x((0 < S(x)) \wedge \mu_{(t,b)}(x, x)) \to (\mu_{(u,c)}(S(x), x) \wedge \pi_{(p)}(x) \wedge \nu(x)))$ states that at any time $x \geqslant 0$[1] if the internal state is s and the head is pointing to a cell containing a b, then at time $x + 1$ the internal state becomes u, the symbol b is replaced with c, and the head position is moved p. Also, the contents of every cell not pointed to by the head at time x is the same at time $x + 1$ as it is at time x. Therefore this sentence clearly implements the rule $(t, b; u, c, p)$. So since $ET_{\mathfrak{R}}$ contains a sentence that implements each rule of $\mathfrak{R}$, if the configuration of $\mathfrak{A}$ at time x is the same as M at time x then the configurations of $\mathfrak{A}$ and M are the same at time $x + 1$.

For halting we have the set:

$$HT_{(s_a, s_r)} = \left\{ \begin{array}{c} \forall x((I(x) = s_a) \to (h = x)), \\ \forall x((I(x) = s_r) \to (h = x)) \end{array} \right\}.$$

So by $HT_{(s_a,s_r)}$ if at time x the machine is in the internal state s_a or the internal state s_r then $x = h$, the halting time step of $\mathfrak{A}$. The output, which is either $\{I(h) = s_a\}$ or $\{I(h) = s_r\}$ is therefore defined at this time.

[1] $0 < S(x)$ means that $0 < (x + 1)$, which means that $x \geqslant 0$.

So the configuration of $\mathfrak{A}$ at time 0, along with the configuration evolution of $\mathfrak{A}$ is same as it is for the Turing machine M with input w. Therefore by induction, for any $t \in \mathbb{N}$ the configuration of $\mathfrak{A}$ at time t is the same as it is for M at time t. Which means that $\mathfrak{A}$ reaches s_a or s_r and therefore halts at the same time that M does.

Clearly the outputs $\{I(h) = s_a\}$ and $\{I(h) = s_r\}$ are unique and completely dependent on the configurations of the machine prior to time h. The outputs therefore cannot be affected by whatever occurs afterwards in $\mathfrak{A}$ without making $\mathfrak{A}$ logically inconsistent. Thus we have that, $\mathfrak{A} \models_{SO^{=}_{\mathcal{V}_M}} (I(h) = s_a)$ if and only if M accepts w, and $\mathfrak{A} \models_{SO^{=}_{\mathcal{V}_M}} (I(h) = s_r)$ if and only if M rejects w.

The above description of a Turing machine machine may seem to be worryingly complicated, leading one to question the utility of the theory machine formalism. However it is worth noting that the Turing machine formalism is itself fairly complicated to explain to someone unfamiliar with it (the original definition given by Alan Turing took over two pages to explain [67]). The definition also relies on concepts such as the integers and a discretely order-able notion of time. These concepts may seem to be intuitively obvious but if we are going to be rigorous then they must be defined in order to be enforced. Indeed, paying such careful attention to the definition of a computational system may reveal inherent contradictions about one's assumptions.

Regardless, the usefulness of the theory machine formalism will be retroactively justified by our results in future chapters.

We shall now demonstrate how we can use a theory machine to characterise a Turing machine that computes a word function problem.

Example 5.1.2 Let $M' = (\Lambda', \Pi', \mathbf{b}, \mathcal{A}, s'_0, \langle s_1 \rangle, \mathcal{R}')$ be a Turing machine[1] which computes the function $f :\subseteq \mathcal{A}^* \to \mathcal{B}^*$.

In a similar manner to Example 5.1.1 characterise M' with an $SO^{=}_{\mathcal{V}_{M'}}$-theory

[1]By Definition 2.1.1 since M' has only one halting state, M' carries out the computation of a function.

machine:

$$\mathcal{TM}_{M'} = (\mathcal{TMT}_{M'}, \hat{\mathcal{A}}^*_{\mathcal{X}_{TM}}, \hat{\mathcal{B}}^*_{\mathcal{Y}_{TM}}),$$

with vocabulary:

$$\mathcal{V}_{M'} = \{<, S, C, H, I, 0, h\} \cup \Lambda' \cup \Pi'.$$

Where each element of $\{<, S, C, H, I, 0, h\}$ is as it is in $\mathcal{V}_M$ in Example 5.1.1, and each of the symbols in $\Lambda' \cup \Pi'$ is a constant.

To input and output the theory machine $\mathcal{TM}_{M'}$ uses the simple sequences:

$$\mathcal{X}_{TM} = \{C(0, S^n(0))\}_{n\in\mathbb{N}} \text{ and } \mathcal{Y}_{TM} = \{C(h, S^n(H(h)))\}_{n\in\mathbb{N}}.$$

So as in Example 5.1.1 $\mathcal{X}_{TM}$ describes how the input word is written to the right of cell 0 at time 0. Whereas $\mathcal{Y}_{TM}$ describes the output word and how it is written to the right of cell pointed to by the tape head at time the halting time h.

The theory of $\mathcal{TM}_{M'}$ is:

$$\mathcal{TMT}_{M'} = ISA \cup CD^{=}_{\mathcal{V}_{M'}} \cup IT_{s'_0} \cup ET_{\mathcal{R}'} \cup HT_{s_1},$$

where $ISA, IT_{s'_0}, CD^{=}_{\mathcal{V}_{M'}}, ET_{\mathcal{R}'}$ are as they are in Example 5.1.1, with s'_0 and $\mathcal{R}'$ replacing s_0 and $\mathcal{R}$. Whereas HT_{s_1} ensures that the machine halts only when its internal state is s_1. Explicitly this is achieved by:

$$HT_{s_1} = \{\forall x((I(x) = s_1) \to (h = x))\}.$$

So if at time x the internal state reaches s_1 then x is at the halting time h of any model $\mathfrak{A}'$ of $\mathcal{TMT}_{M'} \cup \Phi^*_{\mathcal{X}_{TM}}(w)$. The output:

$$\Phi^*_{\mathcal{Y}_{TM}}(v) = \{C(h, S^i(H(h))) = v_i \mid i \in \{0, \ldots, |v| - 1\}\} \cup \{C(h, S^{|v|}(H(h))) = \mathbf{b}\},$$

is then defined at this time.

Hence by the same reasoning as in Example 5.1.1, the configurations of $\mathfrak{A}'$ evolve from time 0 exactly as they do in the Turing machine M' with input w, and likewise the model outputs at the time step when the state s_1 is reached. By definition when M' halts the word $f(w)\mathbf{b}$ is written on the tape going rightwards from the tape head. Hence we must have $\mathfrak{A}' \models_{SO^{=}_{\mathcal{V}_{M'}}} \Phi^*_{\mathcal{Y}_{TM}}(f(w))$, and this output is unique since by definition $\mathbf{b} \notin \mathcal{B}$.

Therefore for any input $w \in \text{dom}(f)$:

$$\mathcal{TM}_{M'}(\Phi^*_{\mathcal{X}_{TM}}(w)) = \Phi^*_{\mathcal{Y}_{TM}}(f(w)).$$

Conversely, if $w \notin \text{dom}(f)$ then M' on input w must never halt, hence $I(p) \neq s_1$ for any $p \in \text{dom}(\mathfrak{A}')$. Which means that h could take any value in $\mathfrak{A}'$, including negative numbers, where $C(x, y)$ could take any value. Consequently any output $\Phi^*_{\mathcal{Y}_{TM}}(v) \in \hat{\mathcal{B}}^*_{\mathcal{Y}_{TM}}$ can be true in some model of $\mathcal{TMT}_{M'} \cup \Phi^*_{\mathcal{X}_{TM}}(w)$, and therefore $\mathcal{TM}_{M'}(\Phi^*_{\mathcal{X}_{TM}}(w))$ is undefined.

Theorem 5.1.3 *Any word problem $A \subseteq \mathcal{A}^*$ or function problem $f :\subseteq \mathcal{A}^* \to \mathcal{B}^*$ that is computable by a Turing machine can be computed by a theory machine.*

Proof: Let M be a Turing machine that computes $A \subseteq \mathcal{A}^*$. The theory machine $\mathcal{TM}_M$ in Example 5.1.1 is then able to compute A.

To see that $\mathcal{TM}_M$ is indeed a theory machine, note firstly that the theory of $\mathcal{TM}_M$ is clearly a set of $SO^{=}_{\mathcal{V}_M}$-sentences, and the inputs and outputs are clearly sets of sets of $SO^{=}_{\mathcal{V}_M}$-sentences. Secondly, since $CD^{=}_{\mathcal{V}_M} \subset \mathcal{TMT}_M$ the outputs $\{I(h) = s_a\}$ and $\{I(h) = s_r\}$ are clearly not mutually satisfiable regardless of the input.

Thirdly, for any $\Phi^*_{\mathcal{X}_{TM}}(w) \in \hat{\mathcal{A}}^*_{\mathcal{X}_{TM}}$ the set $\mathcal{TMT}_M \cup \Phi^*_{\mathcal{X}_{TM}}(w)$ is satisfied by a model such as $\mathfrak{A}$ above in which between time 0 and the halting time the configurations of $\mathfrak{A}$ are identical to M with input w. However $\mathfrak{A}$ is an expansion of $\mathbb{Z}$, so times prior to 0 or after h exist and the configurations of $\mathfrak{A}$ must also be defined at such times. Though these configuration are, for the most part, independent by $\mathcal{TMT}_M \cup \Phi^*_{\mathcal{X}_{TM}}(w)$. So for example, at any time x where $x < 0$ or $x > h$ we can have $\mathfrak{A}$ be such that $C(x, y) = \mathbf{b}$ for every cell y, $H(x) = 0$ and $I(x) = q$, where $q \notin \Pi$. Clearly no rule of $ET_{\mathfrak{R}}$ can be applied at such times so there is no need for any evolution of these configurations, and since $I(x) \neq s_a$ or s_r we do not have $h = x_1$ and $h = x_2$ for $x_1 \neq x_2$, which would give a logically inconsistent model.

To see that $\mathcal{TM}_M$ is able to compute A observe how $\hat{\mathcal{A}}^*_{\mathcal{X}_{TM}}$ is the set of inputs of $\mathcal{TM}_M$, and for any $w \in \mathcal{A}^*$ by our reasoning in in Example 5.1.1 we have:

$$w \in A \iff M \text{ accepts } w \iff \mathcal{M}(\Phi^*_{\mathcal{X}_{TM}}(w)) = \{I(h) = s_a\},$$

and:

$$w \notin A \iff M \text{ rejects } w \iff \mathcal{M}(\Phi^*_{\mathcal{X}_{TM}}(w)) = \{I(h) = s_r\}.$$

Similarly, let M' be a Turing machine that computes $f :\subseteq \mathcal{A}^* \to \mathcal{B}^*$. The theory machine $\mathcal{TM}_{M'}$ in Example 5.1.2 is then able to compute f. By the same reasoning as above, $\mathcal{TM}_{M'}$ is a theory machine, the only difference being the output set, and by Lemma 4.2.9 since $CD^{=}_{\mathcal{V}_M} \subset \mathcal{TMT}_M$ no two elements of $\hat{\mathcal{B}}^*_{\mathcal{Y}_{TM}}$ are mutually satisfiable.

Finally, to see that $\mathcal{TM}_{M'}$ is able to compute f note how $\hat{\mathcal{A}}^*_{\mathcal{X}_{TM}}$ and $\hat{\mathcal{B}}^*_{\mathcal{Y}_{TM}}$ are the input and output sets of $\mathcal{TM}_{M'}$, and by our reasoning in in Example 5.1.2 for any $w \in \mathrm{dom}(f)$ we have:

$$\mathcal{TM}_{M'}(\Phi^*_{\mathcal{X}_{TM}}(w)) = \Phi^*_{\mathcal{Y}_{TM}}(f(w)).$$

❑

5.2 Physical Systems

We may also utilise various analogue physical systems to carry out the computation of a problem. For example, a point light source placed in front of a pair of slits may be used to factorise integers [17] (see Subsection 5.2.2 below). However more typically a physical system A is used to figure out how a larger physical system B behaves. For example, one might want to calculate the flow of air around the wings of an aircraft before building it, to do this one could build a small scale model of the aircraft and place it in a wind tunnel with some sensors attached.

Remark 5.2.1 Suppose we have a theory machine $\mathcal{M} = (\mathcal{T}, \mathcal{I}, \mathcal{O})$ that characterises a physical system $\mathcal{S}$, with $\mathcal{I}$ corresponding to a collection of possible input states, and

$\mathcal{O}$ to a collection of relevant measurable properties. $\mathcal{S}$ may be too large/complicated to actually construct, and computationally infeasible to simulate with a classical computer. However, $\mathcal{M}$ may have many possible models, and if for any input $\Phi \in \mathcal{I}$ there exists a much smaller/simpler model of $\mathcal{M}$, then we know that we can construct this model of $\mathcal{T} \cup \Phi$ and use it to figure out the what the properties of $\mathcal{S}$ are on input Φ.

It is also possible that some aspects of $\mathcal{M}$ can be adjusted whilst still leading to analogous outcomes. That is, we may transform $\mathcal{M}$ into a theory machine $\mathcal{M}' = (\mathcal{T}', \mathcal{I}', \mathcal{O}')$ in such a way that for any $\Phi \in \mathcal{I}$ there exists a $\Phi' \in \mathcal{I}'$ such that $\mathcal{M}'(\Phi') = \Theta'$ if and only if $\mathcal{M}(\Phi) = \Theta$. If we can show this is the case and $\mathcal{M}'$ has easily constructible models then we may use $\mathcal{M}'$ to discern the properties of $\mathcal{S}$.

The above demonstrates that we can use theory machines to carry out the concept of model-based computation [7].

5.2.1 Physical Systems Satisfying Differential Equations

Various different physical systems (e.g. fluid-mechanical systems [55] and electromagnetic systems [33, 49]) are defined via a collection of smooth functions with domain $\mathbb{R}^3 \times \mathbb{R}$. That is, the domain of each function consists of the usual 3-dimensions of space, together with 1-dimensional time. The evolution of these functions typically depends on a set of differential equations (e.g. the Navier-Stokes equations [55] or the electromagnetic wave equations [33, 49]). Given an initial state and some boundary conditions, this set of differential equations will often define every future state of the system. MONIAC [14] (Figure 1) is an example of a computation system which can be defined in such a manner.

We can characterise this sort of physical system $\mathcal{S}$, which computes from an initial time of τ_0 to an end time of τ_1, by a theory machine $\mathcal{M}_\mathcal{S} = (\mathcal{T}_\mathcal{S}, \mathcal{I}_\mathcal{S}, \mathcal{O}_\mathcal{S})$ that uses first-order real logic (Definition 3.2.7). First-order real logic ($FO\mathbb{R}$) contains $\{<, +, \times, 0, 1\}$ as pre-defined symbols, and every $FO\mathbb{R}$-model is an expansion of usual structure of real arithmetic $\langle \mathbb{R}; <, +, \times, 0, 1\rangle$. So the domain of any model of

$\mathcal{M}_\mathcal{S}$ is $\mathbb{R}$, which means that any quaternary function f of $\mathcal{M}_\mathcal{S}$ is a function from $\mathbb{R}^4$ to $\mathbb{R}$. To describe a vector-valued function $\mathbf{g} : \mathbb{R}^4 \to \mathbb{R}^3$ in $\mathcal{M}_\mathcal{S}$ we can split it up into its 3 component functions $g_1, g_2, g_3 : \mathbb{R}^4 \to \mathbb{R}$.

For each dimension i of $\mathbb{R}^4$ we can define using first-order real logic a quaternary function $\partial_i f$ to be equal to the partial derivative of f in the ith dimension. As typically the partial derivative of a function f in the 1st dimension is defined to be:

$$\partial_1 f(x, y, z, t) = \lim_{\delta \to 0} \frac{f(x + \delta, y, z, t) - f(x, y, z, t)}{\delta}.$$

In a first-order real logic such a statement can be expressed by:

$$\begin{aligned} &\forall x \forall y \forall z \forall \epsilon \exists \delta (((0 < \epsilon) \to \\ &\quad ((0 < \delta) \wedge (|(((f(x + \delta, y, z, t) - f(x, y, z, t))/\delta) - \partial_1 f(x, y, z, t))| \leqslant \epsilon))). \end{aligned} \tag{5.1}$$

Where "/" is an additional binary function symbol defined by adding the sentence $\forall x \forall y \forall z (x = (y \times z)) \to ((x/y) = z)$ to $\mathcal{T}_\mathcal{S}$[1].

The partial derivative of f in the other dimensions of $\mathbb{R}^4$ can be defined similarly, and from the $\partial_i f$ functions we can define the second partial derivatives of f.

A differential equation of the form $\frac{\partial f}{\partial x} = f + 1$ can then be implemented within $\mathcal{M}_\mathcal{S}$ by including in $\mathcal{T}_\mathcal{S}$ a sentence of the form:

$$\forall x \forall y \forall z \forall t (\partial_1 f(x, y, z, t) = (f(x, y, z, t) + 1)).$$

Similarly, a boundary condition of the form $\frac{\partial f}{\partial z} = 0$ on the xy-plane where $x = 0 = y$ is implemented by adding to $\mathcal{T}_\mathcal{S}$ the sentence:

$$\forall z \forall t (\partial_3 f(0, 0, z, t) = 0).$$

The theory of $\mathcal{M}_\mathcal{S}$ can then be of the form:

$$\mathcal{T}_\mathcal{S} \approx \text{ Derivative definitions } + \text{ Differential equations } + \text{ Boundary conditions.}$$

[1]Note that this sentence does not define the value of x/y for $y = 0$. Typically division by 0 is of course taken to be undefined, however in any model of $\mathcal{T}_\mathcal{S}$ it will have a value as every function in an $FO\mathbb{R}$-structure is total. As it's undefined, this value could be any element in $\mathbb{R}$. However, if we define $\mathcal{T}_\mathcal{S}$ appropriately then this value will affect neither the rest of the structure, nor the output of any computation of $\mathcal{M}_\mathcal{S}$.

Meanwhile the input and output sets of $\mathcal{M}_{\mathcal{S}}$ can be of the form:

$$\mathcal{I}_{\mathcal{S}} \approx \text{ Initial states, and } \mathcal{O}_{\mathcal{S}} \approx \text{ End states/Measurements.}$$

The initial state of $\mathcal{M}_{\mathcal{S}}$ can be defined in various ways, we could for example define the f at time τ_0 to be equal some polynomial function, say:

$$f(x, y, z, \tau_0) = axyz + bx^2y + cxy^3z^5.$$

In which case we would write the above as a sentence of $\mathcal{T}_{\mathcal{S}}$ with constants a, b, c, these constants could then be specified within the input in similar manner to how we defined a real number in Example 4.2.8.

If $\mathcal{O}_{\mathcal{S}}$ is a set of end states then its elements could take a similar form to an input state. For example, if we knew at the end time τ_1 that f was of the form:

$$f(x, y, z, \tau_1) = px^2yz^2 + y.$$

then an output could be a description of the value of the constant p. However, in many real word cases we are unlikely to already know the form of the end state, and even if we did, its not clear how we would learn its coefficients. Instead, an output of such a system is more likely to take the form of measurement, which can be described in terms of a threshold at some point in the domain for example we could have the output set:

$$\mathcal{O}_{\mathcal{S}} = \{\{(f(1,1,0,\tau_1) < 1)\}, \{(1 \leqslant f(1,1,0,\tau_1))\}\}.$$

Which describes whether or not the value of f is less than 1 at point $(1, 1, 0)$ at time τ_1.

Remark 5.2.2 More generally if $f : D \times \mathbb{R} \to \mathbb{R}$ is a smooth function and $D \subseteq \mathbb{R}^3$ is convex, then the state of f at time $\tau_q \in \mathbb{R}$ may be described via a single infinite word set. This is due to Taylor's Theorem in n-variables [40], which states that the function f at time τ_q is equal to a 3-dimensional Taylor series of the form:

$$f(x_1, x_2, x_3, \tau_q) = \sum_{m=0}^{\infty} \frac{1}{m!} \sum_{i_1,\ldots,i_m=1}^{3} \frac{\partial^m f}{\partial y_{i_1} \cdots \partial y_{i_m}}(\mathbf{a}) \prod_{j=1}^{m} (x_{i_j} - a_{i_j}).$$

Where $\mathbf{a}$ is some element of $D \times \{\tau_q\}$, the collection of values of $\frac{\partial^m f}{\partial y_{i_1} \cdots \partial y_{i_m}}(\mathbf{a})$ for $i_1, \ldots, i_m \in \{1,2,3\}$ and $m \in \mathbb{N}$ then defines f everywhere in $D \times \{\tau_q\}$. This is a countable set of real numbers, which means that it can be defined with a single infinite word, and can therefore be described via a single infinite word set.

Further, if D is not convex but is compact, then we are able to cover D by a finite set of convex balls $\{U_i\}_{i=1}^{K}$. Hence we can describe f at time τ_q within each ball U_i via a 3-dimensional Taylor series defined using the partial derivatives of $f(\mathbf{c})$ for some $\mathbf{c} \in U_i$. The state of f at time τ_q can therefore be entirely defined via an infinite word set which describes each of these Taylor expansions.

5.2.2 Blakey's Double Slit Factoriser

A specific example of a physical computational system that can be described via a collection of functions that satisfies a set of differential equations is Blakey's double slit factoriser. Blakey's double slit factoriser was described by Blakey in [17], notably the device is capable of factorising integers in polynomially bounded space and time whilst still being describable with classical physics.

Blakey's factoriser (Figure 5.2.2) consists of a screen with a pair of slits of distance 1 apart at points $A = (0,0)$ and $B = (1,0)$ with a light source placed in front of the two slits at $C = (\frac{1}{2}, \frac{-1}{2})$. A detector runs perpendicular to the screen from A which detects where sufficiently strong instances of radiation hit the screen.

To factorise the integer $n \in \mathbb{N}$ one makes the light source emit radiation of wavelength $\frac{1}{2\sqrt{n}}$ from C. The two slits then diffract the light wave, causing an interference pattern on the detector. Blakey showed in [17] that if maximal constructive interference is detected at a distance h from the screen then $\sqrt{n}(\sqrt{h^2+1}+h)$ must be a factor of n.

We can describe the propagation of the light wave via an electromagnetic wave function $\mathbf{E} : \mathbb{R}^4 \to \mathbb{R}^3$ that satisfies the electromagnetic wave equations for a point charge [33, 49] at $(0.5, \text{-}0.5, 0)$ with magnitude m, which is a differential equation of

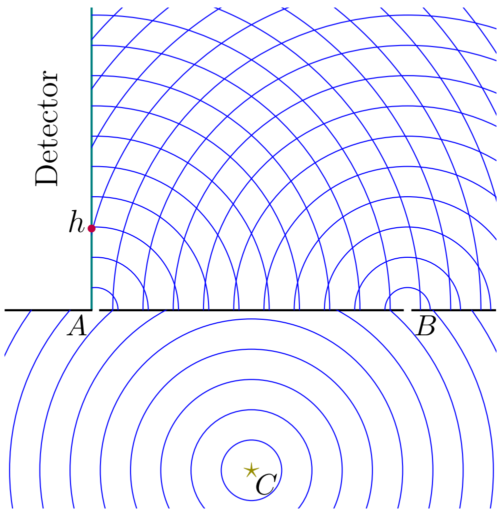


Figure 5.1: A diagram of the wave propagation in Blakey's double-slit factoriser.

the form:

$$\left(v_{ph}^2\nabla^2 - \frac{\partial^2}{\partial t^2}\right)\mathbf{E} = \text{-}m\delta(\vec{x} - (0.5, \text{-}0.5, 0)), \tag{5.2}$$

where $\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}$ is the Laplacian, v_{ph} is the speed of light in the medium of the wave, and δ is the Dirac delta function[1].

Hence we can characterise Blakey's factoriser with an $FO\mathbb{R}$-theory machine whose theory contains the electromagnetic wave equations for a point charge. The screen and slits can be implemented as boundary conditions. Whereas the wavelength of the light source may be defined using the an integer input $n \in \mathbb{N}$, which itself can be defined as binary word. The point of maximal constructive interference h may be similarly defined as the lowest point on the y-axis such that $|\mathbf{E}| = 2\alpha$, where α is the amplitude of the wave. From h, the factor $\sqrt{n}(\sqrt{h^2+1}+h)$ can be outputted by the machine as another binary word.

[1]So $\delta(\vec{y}) = \begin{cases} 1 & \text{if } \vec{y} = (0,0,0), \\ 0 & \text{otherwise.} \end{cases}$ for any $\vec{y} \in \mathbb{R}^3$ [49].

5.3 Extensions of the Turing Machine Model

Turing machines take finite words as inputs and give finite words as outputs. However, as indicated by Definitions 4.2.7 we are able to input infinite words into a theory machine, and a theory machine is able to compute infinite output words. An extension of the Turing machine model that admits infinite inputs and outputs is the type-2 machine (See Section 2.3 for details). Type-2 machines are themselves an extension of the multi-tape Turing machine model (Definition 2.1.8), and in this section we will demonstrate how both of these models of computation can be characterised by theory machines.

5.3.1 Multi-tape Turing machines

If both the input and output sets of a type-2 machine are of type $*$ then it behaves identically to a multi-tape Turing machine. Indeed to highlight their similarities we shall firstly characterise a multi-tape Turing machine before demonstrating how its description can be simply converted into a type-2 machine.

Example 5.3.1 Let $M_m = (\Lambda, \Pi, \mathbf{b}, \mathcal{A}, s_0, \langle s_1 \rangle, \mathcal{P})$ be a multi-tape Turing machine[1] which computes the function $g :\subseteq \mathcal{A}^* \to \mathcal{B}^*$.

We can then characterise M_m by an $SO^{=}_{\mathcal{V}_{M_m}}$-theory machine:

$$\mathcal{TM}_{M_m} = (\mathcal{TMT}_{M_m}, \hat{\mathcal{A}}^*_{\mathcal{X}_{TM_1}}, \hat{\mathcal{B}}^*_{\mathcal{Y}_{TM_m}}),$$

with vocabulary:

$$\mathcal{V}_{M_m} = \{<, S, I, 0, h\} \cup \{C_1, \ldots, C_m\} \cup \{H_1, \ldots, H_m\} \cup \Lambda \cup \Pi.$$

Where $<, S, I, 0, h$ are as in Example 5.1.1, and $\Lambda \cup \Pi$ is a set of constant symbols. Whereas $C_1, \ldots, C_m$ are binary functions and $H_1, \ldots, H_m$ are unary functions such that $C_n(x, y)$ gives the symbol in cell y, on tape $n \in \{1, \ldots, m\}$ at time x, and the cell pointed to by the nth tape head at time x is given by $H_n(x)$.

[1]So as in Definition 2.1.8 $\Lambda, \Pi, \mathbf{b}, \mathcal{A}, s_0$, and $\langle s_1 \rangle$ are as they are for a Turing machine and each rule of $\mathcal{P}$ is an element of $\Pi \setminus \{s_1\}) \times \Lambda^m \times \Pi \times \Lambda^m \times \{LEFT, PAUSE, RIGHT\}^m$.

To input and output $\mathcal{TM}_{M_m}$ uses the simple sequences:

$$\mathcal{X}_{TM_1} = \{C_1(0, S^m(0))\}_{n\in\mathbb{N}} \text{ and } \mathcal{Y}_{TM_m} = \{C_m(h, S^n(0))\}_{n\in\mathbb{N}}.$$

The theory of $\mathcal{TM}_{M_m}$ is then:

$$\mathcal{TMT}_{M_m} = ISA \cup CD^{=}_{\mathcal{V}_{M_m}} \cup IT_{(s_0,m)} \cup ET_{\mathcal{P}} \cup HT_{s_1},$$

where exactly as in Example 5.1.2 ISA is the set of integer successor axioms (Definition A.1.5), $CD^{=}_{\mathcal{V}_{M_m}}$ is the set of distinct constant axioms for $\mathcal{V}_{M_m}$ (Definition 4.2.6), and HT_{s_1}[1] ensures that the machine halts when it reaches s_1. Whereas $IT_{(s_0,m)}$ defines the initial configuration of every tape, and each sentence of $ET_{\mathcal{P}}$ describes a multi-tape rule of the machine.

As before, since the theory contains ISA, any model of $\mathcal{TMT}_{M_n}$ must be isomorphic to an expansion of the usual ordered structure of the integers. Explicitly, the initial configuration of the multi-tape Turing machine is given by:

$$IT_{(s_0,m)} = \left\{ \begin{array}{l} \bigwedge_{n=1}^{m}(H_n(0) = 0) \wedge (I(0) = s_0), \\ \forall y(((C_1(0, y) = \mathbf{b}) \wedge (0 < y)) \to (C_1(0, S(y)) = \mathbf{b})), \\ \forall y((y < 0) \to (C_1(0, y) = \mathbf{b})), \\ \forall y \bigwedge_{n=2}^{m}(C_n(0, y) = \mathbf{b}) \end{array} \right\}.$$

This points every tape head to cell 0 at time 0, and as in Example 5.1.2 ensures that every tape cell on tape 1 not defined by the input is blank at time 0. The last sentence of $IT_{(s_0,m)}$ then implies that every tape cell not on tape 1 is also blank at time 0. For applying the rules of M_n we have:

$$ET_{\mathcal{P}} = \left\{ \begin{array}{l} \forall x((0 < S(x)) \wedge \mu_{(t_l,\vec{b}_l)}(x,x)) \to \\ \qquad (\mu_{(\vec{u}_l,\vec{c}_l)}(S(x), x) \wedge \pi_{(\vec{p}_l)}(x) \wedge \nu_m(x))) \end{array} \middle| \; (t_l, \vec{b}_l; u_l, \vec{c}_l; \vec{p}_l) \in \mathcal{P} \right\}.$$

As in Example 5.1.1 each sentence of $ET_{\mathcal{P}}$ implements a rule of $\mathcal{P}$ via three sorts of terms, this time in a multi-tape form. So firstly for each $s \in \Pi$ and $\vec{a} \in \Lambda^m$ we have:

$$\mu_{(s,\vec{a})}(x_1, x_2) \equiv (I(x_1) = s) \wedge \bigwedge_{n=1}^{m}(C_n(x_1, H_n(x_2)) = a_n),$$

[1]Recall that $HT_{s_1} = \{\forall x((I(x) = s_1) \to (h = x))\}$.

which indicates that at time x_1 the internal state is s, and the cell pointed to by head n at time x_2 contains a_n at time x_1, for each $n \in \{1, \dots, m\}$. Secondly for $p \in \{LEFT, PAUSE, RIGHT\}$ and $n \in \{1, \dots, m\}$ let:

$$\pi_{(p,n)}(x) \equiv \begin{cases} H_n(S(x)) = S(H_n(x)) & \text{if } p = \text{RIGHT}, \\ H_n(S(x)) = H_n(x) & \text{if } p = \text{PAUSE}, \\ S(H_n(S(x))) = H_n(x) & \text{if } p = \text{LEFT}, \end{cases}$$

For $\vec{p} \in \{LEFT, PAUSE, RIGHT\}^m$ we then have:

$$\pi_{(\vec{p})}(x) \equiv \bigwedge_{n=1}^{m} \pi_{(p,n)}(x).$$

which states that at the time step after x the head on each tape is shifted in the manner indicated by p. Finally we have:

$$\nu_m(x) \equiv \forall y \bigwedge_{n=1}^{m} (\neg(H_n(x) = y) \to (C_n(x, y) = C_n(S(x), y))),$$

which ensures that the tape contents of any cell that is not being pointed to by a tape head is preserved moving from time x to time $S(x)$.

Hence by same reasoning as in Examples 5.1.1 and 5.1.2 the configurations of any model $\mathfrak{C}$ of $\mathcal{TTM}_{M_m} \cup \Phi^*_{\mathcal{X}_{TM_1}}(w)$ evolve from time 0 exactly as they do in the multi-tape Turing machine M_m with input $w \in \text{dom}(g)$. Like in Example 5.1.2 HT_{s_1} ensures that when $\mathfrak{C}$ reaches the state s_1 the machine halts, labelling the time step that this happens at by h. By definition of M_m tape m should then contain the word $g(w)\mathbf{b}$ written from tape square 0 at time h, and as the configurations of $\mathfrak{C}$ are identical to those of M_m we have that $\mathfrak{C}$ models the output:

$$\Phi^*_{\mathcal{Y}_{TM_m}}(g(w)) = \begin{array}{l} \{C_m(h, S^i(0)) = g(w)_i \mid i \in \{0, \dots, |g(w)| - 1\}\} \\ \cup \{C_m(h, S^{|g(w)|}(0)) = \mathbf{b}\}, \end{array}$$

Therefore for any $w \in \text{dom}(g)$:

$$\mathcal{TM}_{M_m}(\Phi^*_{\mathcal{X}_{TM_1}}(w)) = \Phi^*_{\mathcal{Y}_{TM_m}}(g(w)).$$

5.3.2 Type-2 Machines

We shall now demonstrate how we can characterise a type-2 machine with a theory machine.

Example 5.3.2 Let $T = (\Lambda, \Pi, \mathbf{b}, \mathcal{A}, s_0, s_1, \mathcal{U}, a, b)$ be a type-2 machine with m tapes[1] that computes the function $f :\subseteq \mathcal{A}^a \to \mathcal{B}^b$.

We can then characterise T by an $SO^{=}_{\overline{\mathcal{V}}_T}$-theory machine:

$$\mathcal{T}2\mathcal{M}_T = (\mathcal{T}2\mathcal{M}\mathcal{T}_T, \hat{\mathcal{A}}^a_{\mathcal{X}_{TM_1}}, \hat{\mathcal{B}}^b_{\mathcal{Y}_{T2}}),$$

with vocabulary:

$$\mathcal{V}_T = \{<, S, O, I, 0, h\} \cup \{C_1, \ldots, C_m\} \cup \{H_1, \ldots, H_m\} \cup \Lambda \cup \Pi.$$

where O is a unary function used in defining the output, and everything in $\mathcal{V}_T \setminus \{O\}$ is as it is for $\mathcal{V}_{M_m}$ in Example 5.3.1. The input sequence $\mathcal{X}_{TM_1}$ is as it is in Example 5.3.1, whereas the output sequence is:

$$\mathcal{Y}_{T2} = \{O(S^n(0))\}_{n\in\mathbb{N}}.$$

The theory of $\mathcal{T}2\mathcal{M}_T$ is:

$$\mathcal{T}2\mathcal{M}\mathcal{T}_T = ISA \cup IT_{(s_0,m)} \cup ET_{\mathcal{U}} \cup HT^b_{s_1}.$$

Where ISA, $CD^{=}_{\overline{\mathcal{V}}_T}$, $IT_{(s_0,m)}$, and $ET_{\mathcal{U}}$ are as they are in Example 5.3.1, with the constants of $\mathcal{V}_T$ replacing those of $\mathcal{V}_{M_m}$ and $\mathcal{P}$ replaced by $\mathcal{U}$. The set $HT^b_{s_1}$ deals with defining the output and halting, however T should only halt if $\mathcal{B}^b = \mathcal{B}^*$, otherwise if $\mathcal{B}^b = \mathcal{B}^\omega$ then T should never halt. Hence we have:

$$HT^b_{s_1} = \begin{cases} HT_{s_1} \cup \{\forall y(O(y) = C_m(h, y))\} & \text{if } b = *, \\ \{\forall x \forall y(\neg(C_m(x, y) = \mathbf{b}) \to (O(y) = C_m(x, y))\} & \text{if } b = \omega. \end{cases}$$

Where as in Example 5.3.1 $HT_{s_1} = \{\forall x((I(x) = s_1) \to (h = x))\}$, so if $b = *$ the additional sentence ensures that $O(y)$ gives the contents of the yth cell of tape m at time h. If T never halts then there is no point at which the whole output will be written on tape m, so in order to extract it we have $HT^\omega_{s_1}$ which defines $O(y)$ to be equal to whatever the contents of cell y of tape m are when it is not blank.

[1]As in Definition 2.3.1; Λ is the alphabet of tape symbols, Π is the set of internal states, $\mathbf{b} \in \Lambda$ is the blank tape symbol, $\mathcal{A} \subseteq (\Lambda \setminus \mathbf{b})$ is the alphabet of input symbols, $s_0 \in \Pi$ is the initial state, $s_1 \in \Pi$ is the halting state, $\mathcal{U}$ is a set of m-tape Turing machine rules, and $a, b \in \{*, \omega\}$ are the input and output types.

This value is well-defined as, by the definition of a type-2 machine, each square of the mth tape eventually has a non-blank symbol written on it, and afterwards this symbol remains fixed.

Let $\mathfrak{D}$ be a model of $\mathcal{T}2\mathcal{M}\mathcal{T}_T \cup \Phi^a_{\mathcal{X}_{TM_1}}(w)$ for some $w \in \mathrm{dom}(f)$. Clearly by the input and $IT_{(s_0,m)}$, for each $i \in \{1, \dots, m\}$ and $y \in \mathbb{N}$ the values of $C_i(0,y)$, $H_i(0)$, and $I(0)$ in $\mathfrak{D}$ correspond exactly to the initial configuration of T. As $\mathfrak{D} \models_{SO^{=}_{\mathcal{V}_T}} ET_{\mathcal{U}}$ the evolution of these values moving from $C_i(n,y)$ to $C_i(S(n),y)$ should similarly correspond exactly to the evolution of T. Therefore by induction for every $x \in \mathbb{N}$ the values of $C_i(x,y)$, $H_i(x)$, and $I(x)$ correspond exactly to the configurations of T at every time x.

Now there are two possible cases.

Case 1. $b = *$, in which case if $w \in \mathrm{dom}(f)$ then T on input w must eventually halt. By $HT^*_{s_1}$, at this time step in $\mathfrak{D}$ the values of $O(y)$ are defined for each $y \in \mathbb{N}$ to be equal to the contents of tape m. So as in Example 5.3.1 since tape m should contain the word $f(w)\mathbf{b}$ written from square 0, we have that:

$$\mathfrak{D} \models_{SO^{=}_{\mathcal{V}_T}} \{O(S^i(0)) = f(w)_i \mid i \in \{0, \dots, |f(w)| - 1\}\} \cup \{O(S^{|f(w)|}(0)) = \mathbf{b}\}.$$

Conversely if $w \notin \mathrm{dom}(f)$ then T on input w must never halt, in which case we have the same scenario as in Example 5.1.2 and any possible output in $\hat{\mathcal{B}}^b_{\mathcal{Y}_{T2}}$ could be true in a model of $\mathcal{T}2\mathcal{M}\mathcal{T}_T \cup \Phi^a_{\mathcal{X}_{TM_1}}(w)$. Therefore $\mathcal{T}2\mathcal{M}(\Phi^a_{\mathcal{X}_{TM_1}}(w))$ is undefined for such w.

Case 2. $b = \omega$, in which case if $w \in \mathrm{dom}(f)$ then T must on input w eventually write a symbol on each cell of tape m. The value of $C_m(x,y)$ in $\mathfrak{D}$ for each $y \in \mathbb{N}$ is therefore eventually defined to be $f(w)_y$. Hence by $HT^\omega_{s_1}$ and the reasoning above the value of $O(y)$ in $\mathfrak{D}$ is then defined to be equal to $f(w)_y$. Thus we have that:

$$\mathfrak{D} \models_{SO^{=}_{\mathcal{V}_T}} \{O(S^i(0)) = f(w)_i \mid i \in \mathbb{N}\}.$$

Conversely if $w \notin \mathrm{dom}(f)$ then T on input w must either halt at some point, or eventually cease to write symbols onto the mth tape. In either scenario there must exist an $N \in \mathbb{N}$ such that $C_m(x, S^n(0)) = \mathbf{b}$ for all $n \geqslant N$. Hence $O(S^n(0))$ is not specified by $\mathcal{T2M} \cup \Phi^a_{\mathcal{X}_{TM_1}}(w)$ and could take any value. Therefore $\mathcal{T2M}(\Phi^a_{\mathcal{X}_{TM_1}}(w))$ is undefined for such w.

In both of the above cases $\mathfrak{D}$ was an arbitrary model of $\mathcal{T2MT}_T \cup \Phi^a_{\mathcal{X}_{TM_1}}(w)$, hence we have that $\mathcal{T2M}_T(\Phi^a_{\mathcal{X}_{TM_1}}(w)) = \Phi^b_{\mathcal{Y}_{T2}}(f(w))$ for each $w \in \mathcal{A}^a$.

Notably, the output function O of $\mathcal{T2M}_T$ is unary. Only taking a cell position as an input, so O does not have any apparent time dependency. In a sense his means that O's value exists outside of time, whilst still depending on the temporally ordered computation. The atemporal nature of this output is therefore an example of how a theory machine is able to compute without conforming to the usual assertion that the entirety of a computational process must follow a causal temporal order.

Theorem 5.3.3 *Any word function problem $f :\subseteq \mathcal{A}^a \to \mathcal{B}^b$ for $a, b \in \{*, \omega\}$ that is computable by a type-2 machine can be computed by a theory machine.*

Proof: Let T be a type-2 machine that computes $f :\subseteq \mathcal{A}^a \to \mathcal{B}^b$, so for any $w \in \mathcal{A}^a$ we have $T(w) = f(w)$. The theory machine $\mathcal{T2M}_T$ in Example 5.3.2 is then able to compute f.

To see that $\mathcal{T2M}_T$ is indeed a theory machine we can follow the same reasoning as in the proof of Theorem 5.1.3, noting that a type-2 machine with input $w \in \mathcal{A}^a$ provides a model of $\mathcal{T2MT}_T \cup \Phi^a_{\mathcal{X}_{TM_1}}(w)$ provided that at negative time steps the internal state of the machine is not in Π. The fact that the outputs are mutually unsatisfiable also follows from Lemma 4.2.9.

To see that $\mathcal{T2M}_T$ is then able to compute f observe how $\hat{\mathcal{A}}^a_{\mathcal{X}_{TM_1}}$ and $\hat{\mathcal{B}}^b_{\mathcal{Y}_{T2}}$ are the input and output sets of $\mathcal{T2M}_T$, and by our reasoning in Example 5.3.2, for any $w \in \mathrm{dom}(f)$ we have:

$$\mathcal{T2M}_T(\Phi^a_{\mathcal{X}_{TM_1}}(w)) = \Phi^b_{\mathcal{Y}_{T2}}(f(w)).$$

❏

Chapter 6

Properties of Theory Machine Computation

We used to think that if we knew one, we knew two, because one and one are two. We are finding that we must learn a great deal more about 'and.' - Sir Arthur Eddington

In Chapter 5 we saw that various sorts of computation devices and frameworks can be characterised with theory machines. For finite word problems, the computational power of each of these examples has so far been less than or equal to that of a Turing machine[1]. So naturally one might be led to speculate that the computational power of *any* theory machine is it at most that of a Turing machine. However, the following result demonstrates that such an assertion is quite spectacularly false.

Proposition 6.0.4 *For any word problem $A \subseteq \mathcal{A}^*$ and any word function problem $f :\subseteq \mathcal{A}^* \to \mathcal{B}^*$ there exists an FO-theory machine that is able to compute A and an FO-theory machine that is able to compute f.*

Proof: Consider the $FO_{\mathcal{V}_A}$-theory machine:

$$\mathcal{M}_A = (\mathcal{T}_A, \hat{\mathcal{A}}^*_{\mathcal{X}_C}, \{\{P\}, \{\neg P\}\}),$$

[1]Note that by Remark 2.1.6 and Theorem 2.1.12 there exist finite word problems that are not computable by a Turing machine.

in vocabulary:

$$\mathcal{V}_A = \{=, S, C, P, 0, \mathbf{b}\} \cup \mathcal{A},$$

where $=$ is a binary relation, S, C are unary functions, P is a 0-ary relation, and $\{0, \mathbf{b}\} \cup \mathcal{A}$ are constant symbols. Let the input sequence be:

$$\mathcal{X}_C = \{C(S^n(0))\}_{n\in\mathbb{N}},$$

and let $\mathcal{M}_A$ have theory:

$$\begin{aligned}\mathcal{T}_A = EQ^{=}_{\mathcal{V}_A} \; &\cup \left\{ \left(\bigwedge_{n=0}^{|w|-1} (C(S^n(0)) = w_n) \wedge (C(S^{|w|}(0)) = \mathbf{b}) \right) \to P \;\middle|\; w \in A \right\} \\ &\cup \left\{ \left(\bigwedge_{n=0}^{|w|-1} (C(S^n(0)) = w_n) \wedge (C(S^{|w|}(0)) = \mathbf{b}) \right) \to \neg P \;\middle|\; w \notin A \right\}.\end{aligned}$$

Clearly then $\mathcal{T}_A \cup \Phi^*_{\mathcal{X}_C}(w) \models_{FO} P$ iff $w \in A$ and otherwise $\mathcal{T}_A \cup \Phi^*_{\mathcal{X}_C}(w) \models_{FO} \neg P$. Therefore $\mathcal{M}_A$ is able to compute A.

Similarly, consider the $FO_{\mathcal{V}_f}$-theory machine:

$$\mathcal{M}_f = (\mathcal{T}_f, \hat{\mathcal{A}}^*_{\mathcal{X}_C}, \hat{\mathcal{B}}^*_{\mathcal{X}_W}),$$

in vocabulary:

$$\mathcal{V}_f = \mathcal{V}_A \cup \{W\} \cup \mathcal{B},$$

where W is a unary function, and $\mathcal{B}$ is a set of constant symbols. Let the output sequence be:

$$\mathcal{X}_W = \{W(S^n(0))\}_{n\in\mathbb{N}},$$

and let $\mathcal{M}_f$ have theory:

$$\mathcal{T}_f = EQ^{=}_{\mathcal{V}_f} \cup \mathcal{T}'_f,$$

where:

$$\mathcal{T}'_f = \left\{ \begin{array}{l} \left(\bigwedge_{n=0}^{|w|-1} (C(S^n(0)) = w_n) \wedge (C(S^{|w|}(0)) = \mathbf{b}) \right) \\ \to \left(\bigwedge_{n=0}^{|f(w)|-1} (C(S^n(0)) = f(w)_n) \wedge (C(S^{|f(w)|}(0)) = \mathbf{b}) \right) \end{array} \;\middle|\; w \in \mathrm{dom}(f) \right\}.$$

Clearly then $\mathcal{T}_f \cup \Phi^*_{\mathcal{X}_C}(w) \models_{FO} \Phi^*_{\mathcal{X}_W}(f(w))$ for every $w \in \mathrm{dom}(f)$. Therefore $\mathcal{M}_f$ is able to compute f. ❏

So with first-order theory machines we are able to compute *any* finite word problem. Arguably, this result is quite problematic as not only do $\mathcal{M}_A$ and $\mathcal{M}_f$ above compute in a manner which provides no real insight into the problems that they are calculating. But it is also very difficult to see how real world computational devices that model $\mathcal{M}_A$ or $\mathcal{M}_f$ could ever be constructed for the majority of problems, even if we already knew A or f in its entirety.

Furthermore the issue here is not just not limited to first-order theory machines. In any logical system which contains first-order logic we can characterise exactly the same sort of device.

Crucially though the theories of $\mathcal{M}_A$ and $\mathcal{M}_f$ are both countably infinite in size, and it is this infinite amount of information that really enables these theory machines to compute arbitrary finite word problems. If we restricted ourselves to considering only theory machines with *finite* theories (consisting of only finite sentences) then questionable examples such as $\mathcal{M}_A$ and $\mathcal{M}_f$ would not be possible. Indeed each of the theory machine examples we gave in Chapter 5 had finite theories. Furthermore, a finite theory can be finitely described, which conforms with Horsman et al.'s assertion in [48] that we can only ever obtain a finite amount of information about a physical system that we are computing with.

Definition 6.0.5 Let $\mathfrak{LS}$ be a logical system. A *finite $\mathfrak{LS}$-theory machine* is an $\mathfrak{LS}$-theory machine $\mathcal{M} = (\mathcal{T}, \mathcal{I}, \mathcal{O})$ such that $\mathcal{T}$ is a finite set of sentences.

In Chapter 8 we will we will show that a finite word problem is computable by a finite first-order theory machine if and only if it is computable by a Turing machine (Theorem 8.2.1). However in this chapter we will consider consider computation with more general logical systems.

Definition 6.0.6 Let $\mathfrak{LS}$ be a logical system. We say that a problem P is *finite $\mathfrak{LS}$-computable* if there exists a finite $\mathfrak{LS}$-theory machine that is able to compute P.

Though the class of finite $\mathfrak{LS}$-theory machines is clearly a very general object, as we shall see, the collection of problems that machines within the class are able to compute satisfies several interesting properties.

Proposition 6.0.7 *Let $\mathfrak{LS}$ be a logical system. If a word problem $A \subseteq \mathcal{A}^a$ for $a \in \{*, \omega\}$ is computable by a finite $\mathfrak{LS}$-theory machine then the problem $\mathcal{A}^* \setminus A$ is computable by a finite $\mathfrak{LS}$-theory machine.*

Proof: Let $\mathcal{M}$ be a finite $\mathfrak{LS}$-theory machine such that $\mathcal{M}$ is able to compute A. There then must exist a simple sequence $\mathcal{X}$ such that $\hat{\mathcal{A}}^a_{\mathcal{X}} \subseteq \mathcal{I}$, and two distinct finite output sets $\Theta, \Psi \in \mathcal{O}$ such that for every $w \in \mathcal{A}^a$:

$$(w \in A \iff \mathcal{M}(\Phi^a_{\mathcal{X}}(w)) = \Theta) \text{ and } (w \notin A \iff \mathcal{M}(\Phi^a_{\mathcal{X}}(w)) = \Psi).$$

It is then the case that $\mathcal{M}$ is itself able to compute $\mathcal{A}^a \setminus A$ by simply swapping the interpretation Θ and Ψ, taking Ψ to indicate that $w \in \mathcal{A}^a \setminus A$ and Θ to indicate that $w \notin \mathcal{A}^a \setminus A$. ❑

The fact that the collection of decision problems that are computable by the class of finite $\mathfrak{LS}$-theory machines is in fact closed under complementation may seem a little surprising. As there are many classes of decision problems, such as Σ^0_1 and $\mathcal{NP}$, which are not closed under complementation (or at least not believed to be). However decision a problem $B \subseteq \mathcal{A}^*$ is in Σ^0_1 or NP not because there is a definite machine which is able to decide it, but because the elements of B and only B can be found in some way. The elements of $\mathcal{A}^* \setminus B$ are not necessarily obtainable in any specified manner.

Proposition 6.0.7 follows from the fact that a finite $\mathfrak{LS}$-theory machine can only compute $A \subseteq \mathcal{A}^*$ if it can decide whether $w \in A$ or $w \notin A$ given *any* element of $\mathcal{A}^*$. So we can just use the same machine to compute $\mathcal{A}^* \setminus A$.

6.1 Combining Theory Machines

Suppose functions $f :\subseteq \mathcal{A}^a \to \mathcal{B}^b$ and $g :\subseteq \mathcal{B}^b \to \mathcal{C}^c$ are both computable by finite $\mathfrak{LS}$-theory machines, an important question is whether $g \circ f :\subseteq \mathcal{A}^a \to \mathcal{C}^c$ is also computable by a finite $\mathfrak{LS}$-theory machine. Similarly, if $A, B \subseteq \mathcal{A}^a$ are both finite $\mathfrak{LS}$-computable then are both $A \cap B$ and $A \cup B$ finite $\mathfrak{LS}$-computable?

For many sorts of computational device [50, 65], this functional concatenation along with the intersection and union of problems follows trivially. But the theory machines in a given class may characterise many different sorts of computational device, so as we shall see, these properties do not always hold.

6.1.1 Joining Theories

Given a finite $\mathfrak{LS}$-theory machine $\mathcal{M}_1$ which is able to compute $f :\subseteq \mathcal{A}^a \to \mathcal{B}^b$ and a finite $\mathfrak{LS}$-theory machine $\mathcal{M}_2$ which is able to compute $g :\subseteq \mathcal{B}^b \to \mathcal{C}^c$, we wish to construct a finite theory machine which is able to compute $g \circ f :\subseteq \mathcal{A}^a \to \mathcal{C}^c$. An obvious candidate would be a $\mathfrak{LS}$-theory machine whose models are exactly the structures which consist of a model of $\mathcal{M}_1$ unified with a model of $\mathcal{M}_2$. Ideally providing a structure which characterises a computation of $\mathcal{M}_1$ followed by a computation of $\mathcal{M}_2$.

Such a machine requires a logical system in which this structural unification can be described. But before we can describe these logical systems, we give the following definition which explains what we formally mean when we refer to the unification of two $\mathfrak{LS}$-structures.

Definition 6.1.1 Let $\mathfrak{LS}$ be a logical system, $\mathcal{V}_1$ and $\mathcal{V}_2$ be disjoint $\mathfrak{LS}$-vocabularies, and $\mathcal{V} \supseteq \mathcal{V}_1 \cup \mathcal{V}_2$. Let $\mathfrak{A}_1$ be an $\mathfrak{LS}_{\mathcal{V}_1}$-structure, and $\mathfrak{A}_2$ be an $\mathfrak{LS}_{\mathcal{V}_2}$-structure.

An $\mathfrak{LS}_{\mathcal{V}}$-structure $\mathfrak{A}$ is a *join* of $\mathfrak{A}_1$ and $\mathfrak{A}_2$ if $\mathrm{dom}(\mathfrak{A}) = \mathrm{dom}(\mathfrak{A}_1) \cup \mathrm{dom}(\mathfrak{A}_2)$, and $\mathfrak{A}|_{\mathrm{dom}(\mathfrak{A}_1)}$ and $\mathfrak{A}|_{\mathrm{dom}(\mathfrak{A}_2)}$ are $\mathfrak{LS}_{\mathcal{V}}$-extensions of $\mathfrak{A}_1$ and $\mathfrak{A}_2$ respectively. We denote this by $\mathfrak{A} = \mathfrak{A}_1 \cup \mathfrak{A}_2$.

Suppose that we are given two $\mathfrak{LG}$-theories $\mathcal{T}_1$ and $\mathcal{T}_2$. Our goal is to construct an $\mathfrak{LG}$-theory $\mathcal{S}$ whose collection of models are exactly the structures which are a model of $\mathcal{T}_1$ unified with a model of $\mathcal{T}_2$. Formally we require that $\mathcal{S}$ is as follows.

Definition 6.1.2 Let $\mathfrak{LG}$ be a logical system, and $\mathcal{V}_1$ and $\mathcal{V}_2$ be disjoint $\mathfrak{LG}$-vocabularies. Let $\mathcal{T}_1$ be an $\mathfrak{LG}_{\mathcal{V}_1}$-theory and $\mathcal{T}_2$ be an $\mathfrak{LG}_{\mathcal{V}_2}$-theory.

Suppose that $\mathcal{S}$ is an $\mathfrak{LG}$-theory such that $\mathfrak{A}$ is a model of $\mathcal{S}$ iff $\mathfrak{A} = \mathfrak{A}_1 \cup \mathfrak{A}_2$, for some $\mathfrak{A}_1 \models_{\mathfrak{LG}_{\mathcal{V}_1}} \mathcal{T}_1$ and some $\mathfrak{A}_2 \models_{\mathfrak{LG}_{\mathcal{V}_2}} \mathcal{T}_2$. We then say that $\mathcal{S}$ *joins* $\mathcal{T}_1$ and $\mathcal{T}_2$.

Definition 6.1.3 A logical system $\mathfrak{LG}$ is *joinable* if for any $\mathfrak{LG}_{\mathcal{V}_1}$-theory $\mathcal{T}_1$ and any $\mathfrak{LG}_{\mathcal{V}_2}$-theory $\mathcal{T}_2$ such that $\mathcal{V}_1$ and $\mathcal{V}_2$ are disjoint, there exists an $\mathfrak{LG}$-theory $\mathcal{S}$ that joins $\mathcal{T}_1$ and $\mathcal{T}_2$.

We shall demonstrate below that if $\mathfrak{LG}$ is joinable, then functional concatenation can be achieved within the class of finite $\mathfrak{LG}$-computable problems.

As we shall prove in Lemma 6.1.7 first-order logic is joinable. To join together two theories of first-order logic $\mathcal{T}_1, \mathcal{T}_2$ we add two new unary relation symbols $P_1, P_2 \notin (\mathcal{V}_1 \cup \mathcal{V}_2)$ to the vocabulary with $P_1(x)$ meaning "x is within a structure that models $\mathcal{T}_1$" and $P_2(x)$ meaning "x is within a structure that models $\mathcal{T}_2$". In each sentence of $\mathcal{T}_i$ we then modify each instance of quantification to quantify only over the elements in the appropriate structure.

Definition 6.1.4 [39, 66] Let $\mathfrak{LG}$ contain first-order logic. For any $\mathfrak{LG}$-sentence ϕ and unary predicate P let ϕ^P denote the sentence obtained by replacing each instance of $\forall$ with $\forall^P$ and $\exists$ with $\exists^P$. Where:

$$\forall^P x(\psi(x)) \equiv \forall x(P(x) \rightarrow \psi(x)) \quad \text{and} \quad \exists^P x(\psi(x)) \equiv \exists x(P(x) \wedge \psi(x)).$$

We then say that ϕ^P is *the sentence* ϕ *sorted by* P.

For a set of first-order sentences $\mathcal{T}$ let $\mathcal{T}^P = \{\phi^P \mid \phi \in \mathcal{T}\}$, and say that $\mathcal{T}^P$ is *the set* $\mathcal{T}$ *sorted by* P.

Example 6.1.5 Let $\mathcal{V} = \{Q, S, f, g, c\}$ where Q is a binary relation, S is a unary relation, f is a binary function, g is a unary function, and c is a constant. Consider the theory:

$$\mathcal{T} = \left\{ \begin{array}{l} S(c), \\ \forall x S(g(x)), \\ \exists x \forall y (Q(x,c) \vee S(f(x,y))), \\ \underline{\forall}^1 R \exists x (R(x) \to S(x)) \end{array} \right\}.$$

Let $P \notin \mathcal{V}$ be a unary relation. We can then sort $\mathcal{T}$ by P, obtaining the set of sentences:

$$\mathcal{T}^P = \left\{ \begin{array}{l} S(c), \\ \forall^P x S(g(x)), \\ \exists^P x \forall^P y (Q(x,c) \vee S(f(x,y))), \\ \underline{\forall}^1 R \exists^P x (R(x) \to S(x)) \end{array} \right\},$$

which is equivalent to:

$$\left\{ \begin{array}{l} S(c), \\ \forall x (P(x) \to S(g(x))), \\ \exists x (P(x) \wedge \forall y (P(y) \to (Q(x,c) \vee S(f(x,y))))), \\ \underline{\forall}^1 R \exists x (P(x) \wedge (R(x) \to S(x))) \end{array} \right\}.$$

The set of sentences $\mathcal{T}^P$ then applies exactly to the elements in which $P(x)$ is true.

Implicitly we take logical structures to be closed under functional assignments and to contain all of a vocabulary's constants. These properties are not usually specified, but might well be necessary in order for a theory to specify the appropriate structures. We therefore have the following definition.

Definition 6.1.6 For a unary relation P, a constant c, and an n-ary function f let:

$$\begin{aligned} &\mathcal{P}_c^P \equiv P(c), \text{ and} \\ &\mathcal{P}_f^P \equiv \forall^P x_1 \cdots \forall^P x_n P(f(x_1, \ldots, x_n)). \end{aligned}$$

For a vocabulary $\mathcal{V}$ the set of *P-preservation sentences for $\mathcal{V}$* is:

$$\mathcal{P}_{\mathcal{V}}^P = \{\mathcal{P}_s^P \mid s \text{ is a constant or a function in } \mathcal{V}\}.$$

Lemma 6.1.7 *First-order logic is joinable.*

Proof: Let $\mathfrak{T}_1$ and $\mathfrak{T}_2$ be two first-order theories in disjoint vocabularies $\mathcal{V}_1$ and $\mathcal{V}_2$ respectively. Let $P_1, P_2 \notin \mathcal{V}_1 \cup \mathcal{V}_2$ be unary relations, and $\mathcal{V} = \mathcal{V}_1 \cup \mathcal{V}_2 \cup \{P_1, P_2\}$. Consider the $\mathcal{V}$-theory:

$$\mathfrak{J}^{(P_1,P_2)}_{(\mathfrak{T}_1,\mathfrak{T}_2;\mathcal{V}_1,\mathcal{V}_2)} = \mathfrak{T}_1^{P_1} \cup \mathfrak{P}^{P_1}_{\mathcal{V}_1} \cup \mathfrak{T}_2^{P_2} \cup \mathfrak{P}^{P_2}_{\mathcal{V}_2}.$$

This theory joins $\mathfrak{T}_1$ and $\mathfrak{T}_2$.

To see this, suppose that $\mathfrak{A}_1 \models_{\mathfrak{LS}_{\mathcal{V}_1}} \mathfrak{T}_1$ and $\mathfrak{A}_2 \models_{\mathfrak{LS}_{\mathcal{V}_2}} \mathfrak{T}_2$. Let $\mathfrak{A}'_1$ be a $\mathcal{V}$-expansion of $\mathfrak{A}_1$ in which $P_1(p)$ is true for every $p \in \mathrm{dom}(\mathfrak{A}'_1)$, and let $\mathfrak{A}'_2$ be a $\mathcal{V}$-expansion of $\mathfrak{A}_2$ in which $P_2(q)$ is true for every $q \in \mathrm{dom}(\mathfrak{A}'_2)$.

The join $\mathfrak{A} = \mathfrak{A}'_1 \cup \mathfrak{A}'_2$ then $FO_{\mathcal{V}}$-models $\mathfrak{J}^{(P_1,P_2)}_{(\mathfrak{T}_1,\mathfrak{T}_2;\mathcal{V}_1,\mathcal{V}_2)}$ as clearly $\mathfrak{A}$ satisfies $\mathfrak{P}^{P_1}_{\mathcal{V}_1} \cup \mathfrak{P}^{P_2}_{\mathcal{V}_2}$, since $\mathfrak{A}'_1$ and $\mathfrak{A}'_2$ are by definition closed under the functions of $\mathcal{V}_1$ and $\mathcal{V}_2$ respectively. We also have $\mathfrak{A} \models_{FO_{\mathcal{V}}} \mathfrak{T}_1^{P_1}$, as the truth of $\mathfrak{T}_1^{P_1}$ depends only on the elements of $\mathfrak{A}$ for which P is true, which are exactly the elements in $\mathfrak{A}'_1$, for which $\mathfrak{T}_1$ holds. Similarly $\mathfrak{A} \models_{FO_{\mathcal{V}}} \mathfrak{T}_2^{P_2}$ as the truth of $\mathfrak{T}_2^{P_2}$ depends only on the elements in $\mathfrak{A}'_2$, for which P_2 is true.

Now suppose that $\mathfrak{B} \models_{FO_{\mathcal{V}}} \mathfrak{J}^{(P_1,P_2)}_{(\mathfrak{T}_1,\mathfrak{T}_2;\mathcal{V}_1,\mathcal{V}_2)}$, then clearly $\mathfrak{B}$ has two partial-substructures $\mathfrak{B}_{P_1}$ and $\mathfrak{B}_{P_2}$, such that $x \in \mathrm{dom}(\mathfrak{B}_{P_i})$ iff $P_i(x)$ is true. As $\mathfrak{B}$ models $\mathfrak{P}^{P_1}_{\mathcal{V}_1} \cup \mathfrak{P}^{P_2}_{\mathcal{V}_2}$, both $\mathfrak{B}_{P_1}$ and $\mathfrak{B}_{P_2}$ must be respectively $\mathcal{V}_1$-closed and $\mathcal{V}_2$-closed. Hence $\mathfrak{B}'_{P_1}$ the $\mathcal{V}_1$-reduct of $\mathfrak{B}_{P_1}$, and $\mathfrak{B}'_{P_2}$ the $\mathcal{V}_2$-reduct of $\mathfrak{B}_{P_2}$ are $FO_{\mathcal{V}}$-structures. Now as $\mathfrak{T}_1^{P_1}$ does not reference any element of $\mathrm{dom}(\mathfrak{B})$ that lies outside of $\mathrm{dom}(\mathfrak{B}_{P_1})$, the structure $\mathfrak{B}'_{P_1}$ must satisfy $\mathfrak{T}_1$. Similarly as $\mathfrak{T}_2^{P_2}$ only references elements of $\mathrm{dom}(\mathfrak{B}_{P_2})$, the structure $\mathfrak{B}'_{P_2}$ satisfies $\mathfrak{T}_2$. Therefore $\mathfrak{B}$ must be a join of structures that satisfy $\mathfrak{T}_1$ and $\mathfrak{T}_2$. ❏

Lemma 6.1.8 *Any logical system which contains first-order logic* [1] *is joinable.*

Proof: If a logical system $\mathfrak{LS}$ contains first-order logic then the theory $\mathfrak{J}^{(P_1,P_2)}_{(\mathfrak{T}_1,\mathfrak{T}_2;\mathcal{V}_1,\mathcal{V}_2)}$ from proof of Lemma 6.1.7 can be defined in exactly the same manner and we can follow exactly the same argument as above with $\mathfrak{LS}$ replacing first-order logic.

[1]Such as every logical system defined in Subsection 3.2.1

In some logical systems (such as first-order real logic) every $\mathfrak{LS}$-model must have the same domain, in which case the sorting relations are unnecessary and the theory $\mathcal{T}_1 \cup \mathcal{T}_2$ joins $\mathcal{T}_1$ and $\mathcal{T}_2$.

❑

6.1.2 Concatenating and Combining Theory Machines

Theorem 6.1.9 *Let $\mathfrak{LS}$ be a logical system which contains first-order logic. If the word function problems $f : \mathcal{A}^a \to \mathcal{B}^b$ and $g : \mathcal{B}^b \to \mathcal{C}^c$ where $a, b, c \in \{*, \omega\}$ are computable by a finite $\mathfrak{LS}$-theory machine, then $g \circ f$ is computable by a finite $\mathfrak{LS}$-theory machine.*

Proof: Let $\mathcal{M}_1 = (\mathcal{T}_1, \mathcal{I}_1, \mathcal{O}_1)$ be a finite $\mathfrak{LS}_{\mathcal{V}_1}$-theory machine that is able to compute f with inputs $\hat{\mathcal{A}}^a_{\mathcal{X}_1} \subseteq \mathcal{I}_1$ and outputs $\hat{\mathcal{B}}^b_{\mathcal{Y}_1} \subseteq \mathcal{O}_1$, as well as equality relation $=_1 \in \mathcal{V}_1$, which satisfies the equality axioms $EQ^{=_1}_{\mathcal{V}_1} \subseteq \mathcal{T}_1$ (Definition A.1.1).

Similarly, let $\mathcal{M}_2 = (\mathcal{T}_2, \mathcal{I}_2, \mathcal{O}_2)$ be a finite $\mathfrak{LS}_{\mathcal{V}_2}$-theory machine that is able to compute g with inputs $\hat{\mathcal{B}}^b_{\mathcal{X}_2} \subseteq \mathcal{I}_2$ and outputs $\hat{\mathcal{C}}^c_{\mathcal{Y}_2} \subseteq \mathcal{O}_2$, as well as equality relation $=_2 \in \mathcal{V}_2$, with $EQ^{=_2}_{\mathcal{V}_2} \subseteq \mathcal{T}_2$.

Clearly $\mathcal{B} \subseteq (\mathcal{V}_1 \cap \mathcal{V}_2)$, so $\mathcal{V}_1$ and $\mathcal{V}_2$ are not disjoint. However to ensure disjointness we can just replace $\mathcal{V}_2$ with $\underline{\mathcal{V}}_2 = \{\underline{v} \mid v \in \mathcal{V}_2\}$ and relabel the sentences of $\mathcal{M}_2 = (\mathcal{T}_2, \mathcal{I}_2, \mathcal{O}_2)$ to give us the finite $\mathfrak{LS}_{\underline{\mathcal{V}}_2}$-theory machine $\underline{\mathcal{M}}_2 = (\underline{\mathcal{T}}_2, \underline{\mathcal{I}}_2, \underline{\mathcal{O}}_2)$. Clearly $\mathcal{M}_2$ and $\underline{\mathcal{M}}_2$ are able to compute exactly the same problems as each model of $\mathcal{M}_2$ is isomorphic to a model of $\underline{\mathcal{M}}_2$.

Also let the output simple sequence of $\mathcal{M}_1$, and the input simple sequence of $\underline{\mathcal{M}}_2$ be:

$$\mathcal{Y}_1 = \{\gamma_1(\sigma_1^n(\delta_1))\}_{n\in\mathbb{N}} \quad \text{and} \quad \underline{\mathcal{X}}_2 = \{\gamma_2(\sigma_2^n(\delta_2))\}_{n\in\mathbb{N}}.$$

Consider the $\mathfrak{LS}_{\mathcal{V}}$-theory machine $\mathcal{M}' = (\mathcal{T}', \mathcal{I}_1, \mathcal{O}_2)$ with vocabulary:

$$\mathcal{V} = \mathcal{V}_1 \cup \underline{\mathcal{V}}_2 \sqcup \{P_1, P_2, \vartheta\},$$

where P_1, P_2 are unary relations and ϑ is a unary function. Let $\mathcal{M}'$ have theory:

$$\mathcal{T}' = \mathcal{J}^{(P_1,P_2)}_{(\mathcal{T}_1,\underline{\mathcal{T}}_2;\mathcal{V}_1,\underline{\mathcal{V}}_2)} \cup \mathcal{K}^{P_1}_{(\mathcal{B};\mathcal{Y}_1,\underline{\mathcal{X}}_2;\vartheta)}.$$

Where $\mathcal{J}^{(P_1,P_2)}_{(\mathcal{T}_1,\underline{\mathcal{T}}_2;\mathcal{V}_1,\underline{\mathcal{V}}_2)}$ is as in proof of Lemma 6.1.7. Whereas $\mathcal{K}^{P_1}_{(\mathcal{B};\mathcal{Y}_1,\underline{\mathcal{X}}_2;\vartheta)}$ defines ϑ to be a function that maps every output in $\hat{\mathcal{B}}^b_{\mathcal{Y}_1}$ onto its corresponding input in $\underline{\hat{\mathcal{B}}}^b_{\underline{\mathcal{X}}_2}$:

$$\mathcal{K}^{P_1}_{(\mathcal{B};\mathcal{Y}_1,\underline{\mathcal{X}}_2;\vartheta)} = \left\{ \begin{array}{l} \vartheta(\delta_1) =_2 \delta_2, \\ \forall^{P_1} x(\vartheta(\sigma_1(x)) =_2 \sigma_2(\vartheta(x))), \\ \forall^{P_1} x \bigwedge\limits_{s \in (\mathcal{B} \cup \{\mathbf{b}\})} ((\gamma_1(x) =_1 s) \to (\gamma_2(\vartheta(x)) =_2 \underline{s})) \end{array} \right\}.$$

$\mathcal{J}^{(P_1,P_2)}_{(\mathcal{T}_1,\underline{\mathcal{T}}_2;\mathcal{V}_1,\underline{\mathcal{V}}_2)}$ and $\mathcal{K}^{P_1}_{(\mathcal{B};\mathcal{Y}_1,\underline{\mathcal{X}}_2;\vartheta)}$ are finite, so $\mathcal{M}'$ is also a finite $\mathfrak{LS}$-theory machine.

Let $\mathfrak{A}$ be a model of $\mathcal{T}'$. For any $p, q \in \mathrm{dom}(\mathfrak{A})$ if $p = \sigma_1^n(\delta_1)$ and $q = \sigma_2^n(\delta_2)$ for some $n \in \mathbb{N}$, then we have $\vartheta(p) =_2 q$ in $\mathfrak{A}$, this follows from the first and second sentences of $\mathcal{K}^{P_1}_{(\mathcal{B};\mathcal{Y}_1,\underline{\mathcal{X}}_2;\vartheta)}$ and induction on n. Hence, by the third sentence we must also have that if $\gamma_1(p) =_1 s$ in $\mathfrak{A}$ then $\gamma_2(q) =_2 \underline{s}$ in $\mathfrak{A}$.

Therefore if $\mathfrak{A} \models_{\mathfrak{LS}_\mathcal{V}} \Phi^b_{\mathcal{Y}_1}(v)$ for some $v \in \mathcal{B}^b$, then we must also have $\mathfrak{A} \models_{\mathfrak{LS}_\mathcal{V}} \Phi^b_{\underline{\mathcal{X}}_2}(v)$.

Now let $\mathfrak{A} \models_{\mathfrak{LS}_\mathcal{V}} \mathcal{T}' \cup \Phi^a_{\mathcal{X}_1}(w)$ for some $w \in \mathcal{A}^a$. By Lemma 6.1.8, since $\mathcal{J}^{(P_1,P_2)}_{(\mathcal{T}_1,\underline{\mathcal{T}}_2;\mathcal{V}_1,\mathcal{V}_2)} \subset \mathcal{T}'$ the structure $\mathfrak{A}$ is a join of a model $\mathfrak{A}_1$ of $\mathcal{T}_1$ and a model $\mathfrak{A}_2$ of $\underline{\mathcal{T}}_2$. Additionally $\mathfrak{A}_1 \models_{\mathfrak{LS}_\mathcal{V}} \Phi^a_{\mathcal{X}_1}(w)$, so since $\mathcal{M}_1$ computes f we also have $\mathfrak{A}_1 \models_{\mathfrak{LS}_\mathcal{V}} \Phi^b_{\mathcal{Y}_1}(f(w))$.

Hence $\mathfrak{A} \models_{\mathfrak{LS}_\mathcal{V}} \Phi^b_{\mathcal{Y}_1}(f(w))$, so by our reasoning above $\mathfrak{A} \models_{\mathfrak{LS}_\mathcal{V}} \Phi^b_{\underline{\mathcal{X}}_2}(f(w))$. Which means that $\mathfrak{A}_2 \models_{\mathfrak{LS}_\mathcal{V}} \Phi^b_{\underline{\mathcal{X}}_2}(f(w))$, and as $\mathcal{M}_2$ computes g we have $\mathfrak{A}_2 \models_{\mathfrak{LS}_\mathcal{V}} \Phi^c_{\underline{\mathcal{Y}}_2}(g(f(w)))$. Therefore $\mathfrak{A} \models_{\mathfrak{LS}_\mathcal{V}} \Phi^c_{\underline{\mathcal{Y}}_2}(g(f(w)))$, and as $\mathfrak{A}$ was arbitrary we have:

$$\mathcal{M}'(\Phi^a_{\mathcal{X}_1}(w)) = \Phi^c_{\underline{\mathcal{Y}}_2}(g \circ f(w)),$$

for any $w \in \mathcal{A}^a$. ❑

Definition 6.1.10 Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be $\mathfrak{LS}$-theory machines as in the proof of Theorem 6.1.9 above. We then refer to $\mathcal{M}'$ above as the $(\mathcal{B};\mathcal{Y}_1,\mathcal{X}_2)$-*concatenation* of $\mathcal{M}_1$ and $\mathcal{M}_2$ which we denote by $\mathcal{M}_2 \circ_{(\mathcal{B};\mathcal{Y}_1,\mathcal{X}_2)} \mathcal{M}_1$.

Note how f and g in Theorem 6.1.9 are total functions, an obvious question is whether the above result works for partial functions as well. It so happens that the concatenation of two finite $\mathfrak{LS}$-computable partial functions is not in general finite $\mathfrak{LS}$-computable, see Corollary 8.3.7 for details.

However as the below example indicates, the concatenation of two Turing computable finite word functions is necessarily Turing computable.

Example 6.1.11 Let M_1 and M_2 be Turing machines that compute functions $f :\subseteq \mathcal{A}^* \to \mathcal{B}^*$ and $g :\subseteq \mathcal{B}^* \to \mathcal{C}^*$ respectively. We can then characterise M_1 and M_2 as in Example 5.1.2 by $SO^=$-theory machines:

$$\mathcal{TM}_{M_1} = (\mathcal{TMT}_{M_1}, \hat{\mathcal{A}}^*_{\mathcal{X}_{TM}}, \hat{\mathcal{B}}^*_{\mathcal{Y}_{TM}}), \text{ and } \mathcal{TM}_{M_2} = (\mathcal{TMT}_{M_2}, \hat{\mathcal{B}}^*_{\mathcal{X}_{TM}}, \hat{\mathcal{C}}^*_{\mathcal{Y}_{TM}}).$$

The $(\mathcal{B}; \mathcal{Y}_{TM}, \mathcal{X}_{TM})$-concatenation $\mathcal{M}_2 \circ_{(\mathcal{B};\mathcal{Y}_{TM},\mathcal{X}_{TM})} \mathcal{M}_1$ is then able to compute $g \circ f :\subseteq \mathcal{A}^* \to \mathcal{C}^*$.

Theorem 6.1.12 *Let $\mathfrak{LS}$ be a logical system which contains first-order logic. If word problems $A, B \subseteq \mathcal{A}^a$ where $a \in \{*, \omega\}$ are computable by a finite $\mathfrak{LS}$-theory machine then $A \cap B$ and $A \cup B$ are computable by a finite $\mathfrak{LS}$-theory machine.*

Proof: Let $\mathcal{M}_1 = (\mathcal{T}_1, \mathcal{I}_1, \mathcal{O}_1)$ be a finite $\mathfrak{LS}_{\mathcal{V}_1}$-theory machine that is able to compute $A \subseteq \mathcal{A}^a$, with inputs $\hat{\mathcal{A}}^a_{\mathcal{X}_1} \subseteq \mathcal{I}_1$, and outputs $\Theta_1, \Psi_1 \in \mathcal{O}_1$ corresponding to accepting and rejecting[1] respectively. Also let $\mathcal{M}_1$ have equality relation $=_1 \in \mathcal{V}_1$, which satisfies the equality axioms $EQ^{=_1}_{\mathcal{V}_1} \subseteq \mathcal{T}_1$.

Similarly, let $\mathcal{M}_2 = (\mathcal{T}_2, \mathcal{I}_2, \mathcal{O}_2)$ be a finite $\mathfrak{LS}_{\mathcal{V}_2}$-theory machine that is able to compute $B \subseteq \mathcal{A}^a$, with inputs $\hat{\mathcal{A}}^a_{\mathcal{X}_2} \subseteq \mathcal{I}_2$, and outputs $\Theta_2, \Psi_2 \in \mathcal{O}_2$ corresponding to accepting and rejecting respectively. Also let $\mathcal{M}_2$ have equality relation $=_2 \in \mathcal{V}_2$, which satisfies the equality axioms $EQ^{=_2}_{\mathcal{V}_2} \subseteq \mathcal{T}_2$.

As in the proof of Theorem 6.1.9 we can replace $\mathcal{V}_2$ with $\underline{\mathcal{V}}_2$ to ensure that $\mathcal{V}_1 \cap \underline{\mathcal{V}}_2 = \emptyset$.

[1]As in Definition 4.2.10

By definition, Ψ_1 must be a finite set of sentences, so let $\psi_1 = \bigwedge_{\phi \in \Psi_1} \phi$. We then have $\Psi_1 \models_{\mathfrak{LS}_{\mathcal{V}_1}} \psi_1$ and $\psi_1 \models_{\mathfrak{LS}_{\mathcal{V}_1}} \Psi_1$. Similarly let $\underline{\psi}_2 = \bigwedge_{\phi \in \underline{\Psi}_2} \phi$.

Consider the finite $\mathfrak{LS}_{\mathcal{V}}$-theory machine $\mathcal{M}'' = (\mathcal{T}'', \mathcal{I}_1, \mathcal{O}'')$, with vocabulary $\mathcal{V} = \mathcal{V}_1 \cup \underline{\mathcal{V}}_2 \sqcup \{P_1, P_2, \vartheta\}$, where P_1, P_2 are unary relations and ϑ is a unary function. Let $\mathcal{M}''$ have theory:

$$\mathcal{T}' = \mathcal{J}^{(P_1,P_2)}_{(\mathcal{T}_1,\underline{\mathcal{T}}_2;\mathcal{V}_1,\underline{\mathcal{V}}_2)} \cup \mathcal{K}^{P_1}_{(\mathcal{A};\mathcal{X}_1,\underline{\mathcal{X}}_2;\vartheta)}.$$

where $\mathcal{J}^{(P_1,P_2)}_{(\mathcal{T}_1,\underline{\mathcal{T}}_2;\mathcal{V}_1,\underline{\mathcal{V}}_2)}$ and $\mathcal{K}^{P_1}_{(\mathcal{A};\mathcal{X}_1,\underline{\mathcal{X}}_2;\vartheta)}$ are as in the proof of Theorem 6.1.9, with the terms of $\mathcal{X}_1$ replacing those of $\mathcal{Y}_1$.

Let $\mathfrak{B}$ be a model of $\mathcal{T}''$. By our reasoning in the proof of Theorem 6.1.9, we can see that if $\mathfrak{B} \models_{\mathfrak{LS}_{\mathcal{V}}} \Phi^a_{\mathcal{X}_1}(w)$ for some $w \in \mathcal{A}^a$, then $\mathfrak{B} \models_{\mathfrak{LS}_{\mathcal{V}}} \Phi^a_{\underline{\mathcal{X}}_2}(w)$. So each input of $\mathcal{M}_1$ is copied and becomes an input of $\underline{\mathcal{M}}_2$ as well.

Let $\mathcal{M}''$ have output set:

$$\mathcal{O}'' = \{\Theta_1 \cup \underline{\Theta}_2, \{\psi_1 \vee \underline{\psi}_2\}\}.$$

So let $\mathfrak{B} \models_{\mathfrak{LS}_{\mathcal{V}}} \mathcal{T}'' \cup \Phi^a_{\mathcal{X}_1}(w)$ for some $w \in \mathcal{A}^a$. By Lemma 6.1.8, since $\mathcal{J}^{(P_1,P_2)}_{(\mathcal{T}_1,\underline{\mathcal{T}}_2;\mathcal{V}_1,\underline{\mathcal{V}}_2)} \subset \mathcal{T}''$ the structure $\mathfrak{B}$ is a join of a model $\mathfrak{B}_1$ of $\mathcal{T}_1 \cup \Phi^a_{\mathcal{X}_1}(w)$ and a model $\mathfrak{B}_2$ of $\underline{\mathcal{T}}_2 \cup \Phi^a_{\underline{\mathcal{X}}_2}(w)$.

If $w \in A \cap B$ then by the definition of $\mathcal{M}_1$ and $\mathcal{M}_2$ we have $\mathfrak{B}_1 \models_{\mathfrak{LS}_{\mathcal{V}}} \Theta_1$ and $\mathfrak{B}_2 \models_{\mathfrak{LS}_{\mathcal{V}}} \underline{\Theta}_2$. Hence $\mathfrak{B} \models_{\mathfrak{LS}_{\mathcal{V}}} \Theta_1 \cup \underline{\Theta}_2$. Otherwise if $w \notin A \cap B$ then either $\mathfrak{B}_1 \models_{\mathfrak{LS}_{\mathcal{V}}} \Psi_1$, in which case $\mathfrak{B}_1 \models_{\mathfrak{LS}_{\mathcal{V}}} \psi_1$, or $\mathfrak{B}_2 \models_{\mathfrak{LS}_{\mathcal{V}}} \underline{\Psi}_2$, in which case $\mathfrak{B}_2 \models_{\mathfrak{LS}_{\mathcal{V}}} \underline{\psi}_2$. Therefore $\mathfrak{B} \models_{\mathfrak{LS}_{\mathcal{V}}} \psi_1 \vee \underline{\psi}_2$. Consequently $\mathcal{M}''$ is able to compute $A \cap B$.

To compute $A \cup B$, let $\theta_1 = \bigwedge_{\phi \in \Theta_1} \phi$ and $\underline{\theta}_2 = \bigwedge_{\phi \in \underline{\Theta}_2} \phi$. By replacing the output set $\mathcal{O}''$ in $\mathcal{M}''$ with:

$$\mathcal{O}''' = \{\{\theta_1 \vee \underline{\theta}_2\}, \Psi_1 \cap \underline{\Psi}_2\},$$

we find as above that if $w \in A \cup B$ then $\mathfrak{B}_1 \models_{\mathfrak{LS}_{\mathcal{V}}} \theta_1$ or $\mathfrak{B}_2 \models_{\mathfrak{LS}_{\mathcal{V}}} \underline{\theta}_2$, so therefore $\mathfrak{B} \models_{\mathfrak{LS}_{\mathcal{V}}} \theta_1 \vee \underline{\theta}_2$. Otherwise if $w \notin A \cup B$ then $w \notin A$ and $w \notin B$, hence $\mathfrak{B}_1 \models_{\mathfrak{LS}_{\mathcal{V}}} \Psi_1$ and $\mathfrak{B}_2 \models_{\mathfrak{LS}_{\mathcal{V}}} \underline{\Psi}_2$, and therefore $\mathfrak{B} \models_{\mathfrak{LS}_{\mathcal{V}}} \Psi_1 \cup \underline{\Psi}_2$. ❏

Remark 6.1.13 By Proposition 6.0.7, and Theorems 6.1.9 and 6.1.12, if $\mathfrak{LS}$ is one of the logical systems we defined in Subsection 3.2.1, then the collection of problems computable by the class of finite $\mathfrak{LS}$-theory machines is closed under complementation, intersection, union, and functional concatenation.

Chapter 7

Further Examples of Theory Machines

In this chapter we explain how we can characterise quantum computers and infinite time Turing machines with finite theory machines.

7.1 Quantum Computers

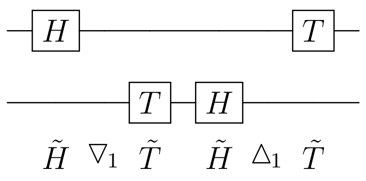
As we explained in Section 2.4, the circuit model of quantum computation involves using a classical computer to computably generate a quantum circuit before implementing a quantum computation on that circuit [50, 59]. We can therefore use the concept of concatenating theory machines that we developed in the previous chapter to characterise this model of quantum computation by a finite theory machine. This is done by combining the quantum circuit implementing machine detailed in Examples 7.1.3 and 7.1.5 below with a Turing machine like the one in Example 5.1.2.

As noted in Section 2.4, we can efficiently approximate any computable quantum circuit using just the Hadamard, $\frac{\pi}{4}$, and controlled-not gates [59]. We can completely describe such a quantum circuit using words constructed from:

$$\Upsilon = \{\tilde{H}, \tilde{T}, \tilde{\oplus}, \triangle_1, \triangledown_1, \triangle_2, \triangledown_2\}. \tag{7.1}$$

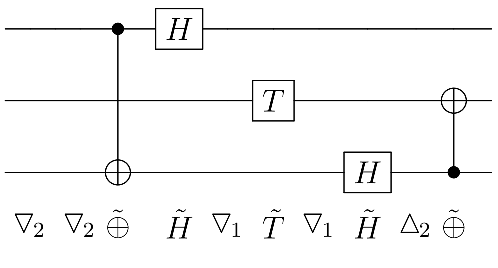
This is sufficient for describing a quantum circuit as the qubit(s) that each gate is applied to can be specified by two pointers which begin at the 0th qubit, and are moved up and down by $\triangle_1, \triangledown_1$ and $\triangle_2, \triangledown_2$. The symbols $\tilde{H}$ and $\tilde{T}$ then indicate that the Hadamard and the $\frac{\pi}{4}$ gate are respectively applied to the qubit pointed to by the first pointer. The symbol $\tilde{\oplus}$ indicates that the controlled-not gate should be implemented with the first pointer pointing to the control qubit and the second pointing to the target qubit. In this way the whole circuit can be specified by a word read from left to right, with the gates being implemented in the order that they are written.

Example 7.1.1 A word of the form $\tilde{H} \triangledown_1 \tilde{T}\tilde{H} \triangle_1 \tilde{T}$ describes the circuit:


H
T
T
H
$\tilde{H}$ $\triangledown_1$ $\tilde{T}$ $\tilde{H}$ $\triangle_1$ $\tilde{T}$

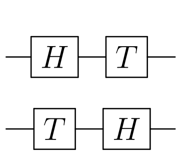
The first symbol $\tilde{H}$ of the word indicates that a Hadamard gate should be applied to the 0th qubit, as the first pointer points to the 0th qubit at the start of the computation. The next symbol $\triangledown_1$ then indicates that this pointer should now point to the 1st qubit, which means that the following symbols $\tilde{T}$ and $\tilde{H}$ indicate that the $\frac{\pi}{4}$ and Hadamard gates should be applied to this qubit. $\triangle_1$ then moves the first pointer back to qubit 0, and so the final symbol $\tilde{T}$ means that the $\frac{\pi}{4}$ gate is applied to this qubit.

Example 7.1.2 A word of the form $\triangledown_2 \triangledown_2 \tilde{\oplus}\tilde{H} \triangledown_1 \tilde{T} \triangledown_1 \tilde{H} \triangle_2 \tilde{\oplus}$ describes the circuit:

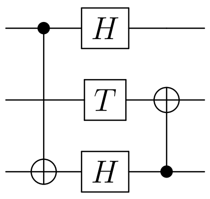

H
T
H
$\triangledown_2$ $\triangledown_2$ $\tilde{\oplus}$ $\tilde{H}$ $\triangledown_1$ $\tilde{T}$ $\triangledown_1$ $\tilde{H}$ $\triangle_2$ $\tilde{\oplus}$

So after the first two symbols $\triangledown_2 \triangledown_2$ of the word move the second pointer to the 2nd qubit, the next symbol $\tilde{\oplus}$ indicates that a controlled-not gate is applied, with the 2nd qubit as the target qubit and the 0th as the control, since the first pointer still points to the 0th qubit. The following string of symbol $\tilde{H}\bigtriangledown_1\tilde{T}\bigtriangledown_1\tilde{H}$ then indicate that a Hadamard, a $\frac{\pi}{4}$, and a Hadamard gate should be applied to the 0th, 1st, and 2nd qubits respectively. Finally $\bigtriangleup_2$ moves the second pointer to the 1st qubit, so $\tilde{\oplus}$ now indicates that a controlled-not gate is applied with the 1st qubit as the target qubit and the 2nd as the control.

Though describing a circuit via this sort of word does imply a sequential implementation of the gates, it is worth noting that the above circuits are equivalent to:



and



So circuits in which single-qubit gates are applied to different qubits simultaneously may still be effectively described in the above manner.

The basis set for N qubits is $\{|x\rangle : x \in \{0,\dots,2^N-1\}\}$, for simplicity in the examples below we will order the qubits from right to left. So for example $|11001\rangle = |1\rangle_4|1\rangle_3|0\rangle_2|0\rangle_1|1\rangle_0$, indicates that the 0th, 3rd and 4th qubits are in state $|1\rangle$, whereas the 1st and 2nd qubits are in state $|0\rangle$.

To apply a single-qubit quantum gate $U = \begin{pmatrix} u_{00} & u_{01} \\ u_{10} & u_{11} \end{pmatrix}$ to the lth qubit of $|\psi\rangle = \sum_{m=0}^{2^N-1}\alpha_m|m\rangle$ we apply the transformation $I^{\otimes(N-l-1)} \otimes U \otimes I^{\otimes(l-1)}$. Such a transformation is fairly complicated to describe, to do so we can look at the binary expansion of each element of $m \in \{0,\dots,2^N-1\}$.

Indeed let B and O be binary functions, such that $B(x,y)$ gives the binary value of the yth digit of x for each $x,y \in \mathbb{N}$. Whereas $O(x,y) = z$ if z and x have the same binary expansion as each other except for the yth digit. So formally for $x,y \in \mathbb{N}$

where $x = \cdots 00b_{N-1}b_{N-2}\cdots b_1 b_0$ in binary[1], we have:

$$B(x,y) = b_y. \tag{7.2}$$

$$O(x,y) = z \text{ if } z = \cdots 00b_N b_{N-1}\cdots b_{y+1}(1-b_y)b_{y-1}\cdots b_1 b_0 \tag{7.3}$$

The values of $I^{\otimes(N-l-1)} \otimes U \otimes I^{\otimes(l-1)} = (u_{mn})_{m,n=1}^{2^N}$ (a $2^N \times 2^N$ complex-valued matrix) are then:

$$v_{mn} = \begin{cases} u_{00} & \text{if } n = m & \text{and } B(m,l) = 0, \\ u_{01} & \text{if } n = O(m,l) & \text{and } B(m,l) = 0, \\ u_{10} & \text{if } n = O(m,l) & \text{and } B(m,l) = 1, \\ u_{11} & \text{if } n = m & \text{and } B(m,l) = 1, \\ 0 & \text{otherwise.} \end{cases}$$

Therefore $I^{\otimes(N-1)} \otimes U \otimes I^{\otimes(l-1)}$ applied to $|\psi\rangle$ results in:

$$\sum_{m=0}^{2^{N-1}}((1-B(m,l))(u_{00}\alpha_m + u_{01}\alpha_{O(m,l)})) + B(m,l)(u_{10}\alpha_{O(m,l)} + u_{11}\alpha_m)))|m\rangle.$$

Note that since $B(m,l) \in \{0,1\}$, for each m, the mth component of the summation above must be equal to either $(u_{00}\alpha_m + u_{01}\alpha_{O(m,l)})|m\rangle$ or $(u_{10}\alpha_{O(m,l)} + u_{11}\alpha_m)|m\rangle$. Specifically, applying the Hadamard gate to the lth qubit of $|\psi\rangle$ results in:

$$\sum_{m=0}^{2^{N-1}}((1-B(m,l))\tfrac{1}{\sqrt{2}}(\alpha_m + \alpha_{O(m,l)})) + B(m,l)\tfrac{1}{\sqrt{2}}(\alpha_{O(m,l)} - \alpha_m)))|m\rangle, \tag{7.4}$$

whereas applying the $\frac{\pi}{4}$ gate to the lth qubit of $|\psi\rangle$ gives:

$$\sum_{m=0}^{2^{N-1}}((1-B(m,l))\alpha_m + B(m,l)e^{\frac{i\pi}{4}}\alpha_m)|m\rangle. \tag{7.5}$$

To apply the controlled-not gate to $|\psi\rangle = \sum_{m=0}^{2^N-1}\alpha_m|m\rangle$ with control qubit l and target qubit k we swap the values of α_m and $\alpha_{O(m,k)}$ if $B(m,l) = 1$, otherwise they remain as they are. Hence we obtain:

$$\sum_{m=0}^{2^{N-1}}((1-B(m,l))\alpha_m + B(m,l)\alpha_{O(m,k)})|m\rangle. \tag{7.6}$$

[1]Note that like with qubits we number the digits of a binary expansion from right to left, allowing for an infinite sequence of zeros on the left.

In the following example we characterise a quantum device by a theory machine in first-order complex logic (FO$\mathbb{C}$) (Definition 3.2.7). This device takes as its input a state $|w\rangle$ where $w \in \{0,1\}^*$ and a quantum circuit description written as a word of Υ^* (from Equation 7.1 above). The device calculates the result of this circuit on input $|w\rangle$ and outputs whether the first qubit is measured to be $|1\rangle$ with a probability greater than $\frac{2}{3}$, or with probability less than or equal to $\frac{1}{3}$.

Example 7.1.3 Consider the finite $FO\mathbb{C}$-theory machine:

$$\mathfrak{QC} = (\mathfrak{QCT}, \mathfrak{I}_Q, \{\{\alpha\}, \{\beta\}\}),$$

in the vocabulary of:

$$\mathcal{V}_Q = \mathcal{V}_{\mathbb{C}+} \cup \{V, B, O, I, C, P_1, P_2, M, s, d\} \cup \Upsilon,$$

where as in Definition A.1.11 the symbols in $\mathcal{V}_{\mathbb{C}+} = \{\leqslant, \mathbb{N}, -, /, \cdot^2, \sqrt{\cdot}, |\cdot|, 2^\wedge, 2, e^{i\frac{\pi}{4}}\}$ have their usual meanings in $\mathbb{C}$. Whereas V, B, O are binary functions such that $V(x,y)$ gives the complex value of the yth basis state at time x, and B, O are defined as in Equations 7.2 and 7.3 above. C, I, P_1, P_2, M are unary functions where $I(m)$ and $C(m)$ give the mth symbol of the input state and circuit word respectively. Whereas $P_1(x)$ and $P_2(x)$ represent the circuit pointers mentioned above at time x, and M expresses the probability that qubit 0 is measured to be $|1\rangle$ at the end of the computation. Finally s, d and the elements of Υ are constants, with s corresponding to the input state, d corresponding to the measurement time, and Υ consisting of the set of symbols used to describe the circuit as in Equation 7.1.

To input a quantum state together with a circuit we have:

$$\mathfrak{I}_Q = \left\{ \Phi^*_{\mathfrak{X}_{QS}}(w) \cup \Phi^*_{\mathfrak{X}_{QC}}(v) \;\middle|\; w \in \{0,1\}^* \text{ and } v \in \Upsilon^* \right\}$$

So the inputs are formed from a finite $\mathfrak{X}_{QS}$-word set and a finite $\mathfrak{X}_{QC}$-word set, where:

$$\mathfrak{X}_{QS} = \{I(\hat{n})\}_{n\in\mathbb{N}} \text{ and } \mathfrak{X}_{QC} = \{C(\hat{n})\}_{n\in\mathbb{N}}.$$

Where $\hat{n} \equiv \underbrace{1 + \cdots + 1}_{n \text{ times}}$ for $n \in \mathbb{N}$. So the finite word set:

$$\Phi^*_{\mathfrak{X}_{QS}}(w) = \left\{ I(\hat{k}) = w_k \mid k \in \{0, \ldots, |w|-1\} \right\} \cup \left\{ I(\widehat{|w|}) = \mathbf{b} \right\},$$

gives the integer binary expansion of the input state via I, with for example $w = 001010$ corresponding to the binary number 10100. Whereas the finite word set:

$$\Phi^*_{\mathcal{X}_{QC}}(v) = \left\{ C(\hat{j}) = v_j \mid k \in \{0, \dots, |w| - 1\} \right\} \cup \left\{ C(\hat{|v|}) = \mathbf{b} \right\},$$

ensures that $C(m)$ maps to the mth symbol of the circuit word v for each $m \in \mathbb{N}$.

The output sentences α and β (defined below in 7.7 and 7.8) indicate whether the quantum circuit should accept, or reject its input. The theory is:

$$\mathfrak{QCT} = CD^{=}_{\mathcal{V}_Q} \cup ACA \cup BOD^{\mathbb{N}} \cup ITQ^{\mathbb{N}} \cup EVQ^{\mathbb{N}} \cup MTQ^{\mathbb{N}} \cup \mathcal{P}^{\mathbb{N}}_{\{B,O,P_1,P_2,M\}}.$$

Where $CD^{=}_{\mathcal{V}_Q}$ is the set of distinct constant axioms for $\mathcal{V}_Q$ (Definition 4.2.6), and ACA is the set of additional real axioms (Definition A.1.11), which ensures that in each model of $\mathfrak{QCT}$ the symbols in $\mathcal{V}_{\mathbb{C}+}$ have their usual definitions (Corollary A.1.13). The set $BOD^{\mathbb{N}}$ defines the functions B and O. The initial configuration of the quantum circuit is defined by the sentences in $ITQ^{\mathbb{N}}$, and is evolved in accordance with those in $EVQ^{\mathbb{N}}$. The function M which expresses the measurement outcome is defined by $MTQ^{\mathbb{N}}$. Finally as in Definition 6.1.6 $\mathcal{P}^{\mathbb{N}}_{\{B,O,P_1,P_2,M\}}$ ensures that B, O, P_1, P_2, M map natural numbers to natural numbers.

Note that much of the theory of $\mathfrak{QC}$ is sorted by $\mathbb{N}$ (Definition 6.1.4), this is because the time step and the qubit states are natural numbers. So the quantified inputs in $BOD \cup ITQ \cup EVQ \cup MTQ$ will all lie in $\mathbb{N}$, but since V is not mentioned in $\mathcal{P}^{\mathbb{N}}_{\{B,O,P_1,P_2,M\}}$, it may map to any element of $\mathbb{C}$.

Let $\mathfrak{Q}$ be an $FO\mathbb{C}$-model of $\mathfrak{QCT} \cup \Phi^*_{\mathcal{X}_{QS}}(w) \cup \Phi^*_{\mathcal{X}_{QC}}(v)$, by definition, such a model is an expansion of the usual structure of the complex numbers, and since $ACA \subseteq \mathfrak{QCT}$ the symbols of $\mathcal{V}_{\mathbb{C}+}$ have their usual definitions in $\mathfrak{Q}$.

So formally, to define B and O as in Equations 7.2 and 7.3 we have:

$$BOD = \left\{ \begin{array}{l} \forall x \forall y (B(x,y) = 0) \vee (B(x,y) = 1), \\ \forall x \forall y (x < (2\hat{\ }(y)))) \to (B(x,y) = 0)), \\ \forall x \forall y \neg (B(x,y) = B(x + (2\hat{\ }(y)), y)), \\ \forall x \forall y \forall z ((B(x,z) = B(O(x,y),z)) \leftrightarrow \neg (y = z)) \end{array} \right\}.$$

$B(x, y)$ corresponds to a bit value, so it always maps to 0 or 1. As $BOD^{\mathbb{N}} \subseteq \mathfrak{QCT}$, the values of $B(x, y)$ for $x, y \in \mathbb{N}$ are defined through induction. The second sentence of $BOD^{\mathbb{N}}$ serves as a base case. If $x < 2^y$ then the yth entry of the binary expansion of x must be 0 as we can view such an expansion to be preceded on the left by an infinite sequence of 0's[1]. The third sentence of $BOD^{\mathbb{N}}$ provides the inductive step, as adding 2^y to x must change the value of the yth entry of x's binary expansion since this corresponds to adding the binary number $1\underbrace{0\cdots0}_{\times(y-1)}$ to x. The value can only be 0 or 1, so any change must cause this value to flip.

The final sentence of $BOD^{\mathbb{N}}$ then states that the function $O(x, y)$ maps to a natural number n such that $B(n, z) = B(x, z)$ for every $z \in \mathbb{N}$ except y, which corresponds to taking:

$$n = \ldots B(n, N) \ldots B(n, y+1)(1 - B(n, y))B(n, y-1) \ldots B(n, 1)B(n, 0).$$

As mentioned before, to express the values of the whole qubit state at each time step we have the function V. Where $V(x, y)$ gives the complex value of the yth basis state at time x, meaning that the whole state of the circuit at time x is:

$$\sum_{y \in \mathbb{N}} V(x, y)|y\rangle.$$

The functions P_1 and P_2 act as the two pointers of the quantum circuit indicating which qubit is to be transformed by the next quantum operation. So at the start of the quantum computation the pointers should both point to the 0th qubit. To ensure this and to define the initial state of V at time 0 we have:

$$ITQ = \left\{ \begin{array}{l} \forall x((I(x) = \mathbf{b}) \to (I(x+1) = \mathbf{b})), \\ \forall x((C(x) = \mathbf{b}) \to (C(x+1) = \mathbf{b})), \\ \forall x(\neg(I(x) = \mathbf{b}) \to (B(s, x) = I(x))), \\ \forall x((I(x) = \mathbf{b}) \to (B(s, x) = 0)), \\ V(0, s) = 1, \\ \forall y(\neg(y = s) \to (V(0, y) = 0)), \\ (P_1(0) = 0) \wedge (P_2(0) = 0) \end{array} \right\}.$$

[1]This corresponds to taking the state of any qubit not affected by the circuit as existing but remaining fixed as $|0\rangle$.

From the input we have $\mathfrak{Q} \models_{FO\mathbb{C}} \{I(|w|+1) = \mathbf{b}\} \cup \{C(|v|+1) = \mathbf{b}\}$. Hence the first two sentences of $ITQ^{\mathbb{N}}$ ensure through induction that $I(m) = \mathbf{b}$ and $C(n) = \mathbf{b}$ in $\mathfrak{Q}$ for all $m > |w|$ and $n > |v|$ (as in Example 4.2.5). The third and fourth sentences of $ITQ^{\mathbb{N}}$ then define the binary expansion of s in $\mathfrak{Q}$ to be $0^{-\omega}w$.

Hence for any basis state $|\psi\rangle$, where $\psi \in \mathbb{N}$ the input $\Phi^*_{\mathfrak{X}_{QS}}(w)$ together with $BOD^{\mathbb{N}}$ and the fifth and sixth sentences of $ITQ^{\mathbb{N}}$ ensure that $V(0, \psi) = 1$ in $\mathfrak{Q}$ iff $|\psi\rangle = |w\rangle$, and otherwise $V(0, \psi) = 0$. Meaning that at time 0 the quantum state of the circuit is exactly $|w\rangle$.

To evolve V via the quantum circuit described by $v \in \Upsilon$ we have:

$$EVQ = \left\{ \begin{array}{l} \forall x((C(x) = \triangle_1) \to (P_1(x+1) = (P_1(x) - 1))), \\ \forall x((C(x) = \nabla_1) \to (P_1(x+1) = (P_1(x) + 1))), \\ \forall x((C(x) = \triangle_2) \to (P_2(x+1) = (P_2(x) - 1))), \\ \forall x((C(x) = \nabla_2) \to (P_2(x+1) = (P_2(x) + 1))), \\ \forall x((C(x) = \tilde{H}) \to \forall y(\eta_1(x,y) \wedge \eta_2(x,y)), \\ \forall x((C(x) = \tilde{T}\,) \to \forall y(\upsilon_1(x,y) \wedge \upsilon_2(x,y)), \\ \forall x((C(x) = \tilde{\oplus}\,) \to \forall y(\kappa_1(x,y) \wedge \kappa_2(x,y)), \\ \forall x(((C(x) = \mathbf{b}) \wedge \neg(C(x-1) = \mathbf{b})) \to (d = x), \end{array} \right\}$$

The first four sentences of $EVQ^{\mathbb{N}}$ define how the pointers move given symbols $\triangle_1, \nabla_1, \triangle_2$, or ∇_2 on the input word $\Phi^*_{\mathfrak{X}_{QC}}(w)$. So at time step x pointer j points to qubit a in $\mathfrak{Q}$ if $P_j(x) = a$, and if $C(x) = v_x = \triangle_j$, then at time $x+1$ pointer j should point to $a - 1$, and so we have $P_j(x+1) = a - 1$ in $\mathfrak{Q}$. Whereas if $C(x) = v_x = \nabla_j$ then the pointer should point to $a + 1$ at time $x + 1$, and so $P_j(x+1) = a + 1$ in $\mathfrak{Q}$.

Given the symbol $\tilde{H}$ a Hadamard gate is applied to the qubit at $P_1(x)$ via the fifth sentence of $EVQ^{\mathbb{N}}$ and the formulas:

$$\eta_1(x,y) \equiv (B(y, P_1(x)) = 0) \to (V(x+1, y) = (V(x,y) + V(x, O(y, P_1(x))))/\sqrt{2}),$$
$$\eta_2(x,y) \equiv (B(y, P_1(x)) = 1) \to (V(x+1, y) = (V(x,y) - V(x, O(y, P_1(x))))/\sqrt{2}).$$

So as in Equation 7.4 $\eta_1(x,y) \wedge \eta_2(x,y)$ ensures that if pointer 1 is pointing to the lth qubit in $\mathfrak{Q}$ and the complex value of the basis state $|y\rangle$ is α_y at time x, then at time

$x+1$ it becomes $\frac{1}{\sqrt{2}}(\alpha_y + \alpha_{O(y,l)})$ in $\mathfrak{Q}$ if the lth digit of y is a 0 and $\frac{1}{\sqrt{2}}(\alpha_y - \alpha_{O(y,l)})$ if the lth digit is a 1.

Given the symbol $\tilde{T}$ a $\frac{\pi}{4}$ gate is applied to the qubit at $P_1(x)$ via the sixth sentence of $EVQ^{\mathbb{N}}$ and the formulas:

$$\upsilon_1(x,y) \equiv (B(y,P_1(x)) = 0) \to (V(x+1,y) = V(x,y)),$$
$$\upsilon_2(x,y) \equiv (B(y,P_1(x)) = 1) \to (V(x+1,y) = (e^{i\frac{\pi}{4}} \times V(x,y))).$$

As in Equation 7.5 $\upsilon_1(x,y) \wedge \upsilon_2(x,y)$ maps $\alpha_y|y\rangle$ at time x to $\alpha_y|y\rangle$ at time $x+1$ in $\mathfrak{Q}$ if $P_1(x)$th digit of y is a 0, and to $e^{\frac{i\pi}{4}}\alpha_y|y\rangle$ at time $x+1$ otherwise.

Given the symbol $\tilde{\oplus}$ a controlled-not gate is applied to the qubits at $P_1(x)$ and $P_2(x)$ via the seventh sentence of $EVQ^{\mathbb{N}}$ and the formulas:

$$\kappa_1(x,y) \equiv (B(y,P_1(x)) = 0) \to (V(x+1,y) = V(x,y)),$$
$$\kappa_2(x,y) \equiv (B(y,P_1(x)) = 1) \to (V(x+1,y) = V(x,O(y,P_2(x)))).$$

As in Equation 7.6 if pointer 1 is pointing to the lth qubit then this is the control qubit. So if the lth digit of y is a 0 then $\kappa_1(x,y)$ just maps $\alpha_y|y\rangle$ at time x to $\alpha_y|y\rangle$ at time $x+1$ in $\mathfrak{Q}$. Whereas if the lth digit of y is a 1 and pointer 2 is pointing to the kth qubit then $\kappa_2(x,y)$ makes the value of the yth basis state equal to the value of the $O(y,k)$th basis state at time $x+1$. Since $O(O(y,k),k) = y$ the seventh sentence swaps the values of yth and $O(y,k)$th basis states in $\mathfrak{Q}$ for each $y \in \mathbb{N}$ with $B(y,l) = 1$.

Recall that the circuit input $\Phi^*_{\chi_{QC}}(w)$ together with the third sentence of $ITQ^{\mathbb{N}}$ implies that $(C(x) = \mathbf{b}) \wedge \neg(C(x-1) = \mathbf{b})$ is true iff $x = |v|$. So at time $|v|$ the circuit ends, hence the quantum computation should end as well, and so the final sentence of $EVQ^{\mathbb{N}}$ fixes this time to be d in $\mathfrak{Q}$.

Finally, we take the quantum circuit as accepting the input if the probability of measuring that the first qubit is $|1\rangle$ is greater than $\frac{2}{3}$, and rejecting if the probability is less than $\frac{1}{3}$. This measurement can happen for a basis state $|y\rangle$ if and only if $B(y,0) = 1$, which is true if and only if $y \in \mathbb{N}$ is odd. Therefore the probability of measuring $|1\rangle$ is equal to $\sum_{y=0}^{\infty} |V(e, 2y+1)|^2$.

To calculate this probability we use the function M, where we define $M(m) = \sum_{y=0}^{m} |V(e, 2y+1)|^2$ inductively as follows:

$$MTQ = \left\{ \begin{array}{l} M(0) = |V(d,1)|^2, \\ \forall y(M(y+1) = (M(y) + |V(d, ((2 \times y) + 1)|^2)) \end{array} \right\}.$$

The outputs are then $\{\alpha\}$ and $\{\beta\}$ where:

$$\alpha \equiv \exists^{\mathbb{N}} y((2/(2+1)) < M(y)), \tag{7.7}$$

$$\beta \equiv \forall^{\mathbb{N}} y(M(y) \leqslant (1/(2+1))). \tag{7.8}$$

Clearly, M is an increasing function, and we may only have $\sum_{y=0}^{\infty} |V(e, 2y+1)|^2 > \frac{2}{3}$ if there exists some $m \in \mathbb{N}$ such that $\sum_{y=0}^{m} |V(e, 2y+1)|^2 > \frac{2}{3}$. Conversely $\sum_{y=0}^{\infty} |V(e, 2y+1)|^2 \leqslant \frac{1}{3}$ only if $\sum_{y=0}^{l} |V(e, 2y+1)|^2 \leqslant \frac{1}{3}$ for every $l \in \mathbb{N}$.

Therefore α is true in $\mathfrak{Q}$ iff the probability of measuring a $|1\rangle$ in the first qubit of the output of the circuit described by v and given input $|w\rangle$ is greater than to $\frac{2}{3}$. Whereas β is true in $\mathfrak{Q}$ iff the probability of this scenario is less than or equal to $\frac{1}{3}$. If the probability is between $\frac{1}{3}$ and $\frac{2}{3}$ then the output is undefined.

Remark 7.1.4 Whilst the output of the theory machine $\mathfrak{QC}$ describes the probability of an event, the result is not itself probabilistic. Instead, the measurement probability of at least $\frac{2}{3}$ or no greater than $\frac{1}{3}$ is obtained by each model of $\mathfrak{QC}$ with certainty.

In Example 7.1.6 below we shall describe how we can combine a theory machine like $\mathfrak{QC}$ above with a Turing machine, creating a device which is able to decide problems in exactly the same way that a quantum computer does (Definition 2.4.5). That is, for a quantum circuit $Q = \{Q_N\}_{n \in \mathbb{N}}$ and each input $w \in \{0,1\}^*$ the device computes with its Turing machine component a word that describes $Q_{|w|}$ before applying $Q_{|w|}$ to $|w\rangle$ and measuring.

However, to concatenate two theory machines they must both use the same logical system. So firstly we shall explain how $\mathfrak{QC}$ can also be characterised in second-order logic with equality, before in Example 7.1.6 combining such a machine with the $SO^=$ description we gave of a Turing machine in Example 5.1.2.

Example 7.1.5 Consider the finite $SO^{=}_{\mathcal{V}'_Q}$-theory machine:

$$\mathcal{QC}' = (\mathcal{QCT}', \mathcal{I}_Q, \{\{\alpha\}, \{\beta\}\}),$$

in the vocabulary of:

$$\mathcal{V}'_Q = \mathcal{V}_Q \cup \{<, \mathbb{R}, +, \times, 0, 1, i\},$$

where $\mathcal{V}_Q$ is as in Example 7.1.3 and $<, \mathbb{R}, +, \times, 0, 1, i$ are the usual symbols of complex arithmetic, with $\mathbb{R}$ being the unary characteristic function of the reals in $\mathbb{C}$. Further let $\mathcal{I}_Q$, α and β be exactly as they are in Example 7.1.3, and let $\mathcal{QC}'$ have the theory:

$$\mathcal{QCT}' = \mathcal{QCT} \cup CAA.$$

Where $\mathcal{QCT}$ is as in Example 7.1.3, and CAA is the set of complex arithmetic axioms (Definition A.1.9), which by Proposition A.1.10 ensures that every model of $\mathcal{QCT}'$ is isomorphic to an expansion of the usual structure of complex arithmetic.

$\mathcal{QCT}'$ is otherwise the same as $\mathcal{QCT}$, and the inputs and the outputs of $\mathcal{QC}'$ are identical to those of $\mathcal{QC}$. Hence by our reasoning in Example 7.1.3, any model of $\mathcal{QC}'$ with input $\Phi^*_{\mathcal{X}_{QS}}(w) \cup \Phi^*_{\mathcal{X}_{QC}}(v)$ must satisfy α if the probability of measuring $|1\rangle$ in the first qubit of the output of the circuit described by v and given input $|w\rangle$ is greater than to $\frac{2}{3}$, and β if this probability is less than or equal to $\frac{1}{3}$.

Example 7.1.6 Let $K = (K_n)_{n\in\mathbb{N}}$ be a computable sequence of quantum circuits constructed from the gate set $\{H, T, CNOT, \}$. Further let M_K be a Turing machine with input alphabet $\{0, 1\}$ which on input $w \in \{0, 1\}^*$ computes the word v where $v \in \Upsilon^*$ describes the circuit $K_{|w|}$.

We can characterise M_K with the finite $SO^{=}$-theory machine:

$$\mathcal{TM}_{M_K} = (\mathcal{TMT}_{M_K}, \widehat{\{0,1\}}^*_{\mathcal{X}_{TM}}, \hat{\Upsilon}^*_{\mathcal{Y}_{TM}}),$$

where the theory $\mathcal{TMT}_{M_K}$ corresponds to $\mathcal{TMT}_{M'}$ in Example 5.1.2 with the rules and states of M' replaced by those of M_K. Let:

$$\mathcal{QC}' \circ_{(\Upsilon;\mathcal{Y}_{TM},\mathcal{X}_{QC})} \mathcal{TM}_{M_K} = (\mathcal{T}_\circ, \widehat{\{0,1\}}^*_{\mathcal{X}_{TM}}, \{\{\underline{\alpha}\}, \{\underline{\beta}\}\}),$$

be the $(\Upsilon; \mathcal{Y}_{TM}, \mathcal{X}_{QC})$-concatenation of $\mathcal{TM}_{M_K}$ and $\mathcal{QC}'$ with vocabulary $\mathcal{V}_\circ = \mathcal{V}_{M_K} \cup \underline{\mathcal{V}}_Q \cup \{P_1, P_2, \vartheta\}$, where as in the proof of Theorem 6.1.9 we have replaced $\mathcal{V}_Q$ with $\underline{\mathcal{V}}_Q$ so as to ensure disjointness.

We can then characterise the computation of the computable quantum circuit K by the finite $SO^=$-theory machine:

$$\mathcal{CQC}_K = (\mathcal{CQCT}_K, \{0 \hat{,} 1\}^*_{\mathcal{X}_{TM}}, \{\{\underline{\alpha}\}, \{\underline{\beta}\}\}),$$

with vocabulary $\mathcal{V}_{CQ_K} = \mathcal{V}_\circ \cup \{\varrho\}$. Where $\mathcal{CQC}_K$'s theory is:

$$\mathcal{CQCT}_K = \mathcal{T}_\circ \cup \mathcal{K}^{P_1}_{(\{0,1\}, \mathcal{X}_{TM}, \underline{\mathcal{X}}_{QS}; \varrho)}.$$

The set $\mathcal{K}^{P_1}_{(\{0,1\}, \mathcal{X}_{TM}, \underline{\mathcal{X}}_{QS}; \varrho)}$ is as in the proofs of Theorems 6.1.9 and 6.1.12, and serves to map the input word of $\mathcal{TM}_{M_K}$ into an input state of $\mathcal{QC}'$.

Since $\mathcal{T}_\circ \subset \mathcal{CQCT}_K$, any model $\mathfrak{K}$ of $\mathcal{CQCT}_K$ is a join of a model $\mathfrak{A}$ of $\mathcal{TMT}_{M_K}$ and a model $\mathfrak{Q}$ of $\mathcal{QCT}$.

Now if $\mathfrak{K} \models_{SO^=_{\mathcal{V}_{CQ_K}}} \Phi^*_{\mathcal{X}_{TM}}(w)$ then $\mathfrak{A} \models_{SO^=_{\mathcal{V}_{M_K}}} \Phi^*_{\mathcal{X}_{TM}}(w)$, and so $\mathfrak{A} \models_{SO^=_{\mathcal{V}_{M_K}}} \Phi^*_{\mathcal{Y}_{TM}}(v)$. Therefore by $\mathcal{T}_\circ$ it follows that $\mathfrak{Q} \models_{SO^=_{\underline{\mathcal{V}}_Q}} \Phi^*_{\underline{\mathcal{X}}_{QC}}(v)$.

From $\mathcal{K}^{P_1}_{(\mathcal{X}_{TM}, \underline{\mathcal{X}}_{QS}; \varrho)}$ and our reasoning in the proof of Theorem 6.1.9 we also have $\mathfrak{Q} \models_{SO^=_{\underline{\mathcal{V}}_Q}} \Phi^*_{\underline{\mathcal{X}}_{QS}}(w)$. Consequently by our reasoning in Example 7.1.3 we have $\mathfrak{Q} \models_{SO^=_{\underline{\mathcal{V}}_Q}} \underline{\alpha}$ if the first qubit is measured to be $|1\rangle$ with a probability greater than $\frac{2}{3}$, and $\mathfrak{Q} \models_{SO^=_{\underline{\mathcal{V}}_Q}} \underline{\beta}$ if it is measured to be $|1\rangle$ with probability less than or equal to $\frac{1}{3}$.

$\mathfrak{K}$ models $\underline{\alpha}$ or $\underline{\beta}$ only if the same scenarios hold. Therefore if the computable quantum circuit K is able to decide the problem $A \subseteq \{0,1\}^*$, then our finite theory machine $\mathcal{CQC}_K$ is also able to decide A.

7.2 Infinite Time Turing Machines

As we described in Section 2.5, infinite time Turing machines (IIT machines) [44] generalise the concept of a multi-tape Turing machine by allowing the computation to take an ordinal number (Definition 2.5.3) of time steps. In the following we characterise an ITT machine by a finite theory machine.

Example 7.2.1 Let $V = (\Pi, s_0, s_\lambda, s_1, \mathcal{S})$ be an ITT machine with $m-2$ work tapes[1]. We can then characterise V by the finite $SO^{=}_{\mathcal{V}_V}$-theory machine:

$$\mathcal{IT}_V = (\mathcal{ITT}_V, \{0\hat{,}1\}^{\omega}_{\mathcal{X}_{TM_1}}, \{0\hat{,}1\}^{\omega}_{\mathcal{Y}_{TM_m}}),$$

with vocabulary:

$$\mathcal{V}_V = \{<, L, S, I, h\} \cup \{C_1, \ldots, C_m\} \cup \{H_1, \ldots, H_m\} \cup \{0, 1\} \cup \Pi.$$

Where the symbols in $\mathcal{V}_V \setminus \{L\}$ are as they are in Example 5.3.1 with 0 taking the place of the blank symbol $\mathbf{b}$. Whereas L is a unary relation such that $L(x)$ is true iff x is a limit ordinal. The sequences $\mathcal{X}_{TM_1}$ and $\mathcal{Y}_{TM_m}$ are also the same as in Example 5.3.1, describing the cell contents of the first tape at time 0 and the mth tape at time h respectively.

The theory of $\mathcal{ITT}_V$ is then:

$$\mathcal{ITT}_V = UOSA \cup CD^{=}_{\mathcal{V}_V} \cup IT_{(s_0,m)} \cup ET_{\mathcal{S}} \cup HT_{s_1} \cup LC_{s_\lambda}.$$

Where $UOSA$ is the set of uncountable ordinal successor axioms (Definition A.1.16), and $CD^{=}_{\mathcal{V}_V}$ is the set of distinct constant axioms for $\mathcal{V}_V$ (Definition 4.2.6). Whereas the sets, $IT_{s_0,m}$, $ET_{\mathcal{S}}$, and HT_{s_1} are as in Example 5.3.1 for a multi-tape Turing machine with m tapes, blank symbol 0, and rule set $\mathcal{S}$. The definition of the limit relation L together with how the limit configuration is defined is then given in LC_{s_λ}.

As $UOSA \subset \mathcal{ITT}_V$, by Proposition A.1.17, any $SO^{=}_{\mathcal{V}_V}$-model of $\mathcal{ITT}_V$ must be isomorphic to an expansion of an uncountable limit ordinal structure $\langle \mathbb{O}_\delta; <, L, S, 0\rangle$, where $\mathbb{O}_\delta$ is the set of ordinals less than some uncountable limit ordinal δ, and L characterises the set of limit ordinals.

So if $\mathfrak{O} \models_{SO^{=}_{\mathcal{V}_V}} \mathcal{ITT}_V \cup \Phi^{\omega}_{\mathcal{X}_{TM_1}}(w)$ for some $w \in \{0,1\}^\omega$ then since $IT_{(s_0,m)} \subset \mathcal{ITT}_V$, the structure must have the same initial configuration as that of a 3-tape Turing

[1]As in Definition 2.5.4; Π is the set of internal states, $s_0 \in \Pi$ is the initial state, $s_\lambda \in \Pi$ is the limit state, $s_1 \in \Pi$ is the halting state, and $\mathcal{S}$ is a set of m-tape Turing machine rules.

machine. Notably, there is no ordinal below 0, so the sentence $\forall y((y < 0) \rightarrow (C_1(0, y) = \mathbf{b}))$ in $IT_{(s_0,m)}$ is redundant.

Also, $ET_{\mathbb{S}} \subset \mathfrak{ITT}_V$, so by our reasoning in Examples 5.3.1 and 5.3.2, for any time step x if the configuration of $\mathfrak{O}$ at time x is equal to the configuration of V at time x, then the configuration of $\mathfrak{O}$ at time $S(x)$ must be the same as the configuration of V at time $S(x)$.

The configuration at each limit stage is determined by:

$$LC_{s_\lambda} = \left\{ \begin{array}{l} \forall^L x((I(x) = s_\lambda) \wedge \bigwedge_{i=1}^m (H_i(x) = 0)), \\ \forall x \forall y \bigwedge_{i=1}^m ((C_i(x, y) = 0) \vee (C_i(x, y) = 1), \\ \forall^L x \forall y \bigwedge_{i=1}^m ((C_i(x, y) = 0) \leftrightarrow \\ \qquad\qquad (\exists z_1 \forall z_2((z_1 < x) \wedge (z_1 < z_2) \wedge (z_2 < x)) \rightarrow (C_i(z_2, y) = 0))) \end{array} \right\}.$$

The first sentence of LC_{s_λ} ensures that at limit ordinal time steps the internal state is s_λ, and each tape head is placed back at square 0. The second sentence states that each square contains either a 0 or a 1. Combining this with the final sentence then dictates what the contents of each square is at each limit time step. At limit step x, the yth cell of tape i contains a 0 iff there exists a time step z_1 prior to x such that for every step z_2 between z_1 and x the yth cell of tape i contains 0. Otherwise the cell contains a 1. Hence $C_i(x, y) = \limsup_{z \to x} C_i(z, y)$.

So $\mathfrak{O}$ and V have both the same initial configuration, and the configurations of $\mathfrak{O}$ at successor and limit time steps evolve as they do in V. Therefore by transfinite induction, every configuration of $\mathfrak{O}$ at every time step of $\mathfrak{O}$ is the same as it is in V.

Finally by Proposition 2.5.6, every halting ITT machine computation is countable, so if the halting state s_1 is eventually reached in V, it must be reached at some countable ordinal time step. Now as $\mathfrak{O}$ is an expansion of an uncountable limit ordinal structure dom($\mathfrak{O}$) it contains every countable ordinal. So if the halting state s_1 is reached by V then it must occur at some time in $\mathfrak{O}$.

We also have $HT_{s_0} \subset \mathfrak{ITT}_V$, so the ordinal time step at which $\mathfrak{O}$ halts is h. The output is then $\Phi^{\omega}_{\mathfrak{Y}_{TM_m}}(v)$ for some $v \in \{0, 1\}^{\omega}$, where clearly v is exactly what is written on the mth tape of V on input w at the halting time step.

Theorem 7.2.2 *Any decision problem $A \subseteq \{0,1\}^\omega$ or function problem $f :\subseteq \{0,1\}^\omega \to \{0,1\}^\omega$, that is computable by an infinite time Turing machine can be computed by a finite $SO^=$-theory machine.*

Proof: Let V be an infinite time Turing machine that computes the decision problem $f :\subseteq \{0,1\}^\omega \to \{0,1\}^\omega$. By Definition 2.5.5 for any $w \in \{0,1\}^\omega$ we have $V(w) = f(w)$. The finite $SO^=$-theory machine $\mathcal{IT}_V$ in Example 7.2.1 is then able to compute f with each input w encoded as $\Phi^\omega_{\mathcal{X}_{TM_1}}(w)$ and by our reasoning in Example 7.2.1 we have that for any $w \in \{0,1\}^\omega$:

$$\mathcal{ITT}_V(\Phi^\omega_{\mathcal{X}_{TM_1}}(w)) = \Phi^\omega_{\mathcal{Y}_{TM_m}}(f(w)).$$

The fact that $\mathcal{IT}_V$ is indeed a theory machine follows by the same reasoning as in the proof of Theorem 5.1.3, with an ITT machine with input $w \in \{0,1\}^\omega$ giving a model of $\mathcal{ITT}_V \cup \Phi^\omega_{\mathcal{X}_{TM_1}}(w)$ provided that at time steps after h the internal state of the machine is not in Π. Whilst the fact that each of the possible outputs are mutually unsatisfiable follows from Lemma 4.2.9.

Similarly, let W be an infinite time Turing machine that computes the decision problem $A \subseteq \{0,1\}^\omega$. By Definition 2.5.5 for any $w \in \{0,1\}^\omega$ the ITT machine W outputs 10^ω if $w \in A$ and 0^ω otherwise. Now the finite $SO^=$-theory machine $\mathcal{IT}_W$ is not able to compute A as the outputs $\Phi^\omega_{\mathcal{Y}_{TM_m}}(10^\omega)$ and $\Phi^\omega_{\mathcal{Y}_{TM_m}}(0^\omega)$ are not finite. However, if we replace the output set $\{0,\hat{1}\}^\omega_{\mathcal{Y}_{TM_m}}$ of $\mathcal{IT}_W$ with $\{\Theta, \Psi\}$ where:

$$\Theta = \left\{ \begin{array}{l} C_m(h,0) = 1, \\ \forall x((\neg(x = 0) \wedge \forall y(L(y) \to (x < y))) \to (C_m(h,x) = 0) \end{array} \right\},$$

and:

$$\Psi = \{\forall x(\forall y(L(y) \to (x < y)) \to (C_m(h,x) = 0)\},$$

then:

$$\mathcal{IT}'_W = (\mathcal{ITT}_V, \{0,\hat{1}\}^\omega_{\mathcal{X}_{TM_1}}, \{\Theta, \Psi\}),$$

is an $SO^=$-theory machine that is able to compute A. This follows from the above reasoning and the fact that Θ and Φ are both finite sets of sentences. Also in any model of $\mathcal{ITT}_W \cup \Phi^\omega_{\mathcal{X}_{TM_1}}(w))$, the truth of Θ is equivalent to the truth of $\Phi^\omega_{\mathcal{Y}_{TM_m}}(10^\omega) =$

$\{C_m(h, S^i(0)) = 1\} \cup \{C_m(h, S^i(0)) = 0 \mid i \in \mathbb{N} \setminus \{0\}\}$ whereas the truth of Ψ is equivalent to the truth of $\Phi^{\omega}_{\mathfrak{y}_{TM_m}}(0^{\omega}) = \{C_m(h, S^i(0)) = 0 \mid i \in \mathbb{N}\}$, hence the two outputs are mutually unsatisfiable. ❏

Remark 7.2.3 There is a potential problem with how an ITT machine is defined, namely that there is a left-most point and it is not clear what would happen if an ITT machine tried to move past this point. This is particularly problematic as an ITT machine does not appear to have any way of knowing where this left-most point is. Hamkins and Lewis did not discuss this issue in [44], and presumably they did not expect an ITT machine to have a problem when reaching the end of the tape.

One solution is to add a new symbol **L** to the language and place it only at the start point of each tape [75], so if the machine sees **L** it knows not to move further left. However, such a solution can be implemented without adding to the alphabet (and complicating what happens at limit stages). All we have to do is encode 0, 1, and **L** in the language of $\{0,1\}^*$ as 00, 11, and 01 respectively, and have the ITT machine rewrite the symbols on the input tape and place 01 at the beginning of every tape. Hence we do not need modify our description of an ITT machine above to obtain Hamkins and Lewis' results for the computational capabilities of an ITT machine.

Remark 7.2.4 By definition, an ITT machine has no blank tape cells and only two sorts of tape symbols. So without altering the input word the only unambiguous finite words that we can input into an ITT machine are unary words, that is, words from $\{1\}^*$. An input 1^n can then be written on the input tape as $1^n 0^{\omega}$. Though unary encoding is significantly less efficient than binary encoding, through Gödel numberings [27] we may encode any finite word from $\mathcal{A}^*$ as a unary word of $\{1\}^*$.

Theorem 7.2.5 *[44] Every arithmetical relation (Definition 2.1.11) is decidable by an infinite time Turing machine.*

Proof (sketch): To see that any relation $R \in \Sigma^0_1$ is ITT computable, let P be such that $R(\vec{x}) \equiv \exists \vec{y} P(\vec{x}, \vec{y})$. Given any input $\vec{x} \in \prod_{i=1}^{n} \mathcal{A}^*$ ($\vec{x}$ may be inputted into an ITT machine by encoding it as a word of $\mathcal{A}^*$ (as in Remark 2.1.9) and encoding this word as a word of $\{1\}^*$), we can compute the truth of $R(\vec{x})$ with an infinite time Turing machine V_R which computably checks the truth of $P(\vec{x}, \vec{y})$ for each $\vec{y} \in \prod_{i=1}^{n} \mathcal{A}^*$ in turn. If V_R finds a $\vec{y}$ for which $P(\vec{x}, \vec{y})$ holds then V_R halts and accepts. Otherwise V_R keeps computing and eventually reaches the limit time step ω with internal state s_λ, in which case we know that there does not exist a $\vec{y} \in \prod_{i=1}^{n} \mathcal{A}^*$ such that $P(\vec{x}, \vec{y})$ is true, hence $R(\vec{y})$ must be false and so V_R halts and rejects.

In similar manner we can compute any relation in the arithmetical hierarchy of the form $\exists \vec{y}_1 \forall \vec{y}_2 \cdots \exists \vec{y}_{n-1} \forall \vec{y}_n P(\vec{x}, \vec{y}_1, \ldots, \vec{y}_n)$ with an ITT machine which checks the truth of each possible entry of a given relation within α to $\alpha + \omega$ time periods. ❏

Corollary 7.2.6 *There exists a problem which is finite $SO^=$-computable that is not Turing machine computable.*

Proof: By Theorem 7.2.5, for any $R \in \bigcup_{n\in\mathbb{N}}(\Sigma_n^0 \cup \Pi_n^0)$ there exists an ITT machine which is able to compute R. By Theorem 2.1.12 we know that there exist arithmetical relations that are not Turing machine computable, so let P be such a relation.

So since we can encode P as a decision problem, by Theorem 7.2.2 P is finite $SO^=$-computable whilst not being Turing machine computable. ❏

Therefore, like infinite theory machines, finite $SO^=$-theory machines are able to compute a wide variety of finite problems beyond the Turing machine computable problems.

Remark 7.2.7 Infinite time Turing machines are not the only example of a super-Turing system that can be characterised by a finite theory machine. Blum-Shub-Smale machines, which can algebraically compute with real numbers are also capable of such feats. We explained how such computational system can be characterised by a finite theory machine in [75].

When characterising a computational process using a theory machine any encoding of an input should really happen within the theory machine itself. So to characterise this encoding we have the following example in which we have a type-2 machine which encodes words from $\mathcal{A}^*$ into words of $\{0,1\}^\omega$, concatenated with an ITT machine.

Example 7.2.8 Let T_E be a type-2 machine with input alphabet $\mathcal{A}$, input type $*$ and output type ω. Given any finite input word $w \in \mathcal{A}^*$ let T_E output $\tilde{w}0^\omega$, where $\tilde{w}$ is an encoding of w in $\{0,1\}^*$.

Let V_R be an ITT machine such that for some $R \subseteq \prod_{i=1}^n \mathcal{A}^*$ we have that V_R accepts $\tilde{w}0^\omega$ iff $w = \langle w_1, \ldots, w_n \rangle$[1] and $(w_1, \ldots, w_n) \in R$.

As in Example 5.3.2 we can characterise T_E by the finite $SO^{=}_{\mathcal{V}_{T_E}}$-theory machine:

$$\mathcal{T}2\mathcal{M}_{T_E} = (\mathcal{T}2\mathcal{M}\mathcal{T}_{T_E}, \hat{\mathcal{A}}^*_{\mathcal{X}_{TM_1}}, \widehat{\{0,1\}}^\omega_{\mathcal{Y}_{T2}}).$$

Whereas as in Example 7.2.1 we can characterise V_R by the finite $SO^{=}_{\mathcal{V}_{V_R}}$-theory machine:

$$\mathcal{I}\mathcal{T}_{V_R} = (\mathcal{I}\mathcal{T}\mathcal{T}_{V_R}, \widehat{\{0,1\}}^\omega_{\mathcal{X}_{TM_1}}, \widehat{\{0,1\}}^\omega_{\mathcal{Y}_{TM_m}}),$$

We can then describe a machine which takes inputs from $\mathcal{A}^*$ and computes whether $w \in R$ by encoding w into an ITT machine V_R which is able to decide R, via the $(\{0,1\}; \mathcal{Y}_{T2}, \mathcal{X}_{TM_1})$-concatenation $\mathcal{I}\mathcal{T}_{V_R} \circ_{(\{0,1\};\mathcal{Y}_{T2},\mathcal{X}_{TM_1})} \mathcal{T}2\mathcal{M}_{T_E}$.

[1] As in Remark 2.1.9.

Chapter 8

Physical Computation and Complete Theories

So far in this document we have presented various computational systems and explained how such systems can be characterised by theory machines. While some of these computational systems are grounded in what is clearly physically possible (e.g. Turing machines [27, 67], quantum computers [50, 59], and physical computers [16]), other systems instead find utility in computing in ways that are questionably achievable (e.g. Type-2 machines [72] and infinite time Turing machines [44])[1].

The diverse inequivalent nature of these formulations presents the question of what a computation actually is, and if computation can be "unphysical" then where does the boundary between "physical" and "unphysical" computation lie? For example, is it the transfinite aspect of an infinite time Turing machine that makes it "unphysical"? If so, then why is a quantum computer able to "physically" compute with an infinite continuous space?

One resolution to these questions is to invoke the Church-Turing thesis [27, 29, 67], which as we noted in Chapter 1 is often rendered as:

"Every effectively calculable function is computable by a Turing machine,"

[1]Though notably these models have not been proven to be physically impossible, however its not clear what form such a proof would take.

and assume that it applies to any physical process that we compute with.

Now this may seem to be a perfectly natural assumption, as the physical world provides us with the means to compute. So if it provided us with the means to compute a function that was not Turing computable then the Church-Turing thesis would have to be false. However, the Turing machine was designed to mimic and describe how a *person* mathematically computes something [67], rather than how a physical system might go about computing it. Indeed there are many aspects of physics we (humanity as a whole) do not yet understand, so we cannot in good faith disregard computational structures capable of violating the Church-Turing thesis and label them "unphysical" just because we do not *like* what they can do.

That being said, in *Quantum theory, the Church-Turing principle and the universal quantum computer* [29] Deutsch suggested that there is at least sufficient evidence to assert that the Church-Turing thesis applies to physical systems that we are able to construct. He stated the modified thesis:

"Every finitely realizable physical system can be perfectly simulated by a universal model computing machine operating by finite means."

There is some theoretical justification for this thesis. In *Church's thesis and principles for mechanisms* [35] Gandy formulated a concept of what it meant for a problem to be computed by an arbitrary constructible mechanism, proving that a problem is computable by such a device if and only if it is computable by a Turing machine.

However, as well as being highly complex and involving a large number of assumptions, Gandy's formulation focussed on devices that were entirely human-constructible. This is of course rather questionable, as the universe we live in is not human-constructed, physical reality exists without us having to build it. It could therefore be possible that there exist elements of reality that are not human-constructible which provide us with the ability to compute problems that are not Turing computable. For example in [8, 9, 10, 11, 12] Beggs and Tucker described various physical experiments that used the physical word as a computational oracle.

Now as we noted in Section 1.2 and Chapter 4, in *When does a physical system compute?* [48] Horsman et al. argued that in order for a person to be able to compute with a physical system they must possess an abstract representation of the system along with a sufficiently correct theory of how the system behaves. Such a theory must also be testable through measurement of the system. Therefore if there did exist elements of reality capable of computing problems that are not Turing computable, then in order to utilise them a person would also have to know and understand how such elements worked. This knowledge and understanding would of course take the form of the person's representation and theory of the system.

As already noted, we assert that this representation and theory can be expressed in terms of logical sentences. This theory should also be finite as it is should be a theory that can be understood by a human and is derived from a finite number of tests. However as we proved in Corollary 7.2.6, for some logical systems there exist finite theory machines which can characterise computational systems that are capable of violating the Church-Turing thesis. This suggests that such logical systems have unreasonable descriptive power, in that they are able to characterise unphysical computation systems.

Instead we assert that a computational system is physically realisable only if its computational aspects can be characterised by a *finite first-order theory (FFOT) machine*, that is a theory machine which is either a finite FO-theory machine or a finite $FO^=$-theory machine.

In this chapter we will prove that a finite problem may be computed by a FFOT machine if and only if it is computable by a Turing machine (Theorem 8.2.1). Therefore if we assume that the Church-Thesis applies to physical process then FFOT machine characterisable systems may also be physically realisable. Further, if the computational aspects of a system cannot be characterised by a FFOT machine then such a system should not be physically realisable.

We will also prove in this chapter that a general word problem may be computed by a FFOT machine if and only if it is computable by a type-2 machine (Theorem 8.3.3). Thereby providing an explanation of how we can compute with infinite inputs and outputs (such as real numbers) in a manner which is physically justifiable.

The key results in this chapter (Theorem 8.2.1 and Theorem 8.3.3), along with many of the concepts, featured in our paper "Physical Computation and First-Order Logic" [75].

8.0.1 Some Useful Results

As the following result demonstrates, finite FO-computability and finite $FO^{=}$-computability are equivalent, which is why we may collectively refer to finite FO-theory machines and finite $FO^{=}$-theory machines as $FFOT$ machines.

Proposition 8.0.9 *A problem is finite FO-computable if and only if it is $FO^{=}$-computable.*

Proof: ($\Rightarrow$) Let $\mathcal{M}$ be a finite $FO_{\mathcal{V}}$-theory machine. If $\mathcal{V}$ contains the binary relation $=$ as one of its symbols then we may replace $=$ with $=' \notin \mathcal{V}$ in every sentence of $\mathcal{M}$. This gives us a new FO-theory machine $\mathcal{M}'$ with vocabulary $\mathcal{V}' = (\mathcal{V} \cup \{='\}) \setminus \{=\}$, and clearly $\mathcal{M}'$ is able to compute any problem that $\mathcal{M}$ is able compute. Also, $\mathcal{M}'$ can be interpreted as a finite $FO^{=}_{\mathcal{V}}$-theory machine that does not mention the true equality relation, which means that any problem computable by $\mathcal{M}$ is then finite $FO^{=}$-computable.

($\Leftarrow$) Let $\mathcal{N} = (\mathcal{T}, \mathcal{I}, \mathcal{O})$ be a finite $FO^{=}_{\mathcal{U}}$-theory machine that is able to compute a decision problem $A \subseteq \mathcal{A}^a$ or a function problem $f :\subseteq \mathcal{A}^a \to \mathcal{B}^b$. As $\mathcal{T}$ is finite the set of symbols used in $\mathcal{T}$ must be finite. Similarly the set of symbols used in the set of input sets $\hat{\mathcal{A}}^a_{\mathcal{X}}$ and the set of output sets $\hat{\mathcal{B}}^b_{\mathcal{Y}}$ must be finite since the sentences within them are constructed from simple sequences, and $\mathcal{A}$ and $\mathcal{B}$ are finite. Without loss of generality we may then suppose that $\mathcal{U}$ is a finite vocabulary. Therefore $EQ^{=}_{\mathcal{U}}$ is a finite set and so there exists a finite $FO_{\mathcal{U}\cup\{=\}}$-theory machine $\mathcal{N}' = (\mathcal{T} \cup EQ^{=}_{\mathcal{U}}, \mathcal{I}, \mathcal{O})$ which is able to compute any problem that $\mathcal{N}$ is able to compute.

To see why this is the case, for any $\Phi \in \mathcal{I}$, suppose that $\mathcal{N}(\Phi) = \Theta$. If $\mathfrak{M}$ is an $FO^{=}_{\mathcal{U}}$-model of $\mathcal{T} \cup \Phi$, then $\mathfrak{M}$ is also an $FO_{\mathcal{U}\cup\{=\}}$-model of $\mathcal{T} \cup EQ^{=}_{\mathcal{U}} \cup \Phi$, where the equality relation is interpreted as one of the normal relations of the vocabulary, rather than a non-logical symbol of the logical system. We then still have $\mathfrak{M} \models_{FO_{\mathfrak{U}\cup\{=\}}} \Theta$.

For the other direction, let $\mathfrak{N}$ be an $FO_{\mathfrak{U}\cup\{=\}}$-model of $\mathfrak{T} \cup EQ^{=}_{\mathfrak{U}} \cup \Phi$, as noted in Remark A.1.2 it is possible that that there may be two distinct elements $p, q \in \mathrm{dom}(\mathfrak{N})$ where $p = q$ in $\mathfrak{N}$. In which case, by $EQ^{=}_{\mathfrak{U}}$ the function and relation assignments of p and q must all be identical in $\mathfrak{N}$.

There therefore exists an $FO_{\mathfrak{U}\cup\{=\}}$-structure $\mathfrak{N}'$ which elementarily embeds into $\mathfrak{N}$ and for any $p, q \in \mathrm{dom}(\mathfrak{N}')$ we have $p = q$ in $\mathfrak{N}'$ iff p and q are the same element of $\mathrm{dom}(\mathfrak{N}')$. It is then the case that $\mathfrak{N}'$ is an $FO^{=}_{\mathfrak{U}}$-structure, and as $\mathfrak{N}' \models_{FO^{=}_{\mathfrak{U}}} \Theta$ then we must also have that $\mathfrak{N} \models_{FO_{\mathfrak{U}\cup\{=\}}} \Theta$. Therefore $\mathcal{N}'(\Phi) = \Theta$. ❑

Remark 8.0.10 So far in this document we have avoided giving specific examples of theory machines that are also FFOT machines. This is in part because the models of a FFOT machine tend to be far from unique. Indeed, if there exists an infinite model of a FFOT machine, then by the Lowenheim-Skolem theorem [46] there must exist models of any infinite cardinality. This highly variable description of a system would have seemed odd to begin with, however Examples 8.1.1, 8.1.1 and 8.3.1 should demonstrate that this varied description can still give clear and computationally sensible outputs.

When describing Turing machines and type-2 machines in Examples 5.1.1, 5.1.2, 5.3.1, and 5.3.2 we made repeated use of the integer successor axioms ISA [56] (Definition A.1.5). However, these are clearly not first-order axioms, instead to characterise Turing machines and type-2 machines with FFOT machines we will use the following set of first-order sentences.

Definition 8.0.11 The *first-order integer successor axioms* in the vocabulary of

$\{=, <, S\}$, where $=, <$ are binary relations, and S is a unary function, is the set:

$$ISA^- = \left\{ \begin{array}{ll} \mathbb{Z}\text{Ax}(1) & \forall x \forall y (S(x) = S(y)) \rightarrow (x = y), \\ \mathbb{Z}\text{Ax}(2) & \forall x \exists y (S(y) = x), \\ \mathbb{Z}\text{Ax}(3) & \forall x (x < S(x)), \\ \mathbb{Z}\text{Ax}(4) & \forall x \neg \exists y ((x < y) \wedge (y < S(x)), \\ \mathbb{Z}\text{Ax}(5) & \forall x \neg (x < x), \\ \mathbb{Z}\text{Ax}(6) & \forall x \forall y (x < y) \rightarrow \neg (y < x), \\ \mathbb{Z}\text{Ax}(7) & \forall x \forall y \forall z ((x < y) \wedge (y < z)) \rightarrow (x < z) \end{array} \right\}.$$

The following set is also useful.

Definition 8.0.12 The set of *second-order axioms in* ISA is:

$$ISA_{SO} = \left\{ \begin{array}{ll} \mathbb{Z}\text{Ax}(8) & \underline{\forall}^1 Z(((\exists x Z(x)) \wedge (\forall y (Z(y) \leftrightarrow Z(S(y)))) \rightarrow \forall z Z(z)), \\ \mathbb{Z}\text{Ax}(9) & \underline{\exists}^1 N(((\exists x N(x)) \wedge (\forall y (N(y) \rightarrow N(S(y))) \wedge (\exists z \neg N(z))) \end{array} \right\}$$

Note how:

$$ISA^- = ISA \setminus ISA_{SO}.$$

Thus ISA^- is just the usual successor axioms for the ordered integers ISA without its two second-order axioms. This means that the usual ordered structure of the integers $\langle \mathbb{Z}; <, S \rangle$ is also a model of ISA^-. However there also exist other non-standard models of ISA^- that are not isomorphic to $\langle \mathbb{Z}; <, S \rangle$. For example the structure with domain $\{(n, 1) \mid n \in \mathbb{Z}\} \cup \{(n, 2) \mid n \in \mathbb{Z}\}$, such that $S((n, i)) = (n + 1, i)$, and $(n, i) < (m, j)$ if either $n < m$ and $i = j$, or $i = 0$ and $j = 1$.

Remark 8.0.13 Due to Gödel's first incompleteness theorem [27, 37] it is impossible to give a first-order axiomatisation of the ordered integers.

Despite the existence of these non-standard models we still have the following useful result.

Proposition 8.0.14 *Let* $\mathcal{V} = \{<, S, \check{0}\}$ *where* $\check{0}$ *is a constant symbol*[1], *and let* $\mathfrak{M} \models_{FO^=_{\mathcal{V}}} ISA^-$.

[1] $\check{0}$ has been given an accent so as to distinguish it from 0 in $\mathbb{Z}$.

Then the map $\mu : \mathbb{Z} \to \mathfrak{M}$ given by $\mu(n) = S^n(\check{0})$ (where $S^0(\check{0}) = \check{0}$ and $S^{-m}(\check{0}) = q$ iff $S^m(q) = \check{0}$) is a well-defined embedding of $\mathbb{Z}$ onto $\mathfrak{M}$. Also, for any $n \in \mathbb{Z}$ there does not exist a $p \in$ dom($\mathfrak{M}$) such that $\mu(n) < p$ and $p < \mu(n+1)$.

Proof: By definition $\mu(0) = \check{0}$. Further, by the definition of μ we also have $S(\mu(a)) = \mu(a+1)$, since $S(\mu(a)) = S(S^a(\check{0})) = S^{a+1}(\check{0}) = \mu(a+1)$.

For each $n \in \mathbb{Z}$ where $n \geqslant 0$ each value of $S^n(\check{0})$ is distinct. To see this, note that by $\mathbb{Z}$Ax(3) for any p in dom($\mathfrak{M}$) we have $p < S(p)$, as $<$ is also defined by $\mathbb{Z}$Ax(7) to be a transitive relation, by induction we have $S^n(\check{0}) < S^l(\check{0})$ for any $l \in \mathbb{Z}$ such that $n < l$ in $\mathbb{Z}$. As $p \not< p$ by $\mathbb{Z}$Ax(5) for any $p \in$ dom($\mathfrak{M}$), the value of $\mu(n)$ must be distinct for each n.

For each $k \in \mathbb{Z}$ where $k < 0$ the value of $\mu(k)$ must also be well-defined. As clearly $S^{-1}(\check{0})$ is well-defined since by $\mathbb{Z}$Ax(2) an element $q \in$ dom($\mathfrak{M}$) such that $S(q) = \check{0}$ is guaranteed to exist. q is also unique, as if $S(q') = \check{0}$ then by $\mathbb{Z}$Ax(1) we have $q' = q$. By similar reasoning $S^{-2}(\check{0})$ must also be well-defined, and as such reasoning can be repeatedly applied, by induction $\mu(k)$ is well-defined. Also by the same argument as above we have $\mu(k) < \mu(l)$ if $k < l$, hence for every $n \in \mathbb{Z}$ the value of $\mu(n)$ is distinct, and therefore $a = b$ iff $\mu(a) = \mu(b)$.

By the above reasoning we also know that if $a < b$ in $\mathbb{Z}$ then $\mu(a) < \mu(b)$ in $\mathfrak{M}$. Now conversely, if $a \not< b$ in $\mathbb{Z}$ then $b \leqslant a$, and if $a = b$ then by $\mathbb{Z}$Ax(5) $\neg(\mu(a) < \mu(b))$, whereas if $b < a$ then $\mu(b) < \mu(a)$ and by $\mathbb{Z}$Ax(6) $\neg(\mu(a) < \mu(b))$. Consequently μ is an embedding of $\mathbb{Z}$ into $\mathfrak{M}$.

Finally for any $n \in \mathbb{Z}$ there does not exist a $p \in$ dom($\mathfrak{M}$) such that $\mu(n) < p$ and $p < \mu(n+1)$. As suppose that $\mu(n) = q$ then $\mu(n+1) = S(q)$, and so by $\mathbb{Z}$Ax(4) there does not exist a p such that $q < p$ and $p < S(q)$. ❑

Remark 8.0.15 Let $\mathfrak{M}$ be any FO or $FO^=$-model of ISA^-. In this chapter we will refer to the part of $\mathfrak{M}$ that contains 0 and is isomorphic to $\mathbb{Z}$ as the *standard part* of $\mathfrak{M}$.

In this chapter a key property for a logical system will be completeness. Recall from Definition 3.4.5 that a logical system $\mathfrak{LS}$ is complete if there exists an $\mathfrak{LS}$-proof system $\mathfrak{P}$ such that for any sets of $\mathfrak{LS}$-sentences Γ, Δ we have $\Gamma \models_{\mathfrak{LS}} \Delta \iff \Gamma \vdash_{\mathfrak{P}} \Delta$. Which means that Δ is true in any $\mathfrak{LS}$-model of Γ iff there exists a finite $\mathfrak{LS}$-proof of every sentence in Δ from Γ. Gödel's completeness theorem states that first-order logic is complete, and the same is true for first-order logic with equality [34, 36, 45].

An important property of a complete logical system is the following.

Lemma 8.0.16 *Let $\mathfrak{LS}$ be a complete logical system, and Γ, Δ be sets of $\mathfrak{LS}$-sentences. If $\Gamma \models_{\mathfrak{LS}} \Delta$ then for every finite subset $\Omega \subseteq \Delta$ there exists a finite subset $\Upsilon \subseteq \Gamma$ such that $\Upsilon \models_{\mathfrak{LS}} \Omega$.*

Proof: By the definition of complete logical system, if $\Gamma \models_{\mathfrak{LS}} \Delta$ then $\Gamma \vdash_{\mathfrak{P}} \Delta$, and so $\Gamma \vdash_{\mathfrak{P}} \Omega$. Therefore for each $\phi \in \Omega$ there exists a finite $\mathfrak{P}$-proof of ϕ from Γ. Clearly such a proof can only reference a finite number of sentences in Γ, so let $\Upsilon_\phi \subseteq \Gamma$ be the finite set of sentences in Γ that are used to prove $\phi \in \Omega$.

Now let $\Upsilon = \bigcup_{\phi \in \Omega} \Upsilon_\phi$, which is clearly finite, and it must also be the case that exists a $\mathfrak{P}$-proof of every element of Ω from Υ. Hence we have $\Upsilon \vdash_{\mathfrak{P}} \Omega$, and so by the definition of completeness we have that $\Upsilon \models_{\mathfrak{LS}} \Omega$. ❏

8.1 Examples of FFOT Machines

8.1.1 Turing Machines

We may characterise a Turing machine computing a decision problem with a FFOT machine in much the same manner as how we characterised it by a finite $SO^=$-theory machine in Example 5.1.1.

Example 8.1.1 Let $M = (\Lambda, \Pi, \mathbf{b}, \mathcal{A}, s_0, \langle s_a, s_r \rangle, \mathfrak{R})$ be a Turing machine which computes the decision problem $A \subseteq \mathcal{A}^*$.

We can characterise M via the finite $FO^{=}_{\mathcal{V}_M}$-machine:

$$\mathcal{TM}^{-}_{M} = (\mathcal{TMT}^{-}_{M}, \hat{\mathcal{A}}^{*}_{\mathcal{X}_{TM}}, \{\{I(h) = s_a\}, \{I(h) = s_r\}\}),$$

in the same vocabulary, $\mathcal{V}_M$, as $\mathcal{TM}_M$ in Example 5.1.1 and with the same input sequence $\mathcal{X}_{TM}$. The theory of $\mathcal{TM}^{-}_{M}$ is:

$$\mathcal{TMT}^{-}_{M} = \mathcal{TMT}_{M} \setminus ISA_{SO}.$$

Where $\mathcal{TMT}_M$ is the theory of $\mathcal{TM}_M$ in Example 5.1.1[1], which except for $ISA_{SO} \subset ISA$ is a finite first-order theory.

As $ISA^{-} \subset \mathcal{TMT}^{-}_{M}$, for any $w \in \mathcal{A}^{*}$ any $FO^{=}_{\mathcal{V}_M}$-model $\mathfrak{D}$ of $\mathcal{TMT}^{-}_{M} \cup \Phi^{*}_{\mathcal{X}_{TM}}(w)$ must be an expansion of an $FO^{=}_{\{<,S,0\}}$-model $\mathfrak{D}'$ of ISA^{-}.

Clearly the input $\Phi^{*}_{\mathcal{X}_{TM}}(w) = \{C(0, S^{i}(0)) = w_i \mid i \in \{0, \ldots, |w|-1\}\} \cup \{C(0, S^{|w|}(0)) = \mathbf{b}\}$ defines C within the standard part of $\mathfrak{D}$. Similarly $IT_{s_0} \subset \mathcal{TMT}^{-}_{M}$ defines the remainder of the initial configuration of M at time 0 in $\mathfrak{D}$. The rules of $\mathcal{R}$ are then applied by $ET_{\mathcal{R}} \subset \mathcal{TMT}^{-}_{M}$, and so as in Example 5.1.1 at times $S(0), S^{2}(0), S^{3}(0), \ldots$ the evolution of the configurations of M is represented correctly in $\mathfrak{D}$.

As M computes A, it must halt on every input. That is, there must exist a finite time step at which the internal state of M is either s_a or s_r. Hence by $HT_{(s_a,s_r)} \subset \mathcal{TMT}^{-}_{M}$ the halting time step h must occur within the standard part of $\mathfrak{D}$. The truth of the output $I(h) = s_a$ or the output $I(h) = s_r$ in $\mathfrak{D}$ therefore depends entirely on what happens in the standard part of $\mathfrak{D}$.

Crucially, as in Example 5.1.1, all the configurations of M on input w necessarily occur within $\mathfrak{D}$, and necessarily lead to the same output. The output cannot be changed by what happens in the non-standard part of the model, as if it was then this would lead to a logical contradiction. Nor can h occur at a non-standard time step, as constants must have a unique value in any $FO^{=}$ model.

It must therefore be the case that $\mathfrak{D} \models_{FO^{=}_{\mathcal{V}_M}} \{I(h) = s_a\}$ iff M accepts w, and $\mathfrak{D} \models_{FO^{=}_{\mathcal{V}_M}} \{I(h) = s_r\}$ iff M rejects w.

[1]Recall that $\mathcal{TMT}_M = ISA \cup CD^{=}_{\mathcal{V}_M} \cup IT_{s_0} \cup ET_{\mathcal{R}} \cup HT_{(s_a,s_r)}$, where IT_{s_0} describes the initial configuration, $ET_{\mathcal{R}}$ describes the application of the rules of $\mathcal{R}$, and $HT_{(s_a,s_r)}$ describes how M halts.

Hence we have that $\mathcal{TM}^-_M$ is able to compute A, even though $\mathcal{TM}^-_M$ has models which are highly non-standard.

Remark 8.1.2 Unlike $\mathcal{TM}_M$ there are of course models of $\mathcal{TM}^-_M$ which have non-standard elements, and these elements are not present in the definition of a Turing machine. However, these non-standard elements are transfinite and so in the real world they are unreachable and unobservable. So whilst such a model of $\mathcal{TM}^-_M$ may not correspond to a Turing machine M in the manner we typically view it, there is no way of us knowing for certain that a real world actualisation of M does not have these transfinite elements an infinite distance/time away from our position. Also, the non-standard parts of these models do not have any effect on the proper computation. We therefore argue that $\mathcal{TM}^-_M$ characterises the Turing machine M.

In a similar manner to Example 5.1.2 we may also characterise a Turing machine computing a function problem as a FFOT machine.

Example 8.1.3 Let $M' = (\Lambda, \Pi, \mathbf{b}, \mathcal{A}, s_0, \langle s_1 \rangle, \mathcal{R})$ be a Turing machine which computes the function problem $f :\subseteq \mathcal{A}^* \to \mathcal{B}^*$.

We can characterise M' via the finite $FO^=_{\mathcal{V}_{M'}}$-machine:

$$\mathcal{TM}^-_{M'} = (\mathcal{TMT}^-_{M'}, \hat{\mathcal{A}}^*_{\mathcal{X}_{TM}}, \hat{\mathcal{B}}^*_{\mathcal{Y}_{TM}}),$$

in the same vocabulary, $\mathcal{V}_{M'}$, and with the same input and output sequences, respectively $\mathcal{X}_{TM}$ and $\mathcal{Y}_{TM}$, as $\mathcal{TM}_{M'}$ in Example 5.1.2. The theory of $\mathcal{TM}^-_{M'}$ is then:

$$\mathcal{TMT}^-_{M'} = \mathcal{TMT}_{M'} \setminus ISA_{SO}.$$

Where $\mathcal{TMT}_{M'}$ is the theory of $\mathcal{TM}_{M'}$ in Example 5.1.2.

As in Example 8.1.1 above we have $ISA^- \subset \mathcal{TMT}^-_{M'}$. So for any $w \in \mathcal{A}^*$ any $FO^=_{\mathcal{V}_{M'}}$-model $\mathfrak{E}$ of $\mathcal{TMT}^-_{M'} \cup \Phi^*_{\mathcal{X}_{TM}}(w)$ contains a standard part which is isomorphic to $\mathbb{Z}$. Similarly to in Example 8.1.1 it must then follow that the entire computation of M' is described within the standard part of $\mathbf{E}$, and if M' halts then it must do so within the standard part of $\mathbf{E}$.

Consequently if $w \in \operatorname{dom}(f)$ then M' on input w halts and outputs $f(w)$, and so by our reasoning in Example 5.1.2 we have $\mathfrak{E} \models_{FO^{=}_{\mathcal{V}_{M'}}} \Phi^*_{\mathcal{Y}_{TM}}(f(w))$. Whereas if $w \notin \operatorname{dom}(f)$ then M' on input w never halts, hence $I(p) \neq s_1$ for all $p > 0$ in the standard part of $\mathfrak{E}$. Therefore, as in Example 5.1.2, any output $\Phi^*_{\mathcal{Y}_{TM}}(v) \in \hat{\mathcal{B}}^*_{\mathcal{Y}_{TM}}$ can be true in some model of $\mathcal{TMT}^{-}_{M'} \cup \Phi^*_{\mathcal{X}_{TM}}(w)$, and consequently $\mathcal{TM}^{-}_{M'}(\Phi^*_{\mathcal{X}_{TM}}(w))$ is undefined.

Therefore $\mathcal{TM}^{-}_{M'}$ is able to compute $f :\subseteq \mathcal{A}^* \to \mathcal{B}^*$.

Remark 8.1.4 By Proposition 8.0.9 both of the finite $FO^{=}$-theory machines $\mathcal{TM}^{-}_{M}$ and $\mathcal{TM}^{-}_{M'}$ above can be converted into finite FO-theory machines, and so every Turing machine computable problem is finite FO-computable.

8.1.2 Differential Equation Systems

In Subsection 5.2.1 we discussed how we may characterise a physical system $\mathcal{S}$ that obeys a set of differential equations with an $FO\mathbb{R}$-theory machine $\mathcal{M}_{\mathcal{S}}$. We can also characterise such a system with an $SO^{=}_{\mathcal{V}}$-theory machine $\mathcal{M}'_{\mathcal{S}}$ in which the non-logical symbols of $FO\mathbb{R}$ are included as part of the vocabulary $\mathcal{V}$. All we have to do is add the real arithmetic axioms RAA (Definition A.1.7) to the theory of $\mathcal{M}'_{\mathcal{S}}$. It then follows from Proposition A.1.8 that any model of $\mathcal{M}'_{\mathcal{S}}$ must be isomorphic to an expansion of the usual structure of real arithmetic $\langle \mathbb{R}; <, +, \times, 0, 1\rangle$, and so such a model is also an $FO\mathbb{R}$-structure. However, the real arithmetic axioms are a second-order set of sentences, so to characterise $\mathcal{S}$ with a FFOT machine we use the following set.

Definition 8.1.5 [28] The set of first-order *dense ordered field axioms* in the vocabulary of $\mathcal{V}_{\mathbb{R}} = \{=, <, +, \times, 0, 1\}$, where $=, <$ are binary relations, $+, \times$ are binary functions, and 0,1 are constants, is:

$$DOF = RAA \setminus \left\{ \begin{array}{l} \underline{\forall}^1 Q((\exists u Q(u) \wedge \exists v \forall w (Q(w) \to (w < v))) \\ \qquad \to \exists x \forall y (\forall z (Q(z) \to (z < y)) \leftrightarrow ((x < y) \vee (x = y)))) \end{array} \right\}.$$

Much like with ISA^- and ISA, the models of DOF include the usual structure of real arithmetic $\langle \mathbb{R}; <, +, \times, 0, 1\rangle$ as well as other structures such as the usual structure of rational arithmetic $\langle \mathbb{Q}; <, +, \times, 0, 1\rangle$. Indeed there exists an embedding from the usual structure of rational arithmetic into any model of DOF [28].

We may then characterise a differential equation system $\mathcal{S}$ (such as MONIAC [14] or Blakey's double slit factoriser [17]) with an $FO^=$-theory machine $\mathcal{M}_{\mathcal{S}}^-$, where $\mathcal{M}_{\mathcal{S}}^-$ is the same as $\mathcal{M}_{\mathcal{S}}'$ above but with DOF included in the theory of $\mathcal{M}_{\mathcal{S}}^-$ instead of RAA.

Though there do exist $FO^=$-models of $\mathcal{M}_{\mathcal{S}}^-$ that are not expansions of the usual structure of real arithmetic, the necessary aspects of real arithmetic must hold in any such model. That is, any model of DOF must be closed under addition, multiplication, have non-zero multiplicative inverses, and contain a dense total order. Further, if a real number r is defined by the theory of $\mathcal{M}_{\mathcal{S}}^-$, then any model of $\mathcal{M}_{\mathcal{S}}^-$ must contain r whilst still being closed under addition, multiplication etc.

Crucially if a function f is defined in $\mathcal{M}_{\mathcal{S}}^-$, then its partial derivative $\partial_1 f$ can still be defined within the theory of $\mathcal{M}_{\mathcal{S}}^-$ in the exact same manner as in $\mathcal{M}_{\mathcal{S}}$ (see Equation 5.1 in Subsection 5.2.1). By definition, the value of $\partial_1 f(\vec{a})$ will then be equal to the usual value of $\frac{\partial f}{\partial x}$ at point $\vec{a}$, or the two values will be infinitesimally close to one another. That is, the distance between them is less than ϵ for all $\epsilon > 0$[1].

Therefore functions described in $\mathcal{M}_{\mathcal{S}}^-$ will either be equal to, or infinitesimally close to their values in $\mathcal{M}_{\mathcal{S}}$. So the same output will be obtained provided that for any input each of the possible outputs of $\mathcal{M}_{\mathcal{S}}^-$ is bounded away from one another. For example, if $\mathcal{O}_{\mathcal{S}} = \{\{f(\tau_1) < 0\}, \{1 < f(\tau_1)\}\}$ then the two possible outputs are of distance 1 away from one another.

[1] Which is possible in a non-standard model of DOF, such as the hyperreals [61].

8.2 FFOT Machines and Turing Computability

In the previous section we saw how various examples of computational systems may be characterised with FFOT machines. A natural question to ask is whether there exist computational systems that can be characterised by FFOT machines which are able to compute problems that are not Turing machine computable. The following theorem demonstrates that the answer to this question is "no".

Theorem 8.2.1 *A finite problem is Turing machine computable if and only if it is finite FO-computable.*

We split up the proof of Theorem 8.2.1 into two Lemmas.

Lemma 8.2.2 *A finite decision problem $A \subseteq \mathcal{A}^*$ is Turing machine computable if and only if it is finite FO-computable.*

Proof: ($\Rightarrow$) This follows from Example 8.1.1 and Remark 8.1.4.

($\Leftarrow$) Conversely, suppose that we are able to compute $A \subseteq \mathcal{A}^*$ with the finite $FO_{\mathcal{V}}$-theory machine $\mathcal{M} = (\mathcal{T}, \mathcal{I}, \mathcal{O})$ with input set $\hat{\mathcal{A}}^*_{\mathcal{X}} \subset \mathcal{I}$ and outputs $\Theta, \Psi \in \mathcal{O}$ corresponding to accepting and rejecting respectively.

By definition, for any $w \in \mathcal{A}^*$ if $w \in A$ we have $\mathcal{T} \cup \Phi^*_{\mathcal{X}}(w) \models_{FO_{\mathcal{V}}} \Theta$. Hence by the completeness of first-order logic (Theorem 3.4.6)[36, 45] we have $\mathcal{T} \cup \Phi^*_{\mathcal{X}}(w) \vdash_{\mathfrak{P}} \Theta$ for some FO-proof system $\mathfrak{P}$, and there exists a finite $\mathfrak{P}$-proof of the truth of Θ given $\mathcal{T} \cup \Phi^*_{\mathcal{X}}(w)$. Conversely if $w \notin A$ we have $\mathcal{T} \cup \Phi^*_{\mathcal{X}}(w) \models_{FO_{\mathcal{V}}} \Psi$ and there exists a finite formal proof of the truth of Ψ given $\mathcal{T} \cup \Phi^*_{\mathcal{X}}(w)$.

As $\mathcal{T} \cup \Phi^*_{\mathcal{X}}(w)$ is a finite set of $FO_{\mathcal{V}}$-sentences the set of all first-order sentences provable from it is computably enumerable. We can therefore construct a Turing machine $M_{\mathcal{M}}$ that, on an input of w, enumerates all sentences provable from $\mathcal{T} \cup \Phi^*_{\mathcal{X}}(w)$.

By the definition of a theory machine, only one of $\Theta, \Psi \in \mathcal{O}$ will be provable from $\mathcal{T} \cup \Phi^*_{\mathcal{X}}(w)$. Further, Θ and Ψ must also be finite, so if $w \in A$ then $M_{\mathcal{M}}$ must eventually enumerate every element of Θ, and will never enumerate every single element of Ψ. Conversely if $w \notin A$ then $M_{\mathcal{M}}$ must eventually enumerate every element of Ψ, and will never enumerate every single element of Θ.

We can then define $M_{\mathcal{M}}$ to halt and accept if it enumerates every element of Θ, and halt and reject if it enumerates every element of Ψ. The Turing machine $M_{\mathcal{M}}$ then computes $A \subseteq \mathcal{A}^*$ and thus A is Turing machine computable. ❑

Lemma 8.2.3 *A finite function problem $f :\subseteq \mathcal{A}^* \to \mathcal{B}^*$ is Turing machine computable if and only if it is finite FO-computable.*

Proof: ($\Rightarrow$) This follows from Example 8.1.3 and Remark 8.1.4.

($\Leftarrow$) Conversely, suppose that we are able to compute $f :\subseteq \mathcal{A}^* \to \mathcal{B}^*$ with the finite $FO_{\mathcal{V}}$-theory machine $\mathcal{M}' = (\mathcal{T}, \mathcal{I}, \mathcal{O})$ with input set $\hat{\mathcal{A}}^*_{\mathcal{X}} \subset \mathcal{I}$ and output set $\hat{\mathcal{B}}^*_{\mathcal{Y}} \subset \mathcal{O}$.

By definition, for each $w \in \text{dom}(f)$ we have $\mathcal{T} \cup \Phi^*_{\mathcal{X}}(w) \models_{FO_{\mathcal{V}}} \Phi^*_{\mathcal{Y}}(f(w))$ and for any $\Psi \in \hat{\mathcal{B}}^*_{\mathcal{Y}} \setminus \{\Phi^*_{\mathcal{Y}}(f(w))\}$ we have $\mathcal{T} \cup \Phi^*_{\mathcal{X}}(w) \not\models_{FO_{\mathcal{V}}} \Psi$. Hence by the completeness of first-order logic [36, 45] we have $\mathcal{T} \cup \Phi^*_{\mathcal{X}}(w) \vdash_{\mathfrak{P}} \Phi^*_{\mathcal{Y}}(f(w))$ for some FO-proof system $\mathfrak{P}$, and there exists a finite $\mathfrak{P}$-proof of the truth of $\Phi^*_{\mathcal{Y}}(f(w))$ given $\mathcal{T} \cup \Phi^*_{\mathcal{X}}(w)$.

As $\mathcal{T} \cup \Phi^*_{\mathcal{X}}(w)$ is a finite set of $FO_{\mathcal{V}}$-sentences the set of all first-order sentences provable from it is computably enumerable. Similarly as $\mathcal{Y}$ is a simple sequence, the sentences of $\hat{\mathcal{B}}^*_{\mathcal{Y}}$ are computably enumerable. We can therefore construct a Turing machine $M_{\mathcal{M}'}$ that, on an input of w, enumerates all sentences provable from $\mathcal{T} \cup \Phi^*_{\mathcal{X}}(w)$, while concurrently enumerating the elements of $\hat{\mathcal{B}}^*_{\mathcal{Y}}$.

By our reasoning above we know that $\Phi^*_{\mathcal{Y}}(f(w))$ is the only set of sentences $\hat{\mathcal{B}}^*_{\mathcal{Y}}$ that can be entirely proven from $\mathcal{T} \cup \Phi^*_{\mathcal{X}}(w)$. So by repeatedly checking whether every sentence in each enumerated element of $\hat{\mathcal{B}}^*_{\mathcal{Y}}$ have been proven, $M_{\mathcal{M}'}$ will eventually find $\Phi^*_{\mathcal{Y}}(f(w))$. The output word $f(w)$ can then be extracted from the set of sentences $\Phi^*_{\mathcal{Y}}(f(w))$.

If $w \notin \text{dom}(f)$ then for no element $\Psi \in \hat{\mathcal{B}}^*_{\mathcal{Y}}$ is $\mathcal{T} \cup \Phi^*_{\mathcal{X}}(w) \models_{FO_{\mathcal{V}}} \Psi$ true. which means that no set in $\hat{\mathcal{B}}^*_{\mathcal{Y}}$ is provable from $\mathcal{T} \cup \Phi^*_{\mathcal{X}}(w)$, and so $M_{\mathcal{M}'}$ on input w will never halt.

Consequently $M_{\mathbb{M}'}$ is able to compute f, and so f is Turing machine computable.

❏

By the above result, if the Church-Turing thesis is true and applies to physical calculations, then it must be the case that the computational aspects of any obtainable physical system is describable by a finite first-order theory machine. Hence the Church-Turing thesis can be reformulated as:

Every effectively calculable function is computable by a finite first-order theory machine (8.1)

This modified thesis fits in with what Kripke discussed in *The Church-Turing Thesis as a Special Corollary of Gödels Completeness Theorem* [53]. Kripke asserted that: "A computation is just another mathematical deduction, albeit one of a very specialized form." He also argued in favour of what he called "Hilbert's thesis" that the steps of any mathematical argument can be given in first-order logic (with equality). The Church-Turing thesis is then a consequence of Gödel's completeness theorem as for any mathematical argument there must exist a Turing machine which can carry it out.

Now we do not quite agree that a computation is necessarily a mathematical deduction, as we of course assert that an arbitrary computation can be characterised by a theory machine in some logic $\mathfrak{LS}$. Such a computation cannot necessarily be described as a deduction as $\mathfrak{LS}$ may not be complete, meaning the truth of some results does not necessarily follow from a finite proof. However, given Theorem 8.2.1 above (as well as Theorem 8.2.4 below) along with our assertions about physical computation, we do agree that any *physically realisable* computation can be described by a mathematical deduction.

8.2.1 Complete Logical Systems and Turing Computability

The above result is not in fact limited to first-order logic or first-order logic with equality. As the proof of Theorem 8.2.1 relies on just two facts; the fact that we can characterise any Turing machine by a finite first-order theory machine, and the fact that first-order logic (with equality) is complete.

Neither of these properties are held by only first-order logic and first-order logic with equality. We therefore have the following generalisation.

Theorem 8.2.4 *If $\mathfrak{LS}$ is a complete logical system and any Turing machine can be characterised by a finite $\mathfrak{LS}$-theory machine then the class of finite problems that are finite $\mathfrak{LS}$-computable is equal to the class of Turing machine computable problems.*

Proof: By definition, if a finite $\mathfrak{LS}$-theory machine characterises a Turing machine M then it is able to compute the finite problem that is computed by M. Therefore the class of Turing machine computable problems is contained within the class of finite problems that are $\mathfrak{LS}$-computable. Equality follows from $\mathfrak{LS}$ being a complete logical system, as we can just replace every instance of FO in the proofs of Lemmas 8.2.2 and 8.2.3 with $\mathfrak{LS}$ to obtain a Turing machine that is able to compute each finite $\mathfrak{LS}$-computable finite problem. ❑

Therefore if we can finitely describe the computational aspects of a system $\mathcal{P}$ using a logical system such as $\mathfrak{LS}$ above then everything computable by $\mathcal{P}$ is Turing machine computable. Conversely, if we can use $\mathcal{P}$ to compute finite problems that are not Turing machine computable, then $\mathcal{P}$ is not characterisable by a finite $\mathfrak{LS}$-theory machine.

Corollary 8.2.5 *Let $\mathfrak{LS}$ be as in Theorem 8.2.4. The infinite theory machines defined in the proof of Proposition 6.0.4 that are capable of computing any finite problem are not in general characterisable by a FFOT machine or a finite $\mathfrak{LS}$-theory machine.*

8.3 FFOT Machines and Type-2 Computability

As we shall see, FFOT machines are not limited to computing finite problems, they are perfectly capable of computing with infinite inputs and outputs. Indeed we may characterise a Type-2 machine by a FFOT machine in a similar manner to how we characterised it with a finite $SO^{=}$-theory machine in Example 5.3.2.

Example 8.3.1 Let $T = (\Lambda, \Pi, \mathbf{b}, \mathcal{A}, s_0, s_1, \mathcal{U}, a, b)$ be a type-2 machine with m tapes that computes the function $f :\subseteq \mathcal{A}^a \to \mathcal{B}^b$. We can characterise T via the $FO^{=}_{\mathcal{V}_T}$-theory machine:

$$\mathcal{T}2\mathcal{M}^{-}_{T} = (\mathcal{T}2\mathcal{M}\mathcal{T}^{-}_{T}, \hat{\mathcal{A}}^{a}_{\mathcal{X}_{TM_1}}, \hat{\mathcal{B}}^{b}_{\mathcal{Y}_{T2}}),$$

in the same vocabulary $\mathcal{V}_T$, and with the same input and output sequences, $\mathcal{X}_{TM_1}$ and $\mathcal{Y}_{T2}$, as $\mathcal{T}2\mathcal{M}_T$ in Example 5.3.2. The only difference is the theory of $\mathcal{T}2\mathcal{M}^{-}_{T}$, which is:

$$\mathcal{T}2\mathcal{M}\mathcal{T}^{-}_{T} = (\mathcal{T}2\mathcal{M}\mathcal{T}_T \setminus ISA_{SO}) \cup EQ_{\mathcal{V}_T}.$$

As in Example 8.1.3 since $ISA \subset \mathcal{T}2\mathcal{M}\mathcal{T}^{-}_{T}$, every model $\mathfrak{F}$ of $\mathcal{T}2\mathcal{M}\mathcal{T}^{-}_{T} \cup \Phi^{a}_{\mathcal{X}_{TM_1}}$ has a standard part which is isomorphic to the usual ordered structure of the integers. In addition, even though the input set may be infinite, as each element of $\Phi^{a}_{\mathcal{X}_{TM_1}}$ is of the form $\{C_1(0, S^i(0)) = w_i\}$, the input set defines the values of $C_1(0, y)$ only within the standard part of $\mathfrak{F}$.

Following the same reasoning as in Example 5.3.2 together with Example 8.1.1 we can see that the computation of T on input w is entirely described within the standard part of $\mathfrak{F}$.

As before in Example 5.3.2, there are two possible cases.

Case 1. $b = *$, in which case if $w \in \mathrm{dom}(f)$ then the halting state s_1 should eventually be reached at time h and the output word $\{O(S^i(0)) = f(w)_i \mid i \in \{0, \ldots, |f(w)| - 1\}\} \cup \{O(S^{|f(w)|}(0)) = \mathbf{b}\}$ is defined within the standard part of $\mathfrak{F}$ (that is, if $O(p)$ is described by the output then p is in the standard part of $\mathfrak{F}$) via $\{\forall y(O(y) = C_m(h, y))\} \subset \mathcal{T}2\mathcal{M}\mathcal{T}^{-}_{T}$. Whereas if $w \notin \mathrm{dom}(f)$ then as in Example 5.3.2 the values of $O(y)$ could be anything in $\mathfrak{F}$, and hence $\mathcal{T}2\mathcal{M}^{-}_{T}(\Phi^{a}_{\mathcal{X}_{TM_1}})$ is undefined.

Case 2. $b = \omega$ in which case, if $w \in \mathrm{dom}(f)$ then for each $n \in \mathbb{N}$ there exists an $l \in \mathbb{N}$ such that $C_m(S^l(0), S^n(0)) \neq \mathbf{b}$. Also, $\{\forall x \forall y(\neg(C_m(x,y) = \mathbf{b}) \to (O(y) = C_m(x,y))\} \subset \mathcal{T}2\mathcal{MT}_T^-$, and the output word $\{O(S^i(0)) = f(w)_i \mid i \in \mathbb{N}\}$ is therefore defined in $\mathfrak{F}$. Notably, the value of $O(q)$ depends on the non-blank values of $C_m(p,q)$ for all $p \in \mathrm{dom}(\mathfrak{F})$, not just the p that are in the standard part of $\mathfrak{F}$. However this cannot prevent the output from being defined, as if $C_m(t,q)$ is not blank at any time step $t \in \mathrm{dom}(\mathfrak{F})$ then we must have $C_m(t,q) = C_m(p,q)$ to prevent a logical contradiction within the model.

Alternatively, if $w \notin \mathrm{dom}(f)$, then as in Example 5.3.2 the value of $O(S^k(0))$ is eventually unspecified for some large enough $k \in \mathbb{N}$. Hence there are multiple possible outputs that could be true in $\mathfrak{F}$ and $\mathcal{T}2\mathcal{M}_T^-(\Phi^a_{\mathfrak{X}_{TM_1}})$ is undefined.

Therefore $\mathcal{T}2\mathcal{M}_T^-$ is able to compute $f :\subseteq \mathcal{A}^* \to \mathcal{B}^*$.

Remark 8.3.2 As in Remark 8.1.4, it follows from Proposition 8.0.9 that we can also characterise any type-2 machine by a finite FO-theory machine. Therefore any type-2 computable problem is finite FO-computable.

Do there exist computational systems that can be characterised by FFOT machines which are able to compute problems that are not Turing machine computable? The following Theorem 8.2.1 implies that the answer to this question is, again, “no”.

Theorem 8.3.3 *Let $a, b \in \{*, \omega\}$. A word function problem $g :\subseteq \mathcal{A}^a \to \mathcal{B}^b$ is computable by a type-2 machine if and only if g is finite FO-computable.*

Proof: ($\Rightarrow$) This follows Remark 8.3.2.

($\Leftarrow$) Suppose that $g :\subseteq \mathcal{A}^a \to \mathcal{B}^b$ is computable by some finite $FO_{\mathcal{V}}$-theory machine $\mathcal{M} = (\mathcal{T}, \mathcal{I}, \mathcal{O})$ with input set $\hat{\mathcal{A}}^a_{\mathfrak{X}} \subset \mathcal{I}$ and output set $\hat{\mathcal{B}}^b_{\mathfrak{Y}} \subset \mathcal{O}$.

As in the proof of Lemma 8.2.3 for every $w \in \mathrm{dom}(g)$ we have $\mathcal{T} \cup \Phi^a_{\mathfrak{X}}(w) \models_{FO_{\mathcal{V}}} \Phi^b_{\mathfrak{Y}}(g(w))$. By the completeness of first-order logic [36, 45] (Theorem 3.4.6) and Lemma 8.0.16, for any finite subset $\Omega \subset \Phi^b_{\mathfrak{Y}}(g(w))$, there exists a finite subset

$\Upsilon \subset \Phi^a_{\mathcal{X}}(w)$, such that $\mathcal{T} \cup \Upsilon \models_{FO_{\mathcal{V}}} \Omega$. Hence again by the completeness of first-order logic there must exist a finite formal proof of the truth of Ω given $\mathcal{T} \cup \Upsilon$.

Thus we can construct a type-2 machine $T_{\mathcal{M}}$, that on input $w \in \mathcal{A}^a$ enumerates the elements of $\Phi^a_{\mathcal{X}}(w)$, and from this enumeration, $T_{\mathcal{M}}$ enumerates all sentences provable from $\mathcal{T} \cup \Phi^a_{\mathcal{X}}(w)$. Let $T_{\mathcal{M}}$ then record each provable sentence of the form $\upsilon_i = d$, for $\mathcal{Y} = \{\upsilon_i\}_{i\in\mathbb{N}}$ and $d \in \mathcal{B}$. Through such sentences $T_{\mathcal{M}}$ can clearly obtain $g(w)$ from which it may then sequentially output $g(w)$.

If $w \notin \text{dom}(g)$ then there is some $l \in \mathbb{N}$ such that $\upsilon_l = e$ is not provable for any $e \in \mathcal{B}$. Therefore $T_{\mathcal{M}}$ is unable to record the lth symbol on the output tape, and hence $T_{\mathcal{M}}(w)$ is similarly undefined. ❏

In the original Church Turing thesis an effectively calculable function was intended to be finite [27, 67]. However if we were to assume that it could also be an infinite function, then Theorem 8.3.3 and our reformulation of the Church Turing thesis (8.1) collectively imply that every effectively calculable function (whether its infinite or not) is computable by a type-2 machine.

This does in fact fit in with the usual view of computability for functions from $\mathcal{A}^\omega$ to $\mathcal{B}^\omega$, in that such a function is computable iff it is type-2 computable [71, 72]. This view is justified by the fact that a type-2 machine computes an infinite word problem in a Turing machine-like manner, and the fact that we can stop a type-2 machine computation at any point and know that whatever has so far been outputted is a prefix of the output word.

Our result provides an alternative justification of this view; a FFOT machine that is able to compute an infinite function problem $f :\subseteq \mathcal{A}^\omega \to \mathcal{B}^\omega$ is just a FFOT machine which is able to admit the infinite word sets of $\hat{\mathcal{A}}^\omega_{\mathcal{X}}$ and $\hat{\mathcal{B}}^\omega_{\mathcal{Y}}$ as inputs and outputs. So infinite problems that are computable in such a manner provide a natural infinite extension to finite FFOT machine computable problems. Now as these finite problems are Turing computable and the infinite problems are type-2 computable, type-2 computability naturally extends Turing computability .

Another consequence of Theorem 8.3.3 is that if a computational system is able to compute problems that are not type-2 computable, then such a system is not characterisable by a FFOT machine.

Corollary 8.3.4 *Infinite time Turing machines are not in general describable by a FFOT machine.*

Corollary 8.3.5 *Blum-Shub-Smale machines are not in general describable by a FFOT machine.*

8.3.1 Complete Logical Systems and Type-2 Computability

Like in Subsection 8.2.1 our result is not limited to first-order logic or first-order logic with equality. We instead have the following generalisation of Theorem 8.3.3.

Theorem 8.3.6 *If $\mathfrak{LS}$ is a complete logical system and any type-2 machine can be characterised by a finite $\mathfrak{LS}$-theory machine then the class of function problems that are finite $\mathfrak{LS}$-computable is equal to the class of type-2 computable function problems.*

Proof: If a finite $\mathfrak{LS}$-theory machine characterises a type-2 machine T then it is able to compute the function problem that is computed by T, and so the class of type-2 computable problems is contained within the class of $\mathfrak{LS}$-computable function problems. Equality then follows from $\mathfrak{LS}$ being a complete logical system, as we can just replace every instance of FO in the proof of Theorem 8.3.3 with $\mathfrak{LS}$ to obtain a type-2 machine that is able to compute each finite $\mathfrak{LS}$-computable word problem. ❑

We can therefore extend the Church-Turing thesis even further and assert that a computational system with finite or infinite inputs and outputs (such as real numbers) is physically realisable only if it is characterisable by a FFOT machine. We also have the following corollary.

Corollary 8.3.7 *Let $\mathfrak{LS}$ be a complete logical system that contains first-order logic. If the word function problems $f :\subseteq \mathcal{A}^a \to \mathcal{B}^b$ and $g :\subseteq \mathcal{B}^b \to \mathcal{C}^c$ where $a, b, c \in \{*, \omega\}$ are finite $\mathfrak{LS}$-computable and $b = *$ or $c = \omega$, then $g \circ f :\subseteq \mathcal{A}^a \to \mathcal{C}^*$ is finite $\mathfrak{LS}$-computable.*

Otherwise if $b = \omega$ and $c = *$ then for both $a = *$ and $a = \omega$, there exists a finite $\mathfrak{LS}$-computable function $f :\subseteq \mathcal{A}^a \to \mathcal{B}^\omega$ and a finite $\mathfrak{LS}$-computable function $g :\subseteq \mathcal{B}^\omega \to \mathcal{C}^*$ such that $g \circ f :\subseteq \mathcal{A}^a \to \mathcal{C}^*$ is not finite $\mathfrak{LS}$-computable.

Proof: From Theorem 8.3.6, the functions f, g, and $g \circ f$ are finite $\mathfrak{LS}$-computable if and only if they are type-2 computable. Therefore the result follows by Proposition. 2.3.4 ❏

Chapter 9

Theory Machine Complexity

"The field, created as it was to cater primarily for Turing-like models of computation, fails to capture the true complexity of many non-standard (analogue, chemical, quantum, etc.) computers." - Ed Blakey

In the 1960's Cobham and Edmonds [26, 30] asserted that a computational problem is feasibly computable if and only if it can be decided in polynomial time on a Turing machine (and thereby lies in $\mathcal{P}$). Though not within its originally intended scope, it has been suggested [70] that Cobham and Edmonds assertion should also apply to what is feasibly computable by *any* physical system. However, this idea has since been challenged by results from quantum computation [59], such as Shor's factorisation algorithm [63], which suggest that the class of problems decidable by a quantum computer in polynomial time ($\mathcal{BQP}$) may include problems that do not lie in $\mathcal{P}$. These results lead naturally to the questions of what it is about quantum systems that makes them capable of feasibly deciding problems that may lie outside of $\mathcal{P}$, and whether there exist other physical systems with such capabilities.

Now as we noted in Subsection 1.1.1, in [5] Baumeler and Wolf looked into the computational power of polynomially bounded circuits acting within closed time-like curves of polynomial length. They asserted that a computation may occur on such a circuit if it is logically consistent and unique, demonstrating that with these assumptions the computational power of these non-causal circuits is equal to

$\mathcal{UP} \cap$ co-$\mathcal{UP}$. Notably, $\mathcal{BQP}$ problems such as the factorisation problem also lie in $\mathcal{UP} \cap$ co-$\mathcal{UP}$, suggesting that there may be a non-causal aspect to the quantum computational speed-up.

Baumeler and Wolf's innovative non-causal circuit model did not have the goal of describing the feasible computational aspects of a general physical system. However, as we saw in Chapter 8, our concept of a FFOT machine appears to be able to characterise the computational aspects of an arbitrary physical system. So in this chapter we shall develop the concept of theory machine complexity, with the goal that the complexity of a physical computation may be understood through the complexity of a corresponding FFOT machine computation.

Notably, rather than describing each computation as a discrete ordered sequence of structures, in a theory machine the whole computation is described via a single consistent structure. Hence any temporal evolution of the machine is described within this structure. This inclusion of the evolution within the structure allows a theory machine to compute in a consistent non-causal and atemporal manner.

Further, in [16] Blakey argued that in order to measure the complexity of unconventional computation devices require unconventional notions of complexity. So the general resource usage of a computation, such as its space, energy, and precision usage (as well as its time usage) should be taken into account when considering the complexity of a problem.

From the computability standpoint, by Theorem 8.2.1 finite-input FFOT machines are exactly as powerful as Turing machines. But as we shall see with Theorem 9.3.1, from the complexity standpoint this equivalence appears to break down.

Much of the work in this chapter has appeared in our publication *An Atemporal Model of Physical Complexity* [74].

9.0.2 Observations on Computational Resource Usage

Though a Turing machine is typically defined as being unbounded in time and space, a halting computation on a Turing machine is usually understood to be finite in time and space. Hence it should be possible to describe a Turing machine computation in time t and space s via a structure with a domain of size $\max(t, s)$.

Similarly we typically may view a kinematic system as occurring within a continuously infinite space. But if when implementing such a system for the purpose of a computation we require only bounded precision, along with bounded space and time, then such a computation may be described by a finite structure that approximates a continuously infinite space. For example a computation of precision ϵ, taking time t and within a space of diameter r, may be described via a structure of size $\max(t, \frac{r}{\epsilon})$.

We therefore argue that if a theory machine on input Φ is satisfied by a finite structure of size n, then the amount of computational resources required to carry out a computation on input Φ is of order at most n.

Definition 9.0.8 Let $A \subseteq \mathcal{A}^*$ be a finite word problem, and $q : \mathbb{N} \to \mathbb{N}$ be a strictly increasing function. We say that an $\mathfrak{LS}_{\mathcal{V}}$-theory machine $\mathcal{M} = (\mathcal{T}, \mathcal{I}, \mathcal{O})$ is able to *compute A with q resources* if $\mathcal{M}$ is able to compute A via some simple sequence $\mathcal{X}$ and $\hat{\Sigma}^*_{\mathcal{X}} \subseteq \mathcal{I}$, such that for every $w \in \mathcal{A}^*$ there exists an $\mathfrak{LS}_{\mathcal{V}}$-structure $\mathfrak{A}$ where:

$$\mathcal{T} \cup \Phi^*_{\mathcal{X}}(w) \models_{\mathfrak{LS}_{\mathcal{V}}} \mathfrak{A}, \quad \text{and} \quad |\text{dom}(\mathfrak{A})| = O(q(|w|)),$$

where $|\text{dom}(\mathfrak{A})|$ denotes the cardinality of the domain of $\mathfrak{A}$.

So the idea behind the above definition is that the theory machine $\mathcal{M}$ compute A with q resources if for any input w there exists a model of $\mathcal{M}$ on input w which requires at most order $q(|w|)$ resources to correctly compute an output. Clearly this mimics the concepts of time and space complexity for Turing machines[1].

Further, as per our reasoning in Chapter 8, we argue that if physical system $\mathcal{S}$ can compute A, and $\mathcal{S}$ can be characterised by a FFOT machine that is able to compute A with q resources then on input w the physical system $\mathcal{S}$ requires at most order $q(|w|)$ resources to decide A.

Definition 9.0.9 A theory machine $\mathcal{M} = (\mathcal{T}, \mathcal{I}, \mathcal{O})$ is *finitely modelable* if for every $\Phi \in \mathcal{I}$ there exists a model $\mathfrak{M}$ of $\mathcal{T} \cup \Phi$, in which $|\text{dom}(\mathfrak{M})|$ is finite.

[1]See Definitions 2.2.1 and 2.2.2.

Clearly, it is a necessary requirement that a theory machine is a finitely modelable in order for it to be a able to compute a problem with q resources for any $q : \mathbb{N} \to \mathbb{N}$.

Definition 9.0.10 Let $A \subseteq \mathcal{A}^*$ be a finite decision problem. We say that a theory machine $\mathcal{M} = (\mathcal{T}, \mathcal{I}, \mathcal{O})$ is able to *compute A with polynomial resources* if $\mathcal{M}$ is able to compute A with p resources, and p is a polynomial function[1].

If $\mathcal{M}$ is a theory machine that is able to compute A but $\mathcal{M}$ is not able to compute A with polynomial resources then we say that $\mathcal{M}$ *requires super-polynomial resources* to compute A. The primary reason why $\mathcal{P}$ is considered to be the class of problems that are feasibly computable with a Turing machine is because polynomial time growth is *relatively* slow. For the similar reasons we argue that a problem $A \subseteq \mathcal{A}^*$ can be feasibly computed with a theory machine $\mathcal{M}$ if and only $\mathcal{M}$ is able to compute A with polynomially resources.

Further, we believe that a randomness-free physical system $\mathcal{S}$ is able feasibly compute A only if $\mathcal{S}$ can be characterised by a FFOT machine which is able to compute A with polynomial resources.

Remark 9.0.11 The reason why we believe that this feasible characterisation only works for randomness-free system is because in order to properly characterise a probabilistic computation system[2], a theory machine has to model *every* possible computation path of the system. This is due to the fact that a theory machine computation only gives outputs that it is certain of, and in order to be certain that one has the correct output of a probabilistic computer one must ensure that it is the most likely outcome. This means that a probabilistic Turing machine that computes in polynomial time but has an exponentially growing number of computation paths can only be characterised with a theory machine with super-polynomial resource growth.

[1]See Definition 2.2.5

[2]Such as a probabilistic Turing machine [4] or quantum computer [50, 59]

9.1 Boundedly Characterising a Turing machine

Despite the fact that a halting Turing machine computation takes place over a finite time period and uses a finite number of tape squares, our characterisation of a Turing machine $\mathcal{TM}^-_M$ in Example 8.1.1 is not finitely modelable. This is because every structure which satisfies the machine's theory contains an expansion of $\mathbb{Z}$ and therefore has a domain with infinite cardinality.

However it is possible to characterise a Turing machine with a FFOT machine that has bounded models of arbitrary finite size. We just need to replace ISA^- in the theory of $\mathcal{TM}^-_M$ with the bounded first-order integer successor axioms $BISA^-$ defined below.

$BISA^-$ is modelled by structures that are similar to $\mathbb{Z}$, but have a specified greatest number r, with $S(r) = r$, and a specified least number l which has no predecessor. We will demonstrate in Proposition 9.1.3 that the set of $FO^=$-models of $BISA^-$ includes finite structures with domain $\{a, a+1, \ldots, -1, 0, 1, \ldots, b-1, b\}$.

Definition 9.1.1 The set of *bounded first-order integer successor axioms* in the vocabulary of $\{=, S, 0, l, r\}$, where $=$ is a binary relation, S is a unary function, and l, r are constants, is:

$$BISA^- = \left\{ \begin{array}{ll} \text{B}\mathbb{Z}\text{Ax}(1.1) & \forall x \neg(S(x) = l), \\ \text{B}\mathbb{Z}\text{Ax}(1.2) & (S(r) = r), \\ \text{B}\mathbb{Z}\text{Ax}(1.3) & \forall x \forall y((S(x) = S(y)) \to ((x = y) \vee (S(x) = r))), \\ \text{B}\mathbb{Z}\text{Ax}(2) & \forall x((x = l) \vee \exists y(S(y) = x)), \\ \text{B}\mathbb{Z}\text{Ax}(3.1) & \forall x((x < S(x)) \vee (x = r)), \\ \text{B}\mathbb{Z}\text{Ax}(3.2) & \forall x((l = x) \vee (l < x)), \\ \text{B}\mathbb{Z}\text{Ax}(3.3) & \forall x((r = x) \vee (x < r)), \\ \mathbb{Z}\text{Ax}(4) & \forall x \neg \exists y((x < y) \wedge (y < S(x)), \\ \mathbb{Z}\text{Ax}(5) & \forall x \neg(x < x), \\ \mathbb{Z}\text{Ax}(6) & \forall x \forall y(x < y) \to \neg(y < x), \\ \mathbb{Z}\text{Ax}(7) & \forall x \forall y \forall z((x < y) \wedge (y < z)) \to (x < z) \end{array} \right\}.$$

B$\mathbb{Z}$Ax(1.1)-B$\mathbb{Z}$Ax(1.3) correspond to a bounded version of the axiom $\mathbb{Z}$Ax(1) in ISA^-. Similarly B$\mathbb{Z}$Ax(2) corresponds to $\mathbb{Z}$Ax(2) and B$\mathbb{Z}$Ax(3.1)-B$\mathbb{Z}$Ax(3.3) correspond to $\mathbb{Z}$Ax(3) in ISA^-. The remaining axioms are then identical to the remaining axioms in ISA^-.

Notation 9.1.2 For any $a, \in \mathbb{Z}$, where $a < 0$ and $b > 0$, let $\mathbb{Z}_{[a,b]}$ denote the set $\{a, a+1, \ldots, -1, 0, 1, \ldots, b-1, b\} \subset \mathbb{Z}$.

Proposition 9.1.3 *Let $\mathcal{V} = \{<, S, \check{0}, l, r\}$. For each $a, b \in \mathbb{Z}$ where $a < 0 < b$ let $\mathfrak{K}_{[a,b]} = \langle \mathbb{Z}_{[a,b]}, <, S, \check{0}, l, r\rangle$ have the usual ordering and a successor function such that $S(b) = b$ and $S(x) = x + 1$ if $x \neq b$. Also let $l = a$, $r = b$, and $\check{0} = 0$. It is then the case that $\mathfrak{K}_{[a,b]}$ is an $FO^{=}_{\mathcal{V}}$-model of $BISA^-$.*

Proof: B$\mathbb{Z}$Ax(1.1) holds in $\mathfrak{K}_{[a,b]}$ as by definition $a-1$ does not lie in $\mathbb{Z}_{[a,b]}$, similarly B$\mathbb{Z}$Ax(1.2) holds as $S(b) = b$. B$\mathbb{Z}$Ax(2) is then true as for every $p \in (\mathbb{Z}_{[a,b]} \setminus \{a\})$ we know that $p - 1$ lies in $\mathbb{Z}_{[a,b]}$. Also B$\mathbb{Z}$Ax(1.3) holds in $\mathfrak{K}_{[a,b]}$ as in $(\mathbb{Z}_{[a,b]} \setminus \{b\})$ the function S is an injection, whereas if $S(q) = S(q')$ and $q \neq q'$ then we must have $q, q' \in \{b-1, b\}$ and so $S(q) = b = r$, which by B$\mathbb{Z}$Ax(1.3) is also fine.

The ordering on $\mathbb{Z}_{[a,b]}$ is the same as the usual ordering on $\mathbb{Z}$, which means that $<$ in $\mathfrak{K}_{[a,b]}$ is still a strict total order, hence $\mathbb{Z}$Ax(5)-$\mathbb{Z}$Ax(7) are satisfied. As with $\mathbb{Z}$, for every $p \in (\mathbb{Z}_{[a,b]} \setminus \{b\})$ the value of $S(p) = p + 1$ is greater than p, hence B$\mathbb{Z}$Ax(3.1) is true in $\mathfrak{K}$.

Also, as in $\mathbb{Z}$ for each $q \in \mathbb{Z}_{[a,b]}$ there are no elements of $\mathbb{Z}_{[a,b]}$ between q and $S(q)$. Since if $q = b$ then $S(q) = q$, and otherwise if there existed an element between q and $S(q)$ then it would also be present in $\mathbb{Z}$, which would mean that $\mathbb{Z}$ would not satisfy $\mathbb{Z}$Ax(4). However from Proposition A.1.6 we know that $\mathbb{Z}$Ax(4) is satisfied in $\mathbb{Z}$, hence it is also true in $\mathfrak{K}_{[a,b]}$. Finally B$\mathbb{Z}$Ax(3.2) and B$\mathbb{Z}$Ax(3.3) are satisfied as clearly by definition a is the least element of $\mathbb{Z}_{[a,b]}$ and b is the greatest element of $\mathbb{Z}_{[a,b]}$. ❏

Corollary 9.1.4 *The bounded first-order integer successor axioms are finitely modelable.*

Proposition 9.1.5 *Let $\mathcal{V} = \{<, S, \check{0}, l, r\}$ and $\mathcal{V}' = \{<, S, \check{0}\}$. Let $\mathfrak{M}$ be an $FO^{=}_{\mathcal{V}}$-model of $BISA^{-}$.*

Suppose that there exists a $d \in dom(\mathfrak{M})$ and an $N \in \mathbb{N} \setminus \{0, 1\}$ such that for any $m, n \in \{0, \dots, N\}$ if $m \neq n$ then $S^m(d) \neq S^n(d)$ in $\mathfrak{M}$, and for some $j \in \{0, \dots, N\}$ we have that $S^l(d) = \check{0}$.

Let $a = -j$ and $b = N - j$. Also let $\mathfrak{K}_{[a,b]}$ be as in Proposition 9.1.3. Then the map $\mu : \mathfrak{K}_{[a,b]} \to \mathfrak{M}$ given by $\mu(n) = S^n(\check{0})$ (where as in Proposition 8.0.14 $S^0(\check{0}) = \check{0}$ and $S^{-m}(\check{0}) = q$ iff $S^m(q) = \check{0}$) is well-defined and such that:

- *$\mu(0) = \check{0}$.*
- *For any $p_1 \in \mathbb{Z}_{[a,b]} \setminus \{b\}$ and $p_2 \in \mathbb{Z}_{[a,b]}$ we have $p_1 + 1 = p_2$ in $\mathfrak{K}_{[a,b]}$ iff $S(\mu(p_1)) = \mu(p_2)$ in $\mathfrak{M}$[1].*
- *For any $p_1, p_2 \in \mathbb{Z}_{[a,b]}$ we have $p_1 < p_2$ in $\mathfrak{K}_{[a,b]}$ iff $\mu(p_1) < \mu(p_2)$ in $\mathfrak{M}$.*

Proof: By assumption μ is a well-defined mapping, as $\text{dom}(\mathfrak{K}_{[a,b]}) = \mathbb{Z}_{[a,b]}$, and so for any $p \in \mathbb{Z}_{[a,b]}$ we have $\mu(p) = S^p(\check{0}) = S^p(S^j(d)) = S^{p+j}(d)$.

Clearly by definition $\mu(0) = S^0(\check{0}) = \check{0}$. Also for any $p_1 \in \mathbb{Z}_{[a,b]} \setminus \{b\}$ and $p_2 \in \mathbb{Z}_{[a,b]}$ if $p_1 + 1 = p_2$ in $\mathfrak{K}_{[a,b]}$ then $S(\mu(p_1)) = S(S^{p_1+j}(d)) = S^{p_1+1+j}(d) = S^{p_2+j}(d) = \mu(p_2)$.

Now by B$\mathbb{Z}$Ax(3.1) for any q in $\text{dom}(\mathfrak{M})$ we have $q < S(q)$. By $\mathbb{Z}$Ax(7) $<$ is also defined to be a transitive relation, so since $d < S(d)$, by induction we have $S^m(d) < S^n(d)$ for any $m, n \in \{0, \dots, N\}$ such that $m < n$ in $\mathbb{N}$. Therefore if $p_1 < p_2$ in $\mathfrak{K}_{[a,b]}$ then $\mu(p_1) = S^{p_1+j}(d) < S^{p_2+j}(d) = \mu(p_2)$, and so $\mu(p_1) < \mu(p_2)$ in $\mathfrak{M}$.

Conversely, if $p_1 \not< p_2$ in $\mathfrak{K}_{[a,b]}$ then $p_1 \leqslant p_2$, and if $p_1 = p_2$ then by $\mathbb{Z}$Ax(5) $\neg(\mu(p_1) < \mu(p_2))$. Whereas if $p_2 < p_1$ in $\mathfrak{K}_{[a,b]}$ then $\mu(p_2) < \mu(p_1)$ and by $\mathbb{Z}$Ax(6) $\neg(\mu(p_1) < \mu(p_2))$. Consequently $p_1 < p_2$ in $\mathfrak{K}_{[a,b]}$ iff $\mu(p_1) < \mu(p_2)$ in $\mathfrak{M}$. ❑

[1]So μ is *almost* an embedding from $\mathfrak{K}_{[a,b]}$ to $\mathfrak{M}$, when looking at $<$, S and $\check{0}$. The difference is that $S(b) = b$ in $\mathfrak{K}_{[a,b]}$, but this need not be true for $\mu(b)$ in $\mathfrak{M}$.

We may now characterise a Turing machine computing a decision problem with a finitely modelable FFOT machine in in a similar manner to how we characterised it by a finite $FO^{=}$-theory machine in Example 8.1.1.

Example 9.1.6 Let $M = (\Lambda, \Pi, \mathbf{b}, \mathcal{A}, s_0, \langle s_a, s_r \rangle, \mathcal{R})$ be a Turing machine which computes the decision problem $A \subseteq \mathcal{A}^*$, in time $t : \mathbb{N} \to \mathbb{N}$ and space $u : \mathbb{N} \to \mathbb{N}$.

We can characterise M via the finite $FO^{=}_{\mathcal{BV}_M}$-machine:

$$\mathcal{BTM}^{-}_{M} = (\mathcal{BTMT}^{-}_{M}, \hat{\mathcal{A}}^{*}_{\mathcal{X}_{TM}}, \{\{I(h) = s_a\}, \{I(h) = s_r\}\}),$$

in the vocabulary of:

$$\mathcal{BV}_M = \mathcal{V}_M \cup \{l, r\},$$

where $\mathcal{V}_M$ is the vocabulary used in Example 5.1.1 and $\mathcal{X}_{TM}$ is the input sequence used in Example 5.1.1 . The theory of $\mathcal{BTM}^{-}_{M}$ is then:

$$\mathcal{BTMT}^{-}_{M} = (\mathcal{TMT}_M \setminus (ISA \cup ET_{\mathcal{R}})) \cup BISA^{-} \cup ET'_{\mathcal{R}}.$$

Where $\mathcal{TMT}_M$ is the theory of $\mathcal{TM}_M$ in Example 5.1.1. So the difference is that $\mathcal{BTMT}^{-}_{M}$ contains the bounded integer successor axioms rather than the standard integer successor axioms, also $\mathcal{BTMT}^{-}_{M}$ contains $ET'_{\mathcal{R}}$, which is a slightly modified version of the set of sentences that implement the rules of $\mathcal{R}$.

Specifically $ET'_{\mathcal{R}}$ is:

$$ET'_{\mathcal{R}} = \left\{ \begin{array}{l} \forall x((0 < S(x)) \wedge \mu_{(s,a)}(x,x)) \to \\ \qquad\qquad (\mu_{(r,b)}(S(x),x) \wedge \pi'_{(p)}(x) \wedge \nu(x))) \end{array} \;\middle|\; (s,a;r,b,p) \in \mathcal{R} \right\}.$$

Where μ and ν are as in Example 5.1.1. Whereas instead of $\pi_{(p)}(x)$, for each $p \in \{LEFT, PAUSE, RIGHT\}$ we have the term:

$$\pi'_{(p)}(x) \equiv \begin{cases} (H(S(x)) = S(H(x))) \wedge \neg(H(x) = S(H(x))) & \text{if } p = \text{RIGHT}, \\ H(S(x)) = H(x) & \text{if } p = \text{PAUSE}, \\ (S(H(S(x))) = H(x) \wedge \neg(H(S(x))) = S(H(S(x)))) & \text{if } p = \text{LEFT}, \end{cases}$$

So for each rule of $\mathcal{R}$ we have the added condition that if the head moves right or left, then it must move to a different tape cell. Otherwise the rules are implemented in the same manner.

Let $\mathfrak{G}$ be an $FO^{=}_{\mathcal{V}_M}$-model of $\mathcal{BTMT}^{-}_{M} \cup \Phi^{*}_{\mathcal{X}_{TM}}(w)$. By the input $\Phi^{*}_{\mathcal{X}_{TM}}(w) = \{C(0, S^{i}(0)) = w_i \mid i \in \{0, \dots, |w| - 1\}\} \cup \{C(0, S^{|w|}(0)) = \mathbf{b}\}$ it follows that $\mathfrak{G} \models_{FO^{=}_{\mathcal{BV}_M}} \neg(C(0, S^{i}(0)) = \mathbf{b})$ for all $i \in \{0, \dots, |w| - 1\}$, but also $\mathfrak{G} \models_{FO^{=}_{\mathcal{BV}_M}} C(0, S^{|w|}(0)) = \mathbf{b}$. Hence $\mathfrak{G} \models_{FO^{=}_{\mathcal{BV}_M}} \neg(S^{m}(0) = S^{n}(0))$ for all $m, n \in \{0, \dots, |w|\}$ such that $m \neq n$. Therefore by Proposition 9.1.5 there is a map from $\mathfrak{K}_{[0,|w|]}$ onto $\mathfrak{G}$ which preserves 0, ordering and the successor function from 0 to $|w| - 1$. The sentences defining the initial configuration of M on input w are the same as in Example 5.1.1, so the initial configuration of $\mathfrak{G}$ is the same as M on cells 0 to $|w|$.

Further the sentences of $ET'_{\mathcal{R}}$ allow $\mathfrak{G}$ to build on this, as they implement the rules of $\mathcal{R}$ whilst (with the addition of $\pi'_{(p)}(x)$) also ensuring that if $\mathfrak{G}$ needs to use a tape cell outside of $\mathrm{dom}(\mathfrak{K}_{[0,|w|]}) = \mathbb{Z}_{[0,|w|]}$ then such a tape cell is distinct from every other tape cell in $\mathbb{Z}_{[0,|w|]}$. So by Proposition 9.1.5 we have a mapping onto $\mathfrak{G}$ from some large enough $\mathfrak{K}_{[a,b]}$ structure. Therefore by the same reasoning as in Example 5.1.1 from time 0 to $|w|$ the configurations of $\mathfrak{G}$ must be the same as the configurations of M on input w

Now, a key property of a halting Turing machine computation is that for any two distinct time steps x_1, x_2 between 0 and the halting time, the configuration of the machine at time x_1 differs from the configuration of the machine at time x_2. As otherwise the machine would become stuck in an infinite loop. So in $\mathfrak{G}$ this means that there is some y such that $\neg(C(x_1, y) = C(x_2, y))$ in $\mathfrak{G}$, and therefore $\neg(x_1 = x_2)$ in $\mathfrak{G}$ for all $x_1, x_2 \in \{0, \dots, t(|w|)\}$ such that $x_1 \neq x_2$. Consequently by Proposition 9.1.5 again we have a mapping onto $\mathfrak{G}$ from a structure $\mathfrak{K}_{[a,b]}$, where $\mathfrak{K}_{[a,b]}$ encompasses all of the time steps and tape cells used by the computation of M on input w.

Therefore by the same reasoning as in Example 8.1.1, the entirety of the computation of M on input w occurs in $\mathfrak{G}$ completely within the region of $\mathbb{Z}_{[a,b]}$, eventually reaching the state s_a or the state s_r and halting. As in Example 8.1.1 this output is directly implied by the input, and so as M computes A, it follows that $\mathcal{BTM}^{-}_{M}$ is able to compute A.

Now, regarding the resources that $\mathcal{BTM}^{-}_{M}$ computes A with. By the definition of the functions t and u the entirety of the computation of M on input w occurs only on the tape between cells $-u(|w|)$, and $u(|w|)$, and takes place between time steps 0 and $t(|w|)$.

Let $a = -u(|w|)$ and $b = \max\{t(|w|), u(|w|)\}$. It is therefore possible that $\mathfrak{G}$ is a $\mathcal{BV}_M$-expansion of $\mathfrak{K}_{[a,b]}$, as by Proposition 9.1.3, $\mathfrak{K}_{[a,b]}$ is a model of $BISA^- \subset \mathcal{BTMT}_M^-$, and by our reasoning above the entirety of the computation of M on input w may be described in $\mathfrak{G}$ completely within the region of $\mathbb{Z}_{[a,b]}$.

The cardinality of $\mathbb{Z}_{[a,b]}$ is $b - a + 1 = \max\{t(|w|), u(|w|)\} + u(|w|)$. By Remark 2.2.3 we see that $u(|w|) \leqslant t(|w|) + |w|$, hence $|\mathrm{dom}(\mathfrak{G})| \leqslant 2t(|w|) + 2|w| = O(t(|w|))$.

Consequently if t is a polynomial function, then $\mathcal{BTM}_M^-$ is able to compute A with polynomial resources.

9.2 Boundedly Characterising Blakey's factoriser

We conjecture that we should be able to describe a finitely bounded version of $\mathbb{R}$ with a modification of the dense ordered field axioms DOF, in a similar manner to how we described a bounded version of $\mathbb{Z}$ with $BISA^-$, a modification of the first-order integer successor axioms. For example, we could approximate $\mathbb{R}$ via a finite structure with a domain of the form $\{\frac{a}{m} \mid a \in \{-m^3, \ldots, m^3\}\}$, in which for elements between $-m$ and m addition can be defined as usual, whereas multiplication could be such that $\frac{a}{m} \times \frac{b}{m}$ is equal to whichever element in the domain is nearest to $\frac{ab}{m^2}$.

As in Subsections 5.2.2 and 8.1.2, it should be possible to characterise Blakey's factorisation system with an FFOT machine that includes this modified version of DOF. Blakey's factorisation system acts within a bounded region and is designed to output even with a degree of error. Indeed, if we define the partial derivatives as we did in Equation 5.1 in Subsection 5.2.1 then they will be defined to be approximations to their true value. So the outputs will also be the same, provided that each model of the system is sufficiently precise. To ensure that we do have enough precision we can define within the theory for the error of the model $\frac{1}{m}$ to be sufficiently small in relation to the input. For example for an input of $n \in \mathbb{N}$ we may have $(\frac{1}{m} \times n \times n \times n) \leqslant 1$, ensuring that $\frac{1}{m} \leqslant \frac{1}{n^3}$.

Now regarding the resource usage of such a machine. Clearly inputting n should, in general, give a different output to inputting $n+1$. Hence there must be a clear separation between $\frac{1}{2\sqrt{n}}$ and $\frac{1}{2\sqrt{n+1}}$, which means that in order to implement the device, the error $\frac{1}{m}$ must be less than $|\frac{1}{2\sqrt{n+1}} - \frac{1}{2\sqrt{n}}|$. This error shrinks at an inverse polynomial rate with respect to n, and at an inverse exponential rate with respect to the length of n's binary expansion. By the definition of the factoriser, the domain of any model of it must at least contain 0 and 1, and between these two numbers there are at least m elements of the domain. Therefore, as m grows exponentially with the size of the input, so must the minimal domain size.

We therefore conclude that such a FFOT machine requires super-polynomial resources in order to compute the factorisation problem, agreeing with Blakey's [17] assertion that his factoriser requires an infeasibly large resource growth.

9.3 Efficient Computation and $\mathcal{NP} \cap$ co-$\mathcal{NP}$.

As we mentioned before in Subsection 9.0.2 we argue that a computational problem is efficiently computable by a randomness-free physical system only if it is computable by FFOT machine with polynomial resources. In this section we offer an answer to what this class of problems is.

Theorem 9.3.1 *A problem is computable by a FFOT machine with polynomial resources if and only if it is in $\mathcal{NP} \cap$ co-$\mathcal{NP}$.*

Proof: ($\Rightarrow$) Let $p : \mathbb{N} \to \mathbb{N}$ be a polynomial function and $\mathcal{M} = (\mathcal{T}, \mathcal{I}, \mathcal{O})$ be a FFOT machine in the vocabulary $\mathcal{V}$ which computes $A \subseteq \mathcal{A}^*$ with p resources.

By assumption, for some simple sequence $\mathcal{X}$ and some $\Theta, \Psi \in \mathcal{O}$, we have $\hat{\mathcal{A}}^*_{\mathcal{X}} \subseteq \mathcal{I}$ and for each $w \in \mathcal{A}^*$ there is a finite $FO_{\mathcal{V}}$-structure $\mathfrak{A}$ satisfying $\mathcal{T} \cup \Phi^*_{\mathcal{X}}(w)$ with

$|\mathrm{dom}(\mathfrak{A})| \leqslant p(|w|)$. Also if $w \in A$ then $\mathfrak{A} \models_{FO_{\mathcal{V}}} \Theta$ and if $w \notin A$ then $\mathfrak{A} \models_{FO_{\mathcal{V}}} \Psi$. We can non-deterministically obtain such a structure as follows.

Let $\mathcal{V}$ contain m relations, k functions, and r constant symbols, also let each relation and function symbol have an arity at most l. We can encode each element of $\mathrm{dom}(\mathfrak{A})$ as a word in $\{0,1\}^{p(|w|)}$. Each relation can then be encoded as a string of length $O(p(|w|)^l)$ by simply listing the strings representing the related elements. Similarly each function can be encoded by a string of length $O(p(|w|)^{l+1})$ and each constant by a string of length $O(p(|w|))$. We can therefore encode an exact description of $\mathfrak{A}$ by a single finite word $\rho_w \in \{0,1\}^{q(|w|)}$, where $q(n) = O(m \cdot p(n)^l + k \cdot p(n)^{l+1} + r \cdot p(n))$, which is polynomial in the length of w.

In a fixed domain $\mathrm{dom}(\mathfrak{A})$ a sentence of the form $\forall x \phi(x)$ is true iff the sentence $\bigwedge_{a \in \mathrm{dom}(\mathfrak{A})} \phi(a)$ is true. Similarly $\exists x \psi(x)$ is true iff $\bigvee_{b \in \mathrm{dom}(\mathfrak{A})} \psi(b)$ is true. Hence to check if:

$$\forall x_1 \exists x_2 \cdots \forall x_{m-1} \exists x_m \theta(x_1, \ldots, x_m) \in \mathcal{T},$$

is true in $\mathfrak{A}$ it is sufficient to determine whether:

$$\bigwedge_{a_1 \in \mathrm{dom}(\mathfrak{A})} \bigvee_{a_2 \in \mathrm{dom}(\mathfrak{A})} \cdots \bigwedge_{a_{m-1} \in \mathrm{dom}(\mathfrak{A})} \bigvee_{a_m \in \mathrm{dom}(\mathfrak{A})} \theta(a_1, \ldots, a_m),$$

is true in $\mathfrak{A}$. This can be achieved by checking whether $\theta(a_1, \ldots, a_m)$ is true in at most $|\mathrm{dom}(\mathfrak{A})|^m$ assignments.

There is a fixed number of sentences in $\mathcal{T}$ and the number of quantifiers in each one is fixed, hence the time taken to test whether $\mathfrak{A} \models_{FO_{\mathcal{V}}} \mathcal{T}$ grows polynomially with $|w|$. The number of sentences in $\Phi^*_{\mathcal{X}}(w)$ is equal to $|w|+1$ and each sentence in $\Phi^*_{\mathcal{X}}(w)$ is a quantifier-free sentence whose length grows linearly with $|w|$, therefore the time to determine whether $\mathfrak{A}$ models $\Phi^*_{\mathcal{X}}(w)$ also takes time polynomial in $|w|$.

We can therefore construct a non-deterministic Turing machine M_1, that given any input $w \in \mathcal{A}^*$, tries to non-deterministically generate a description ρ_w of some structure $\mathfrak{A}$ modelling $\mathcal{T} \cup \Phi^*_{\mathcal{X}}(w)$. After generating ρ_w the machine checks in polynomially many steps whether each sentence of $\mathcal{T} \cup \Phi^*_{\mathcal{X}}(w)$ is true in $\mathfrak{A}$. If it does then M_1 determines whether $\mathfrak{A} \models_{FO_{\mathcal{V}}} \Theta$.

As Θ is a fixed finite set of sentences, like $\mathcal{T}$, this decision process can be carried out in time polynomial in $|w|$. If $\mathfrak{A}$ does $FO_{\mathcal{V}}$-model Θ then M_1 accepts w. If any sentence in $\mathcal{T} \cup \Phi^*_{\mathcal{X}}(w) \cup \Theta$ is false in $\mathfrak{A}$ then M_1 halts. Thus if for all possible ρ_w we have that Θ is false in any structure which models $\mathcal{T} \cup \Phi^*_{\mathcal{X}}(w)$ then M_1 rejects w.

By assumption, for any $w \in \mathcal{A}^*$, if $\mathfrak{A} \models_{FO_{\mathcal{V}}} \mathcal{T} \cup \Phi^*_{\mathcal{X}}(w)$ then $\mathfrak{A} \models_{FO_{\mathcal{V}}} \Theta$ iff $w \in A$. Therefore M_1 accepts w if and only if $w \in A$, and since M_1 computes in non-deterministic polynomial time we have that $A \in \mathcal{NP}$.

Conversely to see that $A \in$ co-$\mathcal{NP}$ we can construct a non-deterministic polynomial time Turing machine M_2 which acts the same as M_1, except that M_2 checks whether $\mathfrak{A}$ models Ψ rather than Θ. By the same reasoning as above M_2 accepts $w \in \mathcal{A}^*$ iff $w \in \mathcal{A}^* \setminus A$, therefore $\mathcal{A}^* \setminus A \in \mathcal{NP}$ and $A \in$ co-$\mathcal{NP}$. Thus by combining this with the above result we have $A \in \mathcal{NP} \cap$ co-$\mathcal{NP}$.

($\Leftarrow$) If $B \in \mathcal{NP} \cap$ co-$\mathcal{NP}$ then $B \in \mathcal{NP}$ and $\mathcal{A}^* \setminus B \in \mathcal{NP}$, hence there must exist two non-deterministic polynomial time Turing machines N_1, N_2 that respectively decide B and $\mathcal{A}^* \setminus B$. To avoid confusion we can assume without loss of generality that N_1 and N_2 have disjoint sets of internal states.

We can then construct a finitely modelable FFOT machine $\mathcal{BTM}^-_{(N_1,N_2)}$ which acts like a non-deterministic Turing machine that can implement the rules from either N_1 or N_2. In addition $\mathcal{BTM}^-_{(N_1,N_2)}$ is defined in such a way that it must always produces a computational path which halts on the accepting state of N_1 or N_2, any other computational path will lead to a logically inconsistency (see Figures 9.3 and 9.3). Such a computation will of course require only polynomial resources (as N_1 and N_2 compute in polynomial time). So by simply looking at which of these accepting states is reached by $\mathcal{BTM}^-_{(N_1,N_2)}$ on input $\Phi^*_{\mathcal{X}_{TM}}(w)$ we are able to compute B with polynomial resources.

For each $i \in \{1, 2\}$ let $N_i = (\Lambda_i, \Pi_i, \mathbf{b}, \mathcal{A}, s_{0_i}, \langle s_{a_i}, s_{r_i} \rangle, \mathcal{R}_i)$. The machine has non-deterministic rule set $\mathcal{R}_i$, so for each $(t, b) \in (\Pi_i \times \Lambda_i)$ let $\mathcal{R}^{(t,b)}_i$ denote the set of rule suffixes of $\mathcal{R}_i$ that are prefixed by (t, b). If N_i is in state t reading b then any one of the rule suffixes in $\mathcal{R}^{(t,b)}_i$ may be applied. Also (by relabelling if necessary) let $\Pi_1 \cap \Pi_2 = \emptyset$.

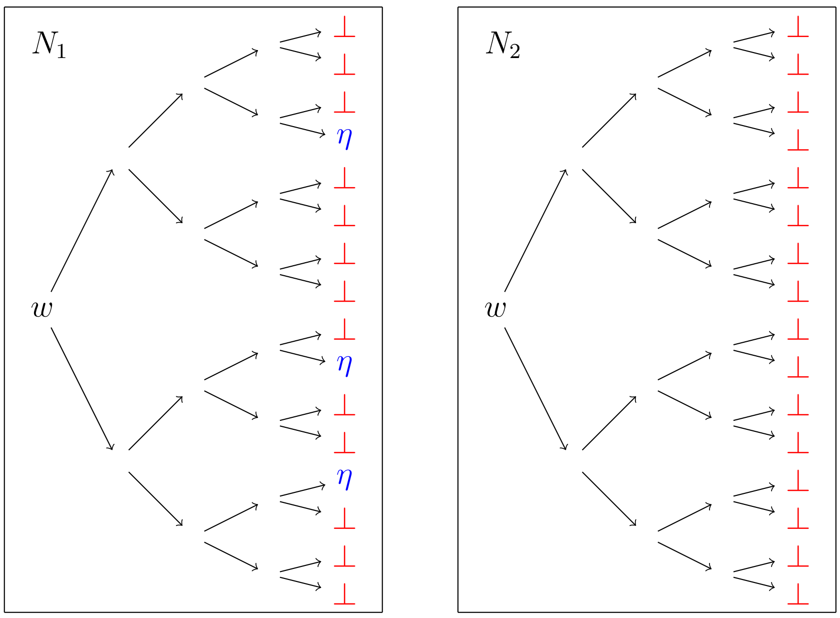


Figure 9.1: The possible computation paths of N_1 and N_2 on input w.

In the vocabulary of $\mathcal{V}_{(N_1,N_2)} = \mathcal{V}^{N_1} \cup \mathcal{V}^{N_2} \cup \{e\}$ let:

$$\mathcal{BTM}^-_{(N_1,N_2)} = (\mathcal{BTMT}^-_{(N_1,N_2)}, \hat{\mathcal{A}}^*_{\mathcal{X}}, \{\{I(h) = s_{a_1}\}, \{I(h) = s_{a_2}\}\}),$$

be an $FO^=_{\mathcal{V}_{(N_1,N_2)}}$ with theory:

$$\mathcal{BTMT}^-_{(N_1,N_2)} = BISA^- \cup CD^=_{\mathcal{V}_{(N_1,N_2)}} \cup IT_{(s_{0_1},s_{0_2})} \cup ET'_{\mathcal{R}_1} \cup ET'_{\mathcal{R}_2} \cup HT'_{(s_{a_1},s_{a_2})}.$$

Where $BISA^-$ is the set of bounded integer successor axioms, and $CD^=_{\mathcal{V}_{(N_1,N_2)}}$ is the set of distinct constant axioms for $\mathcal{V}_{(N_1,N_2)}$. Further, in a similar manner to Example 5.1.1, the set of sentences $IT_{(s_{0_1},s_{0_2})}$ defines the two possible initial configurations of the machine, and the sets $ET'_{\mathcal{R}_1}$ and $ET'_{\mathcal{R}_2}$ describe the two possible modes of evolution of the machine. The set $HT'_{(s_{a_1},s_{a_2})}$ ensures that the machine halts when it reaches s_{a_1} or s_{a_2}, crucially $HT'_{(s_{a_1},s_{a_2})}$ also ensures that the machine must halt in one of these states.

Specifically we have:

$$IT_{(s_{0_1},s_{0_2})} = \left\{ \begin{array}{l} (H(0) = 0), \\ (I(0) = s_{0_1}) \vee (I(0) = s_{0_2}), \\ \forall y(((C(0,y) = \mathbf{b}) \wedge (0 < y)) \to (C(0,S(y)) = \mathbf{b})), \\ \forall y((y < 0) \to (C(0,y) = \mathbf{b})) \end{array} \right\}.$$

Which is the same as IT_{s_0}, except for the fact that the initial state could begin in either state s_{0_1} or state s_{0_2}. So as in Example 5.1.1, for any model $\mathfrak{H}$ of $\mathcal{BTMT}^{-}_{(N_1,N_2)} \cup \Phi^*_{\mathfrak{X}_{TM}}(w)$ the initial configuration of $\mathfrak{H}$ is the same as it is for either N_1 or N_2 on input w.

For each $i \in \{1,2\}$ the non-deterministic rules of $\mathcal{R}_i$ are implemented by:

$$ET'_{\mathcal{R}_i} = \left\{ \begin{array}{l} \forall x\,(((0 < S(x)) \wedge \mu_{(s,a)}(x,x)) \to \\ \quad \bigvee_{(v,b,p)\in\mathcal{R}_i^{(s,a)}} (\mu_{(v,b)}(S(x),x) \wedge \pi'_{(p)}(x) \wedge \nu(x))) \end{array} \;\middle|\; (s,a) \in (\Pi_i \times \Lambda_i) \right\}.$$

Where μ and ν are as they are in Example 5.1.1, and $\pi'_{(p)}$, is as it is in Example 9.1.6. So for each set $\mathcal{R}_i^{(s,a)}$ each sentence of $ET'_{\mathcal{R}_i}$ implements a rule of $\mathcal{R}_i$ beginning with (s,a). Exactly which rule is implemented is not specified, however two different rules of $\mathcal{R}_i^{(s,a)}$ cannot be implemented simultaneously as this would lead to a contradiction.

By the same reasoning as in Example 9.1.6 it follows from $BISA^{-} \cup IT_{(s_{0_1},s_{0_2})} \cup ET'_{\mathcal{R}_1} \cup ET'_{\mathcal{R}_2}$ in $\mathcal{BTMT}^{-}_{(N_1,N_2)}$, and Proposition 9.1.5 that any possible the computation of N_1 or N_2 takes place within a bounded region corresponding to $\mathbb{Z}_{[a,b]}$ in $\mathfrak{H}$.

Now by the same reasoning as in Example 5.1.1, if the configuration of $\mathfrak{H}$ at time x is the same as a possible configuration of N_i at time x then the configuration of $\mathfrak{H}$ at time $x+1$ is also a possible configuration of N_i at time $x+1$.

Finally for halting we have:

$$HT'_{(s_{a_1},s_{a_2})} = \left\{ \begin{array}{l} \forall x((I(x) = s_{a_1}) \to (h = x)), \\ \forall x((I(x) = s_{a_2}) \to (h = x)), \\ \forall x(\neg(I(x) = s_{r_1}) \wedge \neg(I(x) = s_{r_2})), \\ (I(h) = s_{a_1}) \vee (I(h) = s_{a_2}) \end{array} \right\}.$$

Which corresponds to $HT_{(s_a,s_r)}$, with s_a and s_r replaced by s_{a_1} and s_{a_2}, together with (crucially) the last two sentences which state that the rejecting states can never be reached, and any model $\mathfrak{H}$ must eventually halt in one of the two accept states.

So if at time x the machine is in the internal state s_{a_1} or the internal state s_{a_2} then $x = h$ in $\mathfrak{H}$. The output $\{I(h) = s_{a_1}\}$ or $\{I(h) = s_{a_2}\}$ is therefore defined at this time.

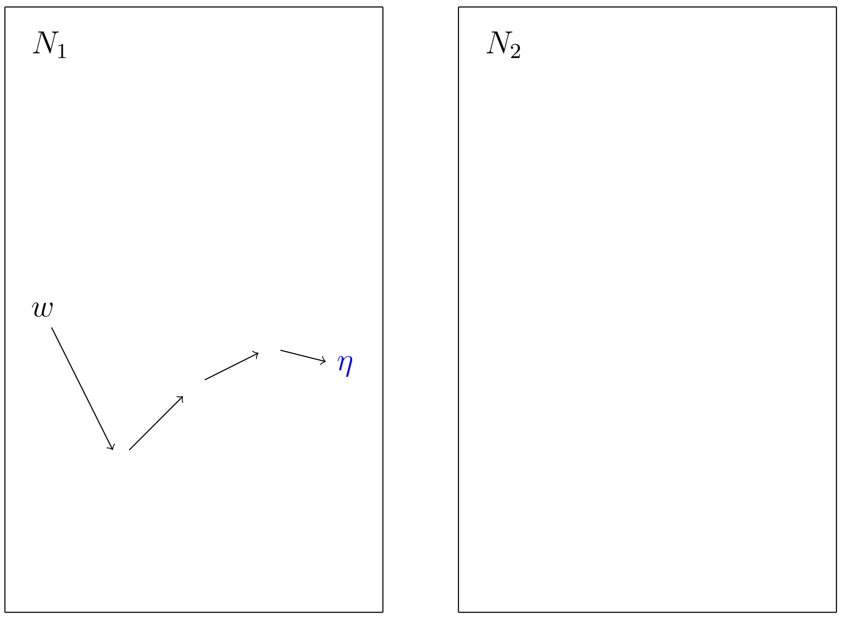


Figure 9.2: A computation path that may occur within $\mathfrak{H}$.

So the configuration of $\mathfrak{H}$ at time 0, along with the configuration evolution of $\mathfrak{H}$ is same as N_1 and N_2 with input w. Therefore by induction, for any $x \in \mathbb{N}$ the configuration of $\mathfrak{H}$ at time x is a possible configuration of N_1 or N_2 at time x. By $HT'_{(s_{a_1},s_{a_2})}$ no computation happening in $\mathfrak{H}$ can be a rejecting computation, as if so then s_{r_1} or s_{r_2} would eventually be reached.

Hence any computation that happens in $\mathfrak{H}$ must be an accepting computation, and if $w \in B$ then $\mathfrak{H}$ must describe a computation of N_1 that ends in state s_{a_1}, as any computation of N_2 on input w would end in the reject state s_{r_2}. Conversely if $w \in \mathcal{A}^* \setminus B$ then $\mathfrak{H}$ must describe a computation of N_2 that ends in state s_{a_2}.

This means that $\mathfrak{H} \models_{FO^=} (I(h) = s_{a_1})$ iff $w \in B$. So $\mathcal{BTM}^-_{(N_1,N_2)}$ is able to compute B. Now we also know that any accepting computation of N_1 or N_2 takes a polynomial number of time steps (and so uses a polynomial number of tape squares). Therefore by our reasoning in Example 9.1.6 and the fact that $BISA^- \subset \mathcal{BTMT}^-_{(N_1,N_2)}$ we know that there exists a model $\mathfrak{H}$ of $\mathcal{BTMT}^-_{(N_1,N_2)} \cup \Phi^*_{\mathfrak{X}_{TM}}(w)$ such that $|\text{dom}(\mathfrak{H})|$ is polynomial in $|w|$.

Consequently $\mathcal{BTM}^-_{(N_1,N_2)}$ is able to compute B in polynomial resources. ❑

The FFOT machine described in the above proof will only follow a computational path if that path eventually leads to an accept state. The only way the machine could know which paths to take would be if potential future states are somehow able to influence the present states. The above theory machine therefore acts in a non-causal and somewhat atemporal manner, whilst still being clearly bounded in its computational capabilities.

Hence if $\mathcal{P} \neq \mathcal{NP} \cap$ co-$\mathcal{NP}$ then our result implies that atemporal/non-causal physical computation is more powerful then classical sequential computation.

The problems with known quantum polynomial time algorithms that are believed to lie in $\mathcal{BQP} \setminus \mathcal{P}$ can all be phrased as a hidden subgroup problem [59], which also lies in $\mathcal{NP} \cap$ co-$\mathcal{NP}$. Our result therefore adds further evidence to the idea that [5] the source of the quantum computational speed-up lies in quantum computers being able to act in an atemporal/non-causal manner.

Chapter 10

Conclusion and Further Work

In this document we have presented the concept of a theory machine, and demonstrated how we can use theory machines equipped with various logical systems to characterise different computational devices, both "physical" and "unphysical". We have also argued that any realisable physical computer (including, potentially, a non-causally acting one) should be characterisable by a theory machine acting within a complete logical system (such as first-order logic).

In Chapter 9 we argued that we can describe the resources used by a theory machine computation by considering the minimal domain size of its satisfying models. We can then use this complexity measure to describe the resource usage of any system which is characterised by this theory machine.

However, in Remark 9.0.11 we noted that an issue with this complexity measure is that it does not faithfully capture the resource usage of a probabilistic computation. We suspect that a solution to this issue is to look at the size of the quantifier-free $\mathfrak{LS}$-sentences that completely describe a given $\mathfrak{LS}$-structure. In first-order logic any structure of size N may be described by a quantifier-free sentence of size $O(N)$. However using some form of probabilistic logic [60] we believe that a probabilistic Turing machine computation with time bound N and $O(2^N)$ computation paths may be completely described by a probabilistic sentence of length $O(N)$

We believe that we should also be able to faithfully capture quantum complexity using this modification of theory machine complexity. Indeed we suspect that any polynomial time quantum computation can be non-causally characterised with a polynomially-resourced theory machine that uses a *classical* probabilistic logic. A possible non-causal model for quantum computation that this theory machine could describe is the two-state vector formalism [1, 2].

Another future avenue of research is to look into more general classes of theory machine input and output sets, instead of just words from $\mathcal{A}^*$ or $\mathcal{A}^\omega$. For example one might consider words of ordinal length, such as the kind that the ordinal computers of Koepke and Koerwien [51] can operate on. Indeed we suspect that the class of finite $SO^=$-computable ordinal problems is equal to the constructible universe $\mathbf{L}$, which is class of problems that an ordinal computer is able to decide.

In general we believe that the computational capabilities of any computational system $\mathcal{S}$ may be finitely captured by some logical system $\mathfrak{LS}$. That is, there exists some logical system $\mathfrak{LS}$, for which a finite $\mathfrak{LS}$-theory machine is able to characterise $\mathcal{S}$, and the class of problems computable by $\mathcal{S}$ is equal to the class of finite $\mathfrak{LS}$-computable problems.

We therefore believe that by studying computational systems via theory machines we should be able to gain a clearer understanding of how and why distinct systems differ in their ability to compute and their ability to efficiently compute.

Appendix A

Axioms

“This is a one line proof... if we start sufficiently far to the left.” - Unknown

In this appendix we list various standard sets of axioms that are used to define theory machines in the main body of this document. We also note relevant results for these sets.

A.1 Standard sets of axioms

A.1.1 Axioms for Equality

Definition A.1.1 In a vocabulary $\mathcal{V}$ for each k-ary relation $R \in \mathcal{V}$ and n-ary function $f \in \mathcal{V}$ let:

$$EQ_R^= \equiv \forall x_1 \ldots \forall x_k \forall y_1 \ldots \forall y_k \left(\bigwedge_{i=1}^{k} (x_i = y_i) \rightarrow (R(x_1, \ldots, x_k) \leftrightarrow R(y_1, \ldots, y_k)) \right),$$

$$EQ_f^= \equiv \forall x_1 \ldots \forall x_n \forall y_1 \ldots \forall y_n \left(\bigwedge_{i=1}^{n} (x_i = y_i) \rightarrow (f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)) \right).$$

The $\mathcal{V}$-*equality axioms* [31] for the binary relation $= \in \mathcal{V}$ are then:

$$EQ_{\mathcal{V}}^= = \{EQ_V^= \mid V \in \mathcal{V}\} \cup \left\{ \begin{array}{l} \forall x(x = x), \\ \forall x \forall y (x = y) \rightarrow (y = x), \\ \forall x \forall y \forall z ((x = y) \wedge (y = z)) \rightarrow (x = z) \end{array} \right\}.$$

So $EQ^{=}_{\mathcal{V}}$ states that $=$ is an equivalence relation that preserves the truth of every relation in $\mathcal{V}$, as well as the value of each function assignment.

Remark A.1.2 Let $\mathfrak{LS}$ be a logical system, and $\mathcal{V}$ be a vocabulary containing the binary relation $='$. If $\mathfrak{M} \models_{\mathfrak{LS}_{\mathcal{V}}} EQ^{='}_{\mathcal{V}}$ then it is possible that there may be two elements $p, q \in \mathrm{dom}(\mathfrak{M})$ where $p \neq q$ but $p =' q$ in $\mathfrak{M}$. However in $\mathfrak{M}$ the function and relation assignments of p and q must be identical, so there is no way of knowing within $\mathfrak{M}$ that p and q are in fact distinct.

Indeed there must exist an $\mathfrak{LS}_{\mathcal{V}}$-structure $\mathfrak{M}'$ which embeds into $\mathfrak{M}$ and for any $p, q \in \mathrm{dom}(\mathfrak{M}')$ we have $p =' q$ iff $p = q$.

If $\mathfrak{LS}^{='}$ is a logical system which consists of $\mathfrak{LS}$ together with a true equality relation $='$[1], then $\mathfrak{M}'$ is also an $\mathfrak{LS}^{='}_{\mathcal{V}}$-structure. Hence for any $\mathfrak{LS}_{\mathcal{V}}$-sentence ϕ (which is also an $\mathfrak{LS}^{='}_{\mathcal{V}}$-sentence), ϕ is true in $\mathfrak{M}$ iff ϕ is true in $\mathfrak{M}'$. Therefore for any $\mathfrak{LS}_{\mathcal{V}}$-theory $\mathcal{T}$, we have $\mathcal{T} \cup EQ^{='}_{\mathcal{V}} \models_{\mathfrak{LS}_{\mathcal{V}}} \phi$ iff $\mathcal{T} \models_{\mathfrak{LS}^{='}_{\mathcal{V}}} \phi$.

A.1.2 Axioms for the Natural Numbers

Definition A.1.3 The set of *Peano successor axioms* [56] in the vocabulary of $\{=, S, 0\}$, where $=$ is a binary relation, S is a unary function and 0 is a constant, is:

$$PSA = \left\{ \begin{array}{l} \forall x \neg(S(x) = 0), \\ \forall x \forall y ((S(x) = S(y)) \to (x = y)), \\ \forall x \neg(S(x) = x), \\ \underline{\forall}^1 N((N(0) \wedge (\forall y (N(y) \to N(S(y))))) \to \forall z N(z)) \end{array} \right\}$$

Recall from Definition 3.2.9 that $\underline{\forall}^1 N \phi(N)$ means that we are quantifying over all unary relations.

Proposition A.1.4 *[56] The Peano successor axioms are uniquely $SO^{=}$-modelled by the usual structure of the natural numbers $\langle \mathbb{N}; S, 0 \rangle$.*

[1] Such as $FO^{=}$ or $SO^{=}$ in Subsection 3.2.1.

A.1.3 Axioms for the Integers

Definition A.1.5 The set of *integer successor axioms* [56] in the vocabulary of $\{=, <, S\}$, where $=, <$ are binary relations and S is a unary function, is:

$$ISA = \left\{ \begin{array}{ll} \mathbb{Z}\text{Ax}(1) & \forall x \forall y (S(x) = S(y)) \to (x = y), \\ \mathbb{Z}\text{Ax}(2) & \forall x \exists y (S(y) = x), \\ \mathbb{Z}\text{Ax}(3) & \forall x (x < S(x)), \\ \mathbb{Z}\text{Ax}(4) & \forall x \neg \exists y ((x < y) \wedge (y < S(x)), \\ \mathbb{Z}\text{Ax}(5) & \forall x \neg (x < x), \\ \mathbb{Z}\text{Ax}(6) & \forall x \forall y ((x < y) \to \neg (y < x)), \\ \mathbb{Z}\text{Ax}(7) & \forall x \forall y \forall z ((x < y) \wedge (y < z)) \to (x < z), \\ \mathbb{Z}\text{Ax}(8) & \underline{\forall}^1 Z(((\exists x Z(x)) \wedge (\forall y (Z(y) \leftrightarrow Z(S(y))))) \to \forall z Z(z)), \\ \mathbb{Z}\text{Ax}(9) & \underline{\exists}^1 N(((\exists x N(x)) \wedge (\forall y (N(y) \to N(S(y))) \wedge (\exists z \neg N(z))) \end{array} \right\}.$$

Proposition A.1.6 *Any $SO^=$-model of ISA is isomorphic to the usual ordered structure of the integers $\langle \mathbb{Z}; <, S \rangle$.*

Proof: $\mathbb{Z}$Ax(1), $\mathbb{Z}$Ax(2), $\mathbb{Z}$Ax(8), and $\mathbb{Z}$Ax(9) together with the convention that every function and relation is total, is equivalent to the integer successor axioms given in [56]. Hence every model of ISA with vocabulary $\mathcal{V} = \{<, S\}$ is isomorphic to an expansion of the usual structure of the integers $\langle \mathbb{Z}; S \rangle$ with successor function $S(x) = x + 1$.

Now if $\mathfrak{M} \models_{SO^=_{\mathcal{V}}} ISA$ then by $\mathbb{Z}$Ax(5)-$\mathbb{Z}$Ax(7) $<$ in $\mathfrak{M}$ is a strict total order, whilst $\mathbb{Z}$Ax(3) and $\mathbb{Z}$Ax(4) ensure that $\mathfrak{M} = \mathbb{Z}$ has ordering:

$$\cdots < -2 < -1 < 0 < 1 < 2 < \cdots .$$

Therefore $\mathfrak{M}$ must be isomorphic to the usual ordered structure of the integers $\langle \mathbb{Z}; <, S \rangle$. ❏

A.1.4 Axioms for the Real Numbers

Definition A.1.7 The set of *real arithmetic axioms* [3, 62] in the vocabulary of $\mathcal{V}_{\mathbb{R}} = \{=, <, +, \times, 0, 1\}$, where $=, <$ are binary relations, $+, \times$ are binary functions, and 0,1 are constants, is:

$$RAA = \left\{ \begin{array}{ll} \mathbb{R}\text{Ax}(1) & \forall x \forall y((x+y)=(y+x)), \\ \mathbb{R}\text{Ax}(2) & \forall x \forall y \forall z(((x+y)+z)=(x+(y+z))), \\ \mathbb{R}\text{Ax}(3) & \forall x((x+0)=x), \\ \mathbb{R}\text{Ax}(4) & \forall x \exists y((x+y)=0), \\ \mathbb{R}\text{Ax}(5) & \forall x \forall y((x \times y)=(y \times x)), \\ \mathbb{R}\text{Ax}(6) & \forall x \forall y \forall z(((x \times y) \times z)=(x \times (y \times z))), \\ \mathbb{R}\text{Ax}(7) & \forall x((x \times 1)=x), \\ \mathbb{R}\text{Ax}(8) & \forall x(\neg(x=0) \to \exists y((x \times y)=1)), \\ \mathbb{R}\text{Ax}(9) & \forall x \forall y \forall z(((x+y) \times z)=((x \times z)+(y \times z))), \\ \mathbb{R}\text{Ax}(10) & \forall x \forall y((x<y) \vee (x=y) \vee (y<x)), \\ \mathbb{R}\text{Ax}(11) & \forall x \neg(x<x), \\ \mathbb{R}\text{Ax}(12) & \forall x \forall y((x<y) \to \neg(y<x)), \\ \mathbb{R}\text{Ax}(13) & \forall x \forall y \forall z((x<y) \wedge (y<z)) \to (x<z), \\ \mathbb{R}\text{Ax}(14) & \forall x \forall y \forall z(x<y) \to ((x+z)<(y+z)), \\ \mathbb{R}\text{Ax}(15) & \forall x \forall y(((0<x) \wedge (0<y)) \to (0<(x \times y))), \\ \mathbb{R}\text{Ax}(16) & \begin{array}{l} \underline{\forall}^1 Q((\exists u Q(u) \wedge \exists v \forall w(Q(w) \to (w<v))) \\ \qquad\qquad \to \exists x \forall y(\forall z(Q(z) \to (z<y)) \leftrightarrow (x \leqslant y))) \end{array} \end{array} \right\}.$$

Proposition A.1.8 *[3, 62] Any $SO^=$-model of RAA is isomorphic to the usual structure of real arithmetic $\langle \mathbb{R}; <, +, \times, 0, 1 \rangle$.*

A.1.5 Axioms for the Complex Numbers

Definition A.1.9 The set of *complex arithmetic axioms* [62] in the vocabulary of $\mathcal{V}_{\mathbb{R}} \cup \{\mathbb{R}, i\}$, where $\mathcal{V}_{\mathbb{R}} = \{=, <, +, \times, 0, 1\}$ is as in Definition A.1.7, $\mathbb{R}$ is a unary relation, and i is a constant, is:

$$CAA = RAA^{\mathbb{R}} \cup \mathcal{P}^{\mathbb{R}}_{\mathcal{V}_{\mathbb{R}}} \cup \left\{ \begin{array}{ll} \mathbb{C}\text{Ax}(1) & \forall x \forall y ((x+y) = (y+x)), \\ \mathbb{C}\text{Ax}(2) & \forall x \forall y \forall z (((x+y)+z) = (x+(y+z))), \\ \mathbb{C}\text{Ax}(3) & \forall x ((x+0) = x), \\ \mathbb{C}\text{Ax}(4) & \forall x \exists y ((x+y) = 0), \\ \mathbb{C}\text{Ax}(5) & \forall x \forall y ((x \times y) = (y \times x)), \\ \mathbb{C}\text{Ax}(6) & \forall x \forall y \forall z (((x \times y) \times z) = (x \times (y \times z))), \\ \mathbb{C}\text{Ax}(7) & \forall x ((x \times 1) = x), \\ \mathbb{C}\text{Ax}(8) & \forall x \neg (x = 0) \to \exists y ((x \times y) = 1), \\ \mathbb{C}\text{Ax}(9) & \forall x \forall y \forall z (((x+y) \times z) = ((x \times z) + (y \times z))), \\ \mathbb{C}\text{Ax}(10) & ((i \times i) + 1) = 0, \\ \mathbb{C}\text{Ax}(11) & \forall x \exists y \exists z (x = (y + (z \times i))) \end{array} \right\}.$$

Recall from Definition 6.1.4 that $RAA^{\mathbb{R}}$ denotes the axioms of RAA from Definition A.1.7 sorted by the relation $\mathbb{R}$, so they are only true when $\mathbb{R}$ is true. Whereas as in Definition 6.1.6, $\mathcal{P}^{\mathbb{R}}_{\mathcal{V}_{\mathbb{R}}}$ is the set of $\mathbb{R}$-preservation sentences for $\mathcal{V}_{\mathbb{R}}$, meaning that the subset in which $\mathbb{R}$ is true contains $0, 1 \in \mathcal{V}_{\mathbb{R}}$ and is closed under the $+$ and $\times$ functions.

Proposition A.1.10 *[62] Any $SO^{=}$-model of CAA is isomorphic to the usual structure of complex arithmetic $\langle \mathbb{C}; <, \mathbb{R}, +, \times, 0, 1, i \rangle$.*

When characterising a quantum computer in Examples 7.1.3-7.1.6 we use the following set of axioms to define some well-known functions, relations and constants on $\mathbb{C}$.

Definition A.1.11 The set of *additional complex axioms* in the vocabulary of $\mathcal{V}_{\mathbb{C}} \cup \{\leqslant, \mathbb{N}, -, /, \cdot^2, \sqrt{\cdot}, |\cdot|, 2^\wedge, 2, e^{i\frac{\pi}{4}}\}$ where $\leqslant$ is a binary relation, N is a unary relation, $-, /$ are binary functions, $\cdot^2, \sqrt{\cdot}, |\cdot|, 2^\wedge$ are unary functions, and $2, e^{i\frac{\pi}{4}}$ are constants, is:

$$ACA = \left\{ \begin{array}{ll} \mathrm{Def}(\leqslant) & \forall^{\mathbb{R}}x\forall^{\mathbb{R}}y(x \leqslant y) \leftrightarrow ((x < y) \vee (x = y)), \\ \mathrm{Def}_1(\mathbb{N}) & \mathbb{N}(0), \\ \mathrm{Def}_2(\mathbb{N}) & \forall^{\mathbb{R}}x((0 \leqslant x) \rightarrow (\mathbb{N}(x) \leftrightarrow \mathbb{N}(x+1))), \\ \mathrm{Def}_3(\mathbb{N}) & \forall^{\mathbb{R}}x(((x < 1) \wedge \neg(x = 0))) \rightarrow \neg\mathbb{N}(x)), \\ \mathrm{Def}(-) & \forall x\forall y\forall z(x = (y + z)) \rightarrow ((x - y) = z), \\ \mathrm{Def}(/) & \forall x\forall y\forall z(\neg(y = 0) \wedge (x = (y \times z))) \rightarrow ((x/y) = z), \\ \mathrm{Def}(\cdot^2) & \forall x(x^2 = (x \times x)), \\ \mathrm{Def}(\sqrt{\cdot}) & \forall^{\mathbb{R}}x((0 \leqslant x) \rightarrow ((\sqrt{x} \times \sqrt{x}) = x)), \\ \mathrm{Def}(|\cdot|) & \forall x\forall^{\mathbb{R}}y\forall^{\mathbb{R}}z(x = (y + (z \times i))) \rightarrow (|x| = \sqrt{y^2 + z^2})), \\ \mathrm{Def}_1(2^\wedge) & (2^\wedge(0)) = 1, \\ \mathrm{Def}_2(2^\wedge) & \forall^{\mathbb{N}}x(2^\wedge(x+1) = ((2^\wedge(x)) \times 2)), \\ \mathrm{Def}(2) & 2 = (1 + 1), \\ \mathrm{Def}(e^{i\frac{\pi}{4}}) & ((e^{i\frac{\pi}{4}})^2 = i) \wedge (\forall^{\mathbb{R}}x\forall^{\mathbb{R}}y((x + (i \times y)) = e^{i\frac{\pi}{4}}) \rightarrow (0 < x)) \end{array} \right\}.$$

We denote the additional complex vocabulary by:

$$\mathcal{V}_{\mathbb{C}+} = \{\leqslant, \mathbb{N}, -, /, \cdot^2, \sqrt{\cdot}, |\cdot|, 2^\wedge, 2, e^{i\frac{\pi}{4}}\}.$$

Proposition A.1.12 *If $\mathfrak{B}$ is an $SO^=$-model of $CAA \cup ACA$ then $\mathfrak{B}$ is isomorphic to the usual structure of complex arithmetic $\langle \mathbb{C}; <, \leqslant, \mathbb{R}, \mathbb{N}, +, -, \times, /, \cdot^2, \sqrt{\cdot}, |\cdot|, 2^\wedge, 0, 1, 2, i\rangle$ with:*

- *The less than or equal to relation "$\leqslant$",*
- *The characteristic relation of the natural numbers "$\mathbb{N}$",*
- *The negation function "$-$",*
- *The division function "$/$",*
- *The squaring function "$\cdot^2$",*
- *The square root function "$\sqrt{\cdot}$",*
- *The modulus function "$|\cdot|$",*

- *The powers of 2 function "2^",*
- *The number 2,*
- *The number $e^{i\frac{\pi}{4}}$,*

all defined as usual on dom$(\mathfrak{B}) = \mathbb{C}$*. However if $y = 0$ then x/y may take any value in $\mathbb{C}$. Similarly if $z \notin \mathbb{R}$ or $z < 0$, and if $n \notin \mathbb{N}$ then $\sqrt{z}$ and $2\hat{}(n)$ may have any value in $\mathbb{C}$.*

Proof: By Proposition A.1.10 any $SO^{=}$-model of CAA is an expansion of the usual model of the complex numbers.

Now clearly Def$(\leqslant)$ defines $x \leqslant y$ on $\mathbb{C}$ in its usual manner if $\mathbb{R}(x) \wedge \mathbb{R}(y)$ is true. The same is similarly true for Def$(-)$, Def$(/)$, Def$(\cdot^2)$, Def$(\sqrt{})$, Def$(|\cdot|)$, and Def(2) which define $-, /, \cdot^2, \sqrt{}, |\cdot|$, and 2 respectively.

Def$(/)$ does not define x/y for $y = 0$. However x/y must take some value in $\mathbb{C}$ as every function in an $SO^{=}$-structure is total, so as it is undefined x/y can take any value of $\mathbb{C}$ in $\mathfrak{B}$. The same is true for the undefined values of $\sqrt{z}$ and $2\hat{}(n)$.

The fact that $\mathbb{N}(x)$ is true in $\mathfrak{B}$ for $x \in \mathbb{C}$ iff $x \in \mathbb{N}$ follows by $\text{Def}_1(\mathbb{N}) - \text{Def}_3(\mathbb{N})$ and induction. By $\text{Def}_1(\mathbb{N})$ we have that $\mathbb{N}(0)$ is true in $\mathfrak{B}$, and $\text{Def}_2(\mathbb{N})$ provides the inductive step to ensure that $\mathbb{N}(x)$ is true in $\mathfrak{B}$ if $x \in \mathbb{N}$. Conversely, if $x \in \mathbb{R} \setminus \mathbb{N}$ and $x < 1$ and $x \neq 0$ then by $\text{Def}_3(\mathbb{N})$ we have that $\mathbb{N}(x)$ is false in $\mathfrak{B}$.

The function $2\hat{}$ is defined inductively $\text{Def}_1(2\hat{})$ and $\text{Def}_2(2\hat{})$ to be such that $2\hat{}n = 2^n$ for each natural number $n \in \mathbb{N}$, as by the usual definition of exponentiation $2^0 = 1$ and $2^{x+1} = 2^x \times 2$.

Finally Def$(e^{i\frac{\pi}{4}})$ defines the constant $e^{i\frac{\pi}{4}}$ to have its usual value in $\mathbb{C}$. As if $z^2 = i = e^{i\frac{\pi}{2}}$ then either $z = e^{i\frac{\pi}{4}} = \frac{1}{\sqrt{2}} + \frac{1}{\sqrt{2}}i$ or $z = e^{i\frac{5\pi}{4}} = -\frac{1}{\sqrt{2}} - \frac{1}{\sqrt{2}}i$. So if in addition the real part of z is greater 0, then it must be the case $z = e^{i\frac{\pi}{4}}$. ❏

Corollary A.1.13 *If $\mathfrak{C}$ is an $FO\mathbb{C}$-model of ACA then $\mathfrak{C}$ is isomorphic to the usual structure of complex arithmetic in which the conditions of Proposition A.1.12 hold.*

Proof: By definition, any $FO\mathbb{C}$-structure is an expansion of the usual structure of complex arithmetic, so if $\mathfrak{C}$ is an $FO\mathbb{C}$-model of ACA we may follow the proof of Proposition A.1.12 to show that the conditions of Proposition A.1.12 hold for $\mathfrak{C}$. ❑

A.1.6 Axioms for the Ordinals

As noted in Definition 2.5.3 the ordinal numbers serve as an extension of the natural numbers, and each ordinal number corresponds to the order type of a well-order.

Definition A.1.14 [25, 64] The set of *ordinal successor axioms* in the vocabulary of $\{=, <, L, S, 0\}$, where $=, <$ are binary relations, L is a unary relation, S is a unary function and 0 is a constant, is:

$$OSA = \left\{ \begin{array}{ll} \mathcal{O}\mathrm{Ax}(1) & \forall x \neg(S(x) = 0), \\ \mathcal{O}\mathrm{Ax}(2) & \forall x \forall y (S(x) = S(y)) \to (x = y), \\ \mathcal{O}\mathrm{Ax}(3) & \forall x \neg(S(x) = x), \\ \mathcal{O}\mathrm{Ax}(4) & \forall x (\neg(x = 0) \to (0 < x)), \\ \mathcal{O}\mathrm{Ax}(5) & \forall x (x < S(x)), \\ \mathcal{O}\mathrm{Ax}(6) & \forall x \neg(x < x), \\ \mathcal{O}\mathrm{Ax}(7) & \forall x \forall y ((x < y) \to \neg(y < x)), \\ \mathcal{O}\mathrm{Ax}(8) & \forall x \forall y \forall z ((x < y) \wedge (y < z)) \to (x < z), \\ \mathcal{O}\mathrm{Ax}(9) & \forall x (L(x) \leftrightarrow (\neg \exists y (S(y) = x) \wedge \neg(x = 0))), \\ \mathcal{O}\mathrm{Ax}(10) & \begin{array}{l} \underline{\forall}^1 R(\exists x (R(x))) \\ \qquad \to \exists y (R(y) \wedge \forall z (R(z) \to ((y < z) \vee (y = z)))) \end{array} \end{array} \right\}.$$

$\mathcal{O}\mathrm{Ax}(10)$ is commonly known as the well-ordering axiom.

Proposition A.1.15 *Every $SO^{=}$-model of the ordinal successor axioms is isomorphic to a limit ordinal structure of the form $\langle \mathbb{O}_\lambda; =, <, L, S, 0 \rangle$ where for some limit ordinal λ we have:*

$$\mathbb{O}_\lambda = \{\sigma \in ORD \mid \sigma < \lambda\},$$

and $L(p)$ is true iff $p \in \mathbb{O}_\lambda$ is a limit ordinal.

Proof: Let $\mathfrak{M} \models_{SO^=} OSA$, then $<$ in $\mathfrak{M}$ must be strict total and discrete well-order with least element 0, such that for any $\gamma \in \mathfrak{M}$ the least element of $\mathfrak{M}$ that is greater than γ is $S(\gamma)$. Therefore $\mathfrak{M}$ must be isomorphic to an ordinal structure with domain of the form:

$$\mathbb{O}_\delta = \{\sigma \in ORD \mid \sigma < \delta\},$$

for some ordinal δ. Now by convention any $SO^=_{\mathcal{V}}$-structure is closed under any function in $\mathcal{V}$. So for every $\gamma \in \mathbb{O}_\delta$ the ordinal $S(\gamma)$ must be contained within $\mathbb{O}_\delta$, hence δ cannot be a successor ordinal, and as $\mathbb{O}_\delta$ is non-empty δ must be a limit ordinal.

By $\mathcal{O}$Ax(9), for any $p \in \mathrm{dom}(\mathfrak{M})$, we have $L(p)$ is true iff $p \neq 0$ and there does not exist some $q \in \mathrm{dom}(\mathfrak{M})$ such that $S(q) = p$. So clearly p cannot be 0 or a successor ordinal, hence p must be a limit ordinal. ❑

For our characterisation of an infinite time Turing machine computation in Section 7.2 it is necessary that such a computation does not end prematurely. We avoid this by ensuring that the computation occurs within an uncountable ordinal number of time steps. Hence we have the following definition.

Definition A.1.16 The set of *uncountable ordinal successor axioms* in the vocabulary of $\{=, <, L, S, 0\}$, where $=, <$ are binary relations, L is a unary relation, S is a unary function and 0 is a constant, is:

$$UOSA = OSA \cup \left\{ \begin{array}{ll} \mathcal{UO}\mathrm{Ax} & \neg\exists^1(f\forall x\forall y(\neg(x=y) \rightarrow \neg(f(x)=f(y))) \\ & \qquad\qquad \wedge\ \forall z\forall\delta(L(\delta) \rightarrow (f(z)<\delta))) \end{array} \right\}.$$

Recall from Definition 3.2.9 that $\exists^1 f\phi(f)$ means that we are quantifying over unary functions.

Proposition A.1.17 *Every $SO^=$-model of the uncountable ordinal successor axioms is isomorphic to an uncountable limit ordinal structure.*

Proof: Since $OSA \subset UOSA$, by Proposition A.1.15, every $SO^{=}$-model of $UOSA$ is isomorphic to a limit ordinal structure of the form $\mathfrak{W} = \langle \mathbb{O}_\lambda; =, <, L, S, 0\rangle$ for some limit ordinal λ.

Now $\mathcal{UO}$Ax implies that there cannot exist a function f in $\mathfrak{W}$ such that f is an injection, and $f(z)$ is less than any limit ordinal. ω is the least limit ordinal and the ordinals below it are exactly the numbers contained within $\mathbb{N}$. So if f did exist then it would be an injective mapping from $\mathbb{O}_\lambda$ to $\mathbb{N}$, and so by definition $\mathbb{O}_\lambda$ would be countable. Therefore since such a function f cannot exist, $\mathbb{O}_\lambda$ must be uncountable.

❏

References

[1] Yakir Aharonov, Eliahu Cohen, Eyal Gruss, and Tomer Landsberger. Measurement and collapse within the two-state vector formalism. *Quantum Studies: Mathematics and Foundations*, 1(1-2):133–146, 2014. 158

[2] Yakir Aharonov and Lev Vaidman. The two-state vector formalism: an updated review. In *Time in quantum mechanics*, pages 399–447. Springer, 2008. 158

[3] Christopher Apelian and Steve Surace. *Real and Complex Analysis*. CRC Press, 2009. 161, 162

[4] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009. 16, 22, 142

[5] Ämin Baumeler and Stefan Wolf. Computational tameness of classical non-causal models. In *Proc. R. Soc. A*, volume 474. The Royal Society, 2018. 2, 8, 24, 25, 53, 139, 155

[6] Mark Beck. *Quantum mechanics: theory and experiment*. Oxford University Press, 2012. 28

[7] Cameron Beebe. Model-based computation. In *Unconventional Computation and Natural Computation - 15th International Conference, UCNC 2016, Manchester, UK, July 11-15, 2016, Proceedings*, pages 75–86, 2016. 2, 13, 73

[8] Edwin Beggs, José Félix Costa, Bruno Loff, and John V. Tucker. *Oracles and advice as measurements*, volume 5204 of *Lecture Notes in Comput. Sci.* Springer, Berlin, 2008. 5, 118

[9] Edwin Beggs, José Félix Costa, Diogo Poças, and John V. Tucker. Oracles that measure thresholds: the Turing machine and the broken balance. *J. Logic Comput.*, 23(6):1155–1181, 2013. 5, 118

[10] Edwin Beggs, José Félix Costa, Diogo Poças, and John V. Tucker. An analogue-digital Church-Turing thesis. *International Journal of Foundations of Computer Science*, 25(4):373–389, 2014. 5, 118

[11] Edwin J Beggs, José Félix Costa, and John V Tucker. Limits to measurement in experiments governed by algorithms. *Mathematical Structures in Computer Science*, 20(06):1019–1050, 2010. 5, 118

[12] Edwin J. Beggs, José Félix Costa, and John V. Tucker. The impact of models of a physical oracle on computational power. *Mathematical Structures in Computer Science*, 22(5):853–879, 2012. 5, 118

[13] Edwin J. Beggs and John V. Tucker. Can Newtonian systems, bounded in space, time, mass and energy compute all functions? *Theoretical Computer Science*, 371(1-2):4–19, 2007. 2, 52

[14] Chris Bissell. Historical perspectives-the moniac a hydromechanical analog computer of the 1950s. *IEEE Control Systems*, 27(1):69–74, 2007. 3, 73, 128

[15] Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal logic*, volume 53 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, 2001. 35

[16] Ed Blakey. Unconventional complexity measures for unconventional computers. *Natural Computing*, 10(4):1245–1259, 2011. 12, 117, 140

[17] Edward William Blakey. *A model-independent theory of computational complexity : from patience to precision and beyond*. PhD thesis, University of Oxford, UK, 2010. 2, 12, 72, 76, 128, 149

[18] Lenore Blum, Felipe Cucker, Michael Shub, and Steve Smale. *Complexity and real computation*. Springer Science & Business Media, 2012. 4

[19] Lenore Blum, Mike Shub, Steve Smale, et al. On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. *Bulletin (New Series) of the American Mathematical Society*, 21(1):1–46, 1989. 4

[20] George S Boolos, John P Burgess, and Richard C Jeffrey. *Computability and logic*. Cambridge university press, 2002. 1

[21] Olivier Bournez, Nachum Dershowitz, and Pierre Néron. Axiomatizing analog algorithms. In *Pursuit of the universal*, volume 9709 of *Lecture Notes in Comput. Sci.*, pages 215–224. Springer, [Cham], 2016. 7

[22] Vasco Brattka. The emperors new recursiveness: The epigraph of the exponential function in two models of computability. In *Words, Languages & Combinatorics III*, pages 63–72. World Scientific, 2003. 4

[23] Alonzo Church. A set of postulates for the foundation of logic. *Ann. of Math. (2)*, 33(2):346–366, 1932. 1

[24] Alonzo Church. An unsolvable problem of elementary number theory. *American journal of mathematics*, 58(2):345–363, 1936. 1

[25] Krzysztof Ciesielski. *Set theory for the working mathematician*, volume 39. Cambridge University Press, 1997. 32, 37, 166

[26] Alan Cobham. The intrinsic computational difficulty of functions. *Logic, Methodology and Philosophy of Science: Proceedings of the 1964 International Congress*, pages 24–30, 1965. 139

[27] S. Barry Cooper. *Computability theory*. Chapman & Hall/CRC, Boca Raton, FL, 2004. 1, 2, 16, 21, 114, 117, 122, 135

[28] Christian d'Elbée. On the complete ordered field. *http://choum.net/∼chris /cours_et_notes/reals.pdf*, 2013. 127, 128

[29] David Deutsch. Quantum theory, the Church-Turing principle and the universal quantum computer. In *Proceedings of the Royal Society of London A:*

Mathematical, Physical and Engineering Sciences, volume 400, pages 97–117. The Royal Society, 1985. 1, 117, 118

[30] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17(3):449–467, 1965. 139

[31] Richard L Epstein. *Classical mathematical logic: the semantic foundations of logic*. Princeton University Press, 2011. 35, 57, 159

[32] John Friedman, Michael S Morris, Igor D Novikov, Fernando Echeverria, Gunnar Klinkhammer, Kip S Thorne, and Ulvi Yurtsever. Cauchy problem in spacetimes with closed timelike curves. *Physical Review D*, 42(6):1915, 1990. 8

[33] Minoru Fujimoto. *Physics of classical electromagnetism*, volume 240. Springer Science & Business Media, 2007. 73, 76

[34] Jean H Gallier. *Logic for computer science: foundations of automatic theorem proving*. Courier Dover Publications, 2015. 48, 124

[35] Robin Gandy. Church's thesis and principles for mechanisms. *Studies in Logic and the Foundations of Mathematics*, 101:123–148, 1980. 118

[36] Kurt Gödel. Die vollständigkeit der axiome des logischen funktionenkalküls. *Monatshefte für Mathematik*, 37(1):349–360, 1930. 50, 124, 129, 130, 134

[37] Kurt Gödel. Über formal unentscheidbare sätze der principia mathematica und verwandter systeme i. *Monatshefte für mathematik und physik*, 38(1):173–198, 1931. 122

[38] Kurt Gödel. An example of a new type of cosmological solutions of Einstein's field equations of gravitation. *Reviews of modern physics*, 21(3):447, 1949. 8

[39] Joseph A. Goguen and José Meseguer. Order-sorted algebra i: Equational deduction for multiple inheritance, overloading, exceptions and partial operations. *Theor. Comput. Sci.*, 105(2):217–273, November 1992. 90

[40] Stanley I Grossman. *Multivariable calculus, linear algebra, and differential equations*. Academic Press, 2014. 75

[41] Yuri Gurevich. Logic in computer science column. *Bulletin of the EATCS*, 35:71–81, 1988. 6

[42] Yuri Gurevich. Sequential abstract-state machines capture sequential algorithms. *ACM Transactions on Computational Logic*, 1(1):77–111, 2000. 2, 6, 7

[43] Joel David Hamkins. A survey of infinite time Turing machines. In *International Conference on Machines, Computations, and Universality*, pages 62–71. Springer, 2007. 4, 33

[44] Joel David Hamkins and Andy Lewis. Infinite time Turing machines. *The Journal of Symbolic Logic*, 65(2):567–604, 2000. 4, 32, 33, 34, 110, 114, 117

[45] Leon Henkin. The completeness of the first-order functional calculus. *The journal of symbolic logic*, 14(3):159–166, 1949. 50, 124, 129, 130, 134

[46] Wilfrid Hodges. *Model theory*. Cambridge University Press, 1993. 35, 41, 43, 121

[47] Roger A. Horn and Charles R. Johnson. *Topics in Matrix Analysis*. Cambridge University Press, 1991. 28

[48] Clare Horsman, Susan Stepney, Rob C Wagner, and Viv Kendon. When does a physical system compute? In *Proceedings of the Royal Society A*, volume 470, page 20140182. The Royal Society, 2014. 6, 10, 51, 53, 87, 119

[49] John David Jackson. *Classical electrodynamics*. John Wiley & Sons, 2012. 73, 76, 77

[50] Alexei Yu Kitaev, Alexander Shen, and Mikhail N Vyalyi. *Classical and quantum computation*. Number 47. American Mathematical Soc., 2002. 3, 28, 29, 30, 89, 99, 117, 142

[51] Peter Koepke and Martin Koerwien. Ordinal computations. *Math. Structures Comput. Sci.*, 16(5):867–884, 2006. 5, 158

[52] Peter Koepke and Benjamin Seyfferth. Towards a theory of infinite time Blum-Shub-Smale machines. In *CiE*, pages 405–415. Springer, 2012. 4, 5

[53] Saul A. Kripke. The Church-Turing thesis as a special corollary of Gödel's completeness theorem. In B. J. Copeland, C. Posy, and O. Shagrir, editors, *Computability: Turing, Gödel, Church, and Beyond*. MIT Press, 2013. 131

[54] Azriel Levy. *Basic set theory*. Springer-Verlag, 1979. 33

[55] L.D Lifshitz, Landau. *Fluid Mechanics. Course of Theoretical Physics*. Pergamon,Oxford, 1987. 2, 73

[56] Angelo Margaris. Successor axioms for the integers. *The American Mathematical Monthly*, 68(5):441–444, 1961. 121, 160, 161

[57] István Németi and Gyula Dávid. Relativistic computers and the Turing barrier. *Applied Mathematics and Computation*, 178(1):118–142, 2006. 5

[58] Allen Newell and Herbert Simon. The logic theory machine–a complex information processing system. *IRE Transactions on information theory*, 2(3):61–79, 1956. 5

[59] Michael A. Nielsen and Isaac L. Chuang. *Quantum computation and quantum information*. Cambridge University Press, Cambridge, 2000. 3, 28, 29, 30, 99, 117, 139, 142, 155

[60] Nils J Nilsson. Probabilistic logic. *Artificial intelligence*, 28(1):71–87, 1986. 157

[61] Abraham Robinson. *Non-standard analysis: Studies in logic and the foundations of mathematics*. Princeton University Press, Princeton, NJ. Reprint of the second (1974) edition, With a foreword by Wilhelmus AJ Luxemburg, 1966. 128

[62] Eric Schmidt. Reductions in norman megills axiom system for complex numbers. 161, 162, 163

[63] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, 1997. 3, 9, 12, 31, 139

[64] Wacław Sierpiński. *Cardinal and ordinal numbers*, volume 34. Państwowe Wydawn. Naukowe, 1958. 32, 166

[65] Michael Sipser. *Introduction to the Theory of Computation*, volume 2. Thomson Course Technology Boston, 2006. 16, 18, 19, 89

[66] Cesare Tinelli and Calogero G. Zarba. Combining decision procedures for sorted theories. In *Logics in Artificial Intelligence, 9th European Conference, JELIA 2004, Lisbon, Portugal, September 27-30, 2004, Proceedings*, pages 641–653, 2004. 90

[67] Alan Mathison Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London mathematical society*, 2(1):230–265, 1937. 1, 16, 51, 69, 117, 118, 135

[68] Bernd Ulmann. *Analog computing*. Walter de Gruyter, 2013. 13

[69] Leslie G Valiant. Relative complexity of checking and evaluating. *Information processing letters*, 5(1):20–23, 1976. 24

[70] Anastasios Vergis, Kenneth Steiglitz, and Bradley Dickinson. The complexity of analog computation. *Mathematics and computers in simulation*, 28(2):91–113, 1986. 12, 139

[71] Klaus Weihrauch. Type 2 recursion theory. *Theoretical Computer Science*, 38:17–33, 1985. 4, 135

[72] Klaus Weihrauch. *Computable analysis*. Springer-Verlag, Berlin, 2000. 4, 25, 27, 117, 135

[73] Richard Whyman. Physical computation, p/poly and p/log. In *Proceedings of the 7th International Workshop on Physics and Computation, PC 2016, Manchester, UK, 14 July 2016.*, pages 41–52, 2016. 5

[74] Richard Whyman. An atemporal model of physical complexity. In *Electronic Proceedings in Theoretical Computer Science*, volume 273, pages 39–51, 07 2018. ii, 6, 140

[75] Richard Whyman. Physical computation and first-order logic. In *Machines, Computations, and Universality - 8th International Conference, MCU 2018, Fontainebleau, France, June 28-30, 2018, Proceedings*, pages 153–169, 2018. ii, 6, 114, 115, 120

[76] Nicholas Young. *An introduction to Hilbert space.* Cambridge university press, 1988. 28

Index