mathematics

MDPI

Article

Computation and Hypercomputation

Andrew Powell

Institute for Security Science and Technology, Imperial College, South Kensington Campus, London SW7 2AZ, UK; andrew.powell@imperial.ac.uk

Abstract: This paper shows some of the differences and similarities between computation and hypercomputation, the similarities relating to the complexity of propositional computation and the differences being the propositions that can be decided computationally or hypercomputationally. The methods used are ordinal Turing machines with infinitely long programs and diagonalization out of computing complexity classes. The main results are the characterization of inequalities of run time complexities of serial, indeterministic serial and parallel computers and hypercomputers and the specification of a hierarchy of hypercomputers that can hypercompute the truths of all propositions in the standard class model of set theory, the von Neumann hierarchy of pure sets.

Keywords: computation; diagonalization; hypercomputation; recursion theory; von Neumann hierarchy of pure sets

MSC: 03D15; 03D55; 03D60

1. Introduction

This paper sets out to characterize computations and hypercomputations as trees. Both computers and hypercomputers are defined in this paper, and the difference between them is that a computer has a program of finite length, has finitely many non-empty registers to store data and a computation will only be successful if the program runs for a finite time. At least one of these assumptions is false for a hypercomputer: in the existing literature, a cell-based computer with an infinite run time is called an *infinite run time Turing machine* (see Hamkins and Lewis [1]), a register-based computer with an infinite run time is called an *infinite time register machine* (see Koepke [2], Koepke and Miller [3], a cell-based computer with an infinite run time and an infinite number of non-empty cells on a tape but with a finite program is called an *ordinal Turing machine* (see Koepke [4], Koepke and Koerwien [5]), while a register-based computer with an infinite run time and an infinite number of non-empty registers but with a finite program is called an *ordinal register machine* (see Koepke and Siders [6]). (Strictly, ordinal machines have a tape length or number of registers which is the proper class of all ordinals, but are such that any computation uses only finitely many ordinals.) There is now a large amount of literature on hypercomputation due to (in alphabetical order) Carl, Hamkins, Koepke, Welch and others, see Hamkins and Lewis [1], Koepke [2,4], Koepke and Koerwien [5], Koepke and Siders [6], Carl et al. [7,8], Carl [9,10], Koepke [11], Hamkins and Miller [12], Welch [13,14], Blanchetti [15], Rin [16], Welch [17]. The main focus of these papers is to characterize the strength of the propositions that can be decided hypercomputationally by different types of hypercomputers, although hypercomputation can also be regarded as a putative alternative foundation for a particular formal theory, such as admissible recursion theory, see Koepke and Seyfferth [18]. In addition, a literature work looking at hypercomputation of computational complexity classes, such as the polynomial space and time classes and the non-deteministic polynomial space and time classes (see Schindler [19], Hamkins and Welch [20], Deolalikar et al. [21], Carl [22]) is relevant to this paper because this paper contains results on hypercomputa-

check for updates

Citation: Powell, A. Computation and Hypercomputation. *Mathematics* **2022**, *10*, 997. https://doi.org/10.3390/math10060997

Academic Editor: Christos G. Massouros

Received: 15 February 2022
Accepted: 15 March 2022
Published: 20 March 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

tional complexity classes, which are generalizations of existing results about computational complexity classes.

This paper, unlike those just cited, allows hypercomputers with programs of infinite length. Allowing programs of infinite length produces very powerful hypercomputers, sufficiently powerful to decide the truth of any proposition of second-order Zermelo–Fraenkel set theory (ZF^2 for short, see Hellman [23] for an overview of standard set models of ZF^2) that does not involve proper classes, and therefore suffices to decide set membership for any set in the von Neumann Hierarchy of pure sets, V. Although it is not possible in general, even in principle, to implement a general purpose hypercomputer, hypercomputers are very useful for characterizing the computational strength required to realize certain mathematical systems. It is not the purpose of this paper to dispute the Church–Turing thesis that all computable algorithms can be programmed on a Turing machine (i.e., a finite but unbounded computer), but if computability defines an upper bound for what we mean by finitary methods in mathematics, then propositions that can only be decided by hypercomputational methods are fundamentally infinitary in nature.

It is well known that any total computation (that always returns a result no matter what the input) can be expressed as a branch or set of branches of a rooted decorated tree of finite length, where each branch represents a transition between states (which could keep the same state and could represent a *line number*, *command* or *loop*). We can say that a state label decorates the nodes of the tree, where the state label can be replaced by a "snapshot" of the state of the computer executing a program. (The term "snapshot" is due to Hamkins and Lewis [1].) Similarly, a hypercomputation can be expressed as a branch or set of branches of a rooted decorated tree of infinite length.

Two different approaches to hypercomputation are described: one where a branch of infinite length represents an ω-sequence of values, and the other where hypercomputations are extended to branches of transfinite length. Transfinite lengths are possible if the registers are left in a consistent condition at limit ordinals during computations. The latter approach is in general much more powerful but being able to reason about ω-sequences as completed infinities is simpler (although we do have the ability to use the limit value in one branch as input to another branch, so can represent the transfinite). In the following definitions, we split out the number of registers, the length of computations and the length of the program as separate parameters.

2. Materials and Methods

This paper contains some cardinal and ordinal arithmetic, so it is worth saying how cardinals and ordinals are treated. Cardinals are used to answer "how many" questions (of cardinality), while ordinals are used for order dependent questions such as counting, coding and enumeration. As is standard, here all cardinals are ordinals, finite cardinals and ordinals are identical, and infinite cardinals are the least ordinal of that cardinality. Thus, ordinal ω, the least infinite ordinal, is identified with $\aleph_0$, the first infinite cardinal. In this paper, only cardinal exponentiation is used because we want to compute how many computation paths there are. On the other hand, addition and multiplication here are ordinal addition and multiplication if we are considering an ordering of computation paths or an enumeration. Where cardinal addition or multiplication is intended, we will write $|\alpha|$ to indicate that α and any operations on it should be treated as cardinal operations. Finally, to make treatment of sets easier from an ordinal perspective, the Axiom of Choice will be assumed in the operations that can be performed by hypercomputers.

This paper uses classical methods of recursion theory such as *diagonalization* to define hypercomputational variants of complexity classes and builds on the literature on hypercomputation. The methods used are often "brute force" because the underlying hypercomputer is extremely powerful. Using infinitely long programs, it is possible to form structures by direct construction, but also to construct functions by transfinite recursion along a well-ordered set of any order type α and from any initial function and any iterator function (whether finitely defined or otherwise). Whilst enumeration of an infinite set can

be constructed by using a finitely long program that proceeds by transfinite recursion and calls out to the Axiom of Choice to choose a member of the set that has not been chosen so far and to add it to a well-order, choice functions provided by the Axiom of Choice are not finitely defined. The use of Axiom of Choice to provide a choice function as the iterator function can be eliminated by making the choice of members "hard coded" in the enumeration in the program itself if the program is sufficiently infinitely long. Since the Axiom of Choice is used as an oracle in computing terms, finite programs plus an oracle are an alternative to allowing infinitely long programs.

Infinitely long programs are a key method of this paper. For a serial hypercomputer where only one instruction can be executed at a time, infinite programs are treated using the limit of the supremum of the states at the limit time. This is possible because the program state is a "line number" of a sort which will execute when the state is reached. In some cases, the limit state will be one of finitely many states, but in other cases a state represented by an infinite ordinal will be required. For a parallel hypercomputer, effectively a number of serial programs can be executed at one time, which only makes sense if the programs are independent of one another. To address the concern that infinite programs are less true to computing than an infinitely long tape (number of registers) or infinite run time, in the author's opinion, allowing an infinitely long program is as well formed in terms of the rules at limits and is not conceptually different to an infinitely long tape or infinite run time. Without infinitely long programs, the approach would reduce to that of ordinal Turing machines with bounds on the number of calls on the tape, the number of states and the run time.

Quantifiers can be viewed as infinite conjunctions "and" (for "for all") and infinite disjunctions "or" (for "there exists") and truth predicates computed by enumerating the domain of quantification. All of the hypercomputations of truth values of propositions in this paper are no more than hypercomputations of the standard Tarski semantics applied to the von Neumann hierarchy of pure sets, V. It is possible to perform logical operations (including quantifiers) by means of a finite program provided that there are sufficiently many registers to hold each value of the domain of quantification and the computation is sufficiently long to evaluate the truth of a quantifier-free predicate for each member of the domain of quantification and to perform logical operations on those truth-values. Using a finite program for conjunction, c say, and a program p that hypercomputes the truth value of formula P on input n, we can compute the truth of $(\forall x \in 2^2)P(x)$, where 2^2 here is the set of all binary sequences of length 2, by hypercomputing the truth values of $P(\langle 0,0\rangle)$, $P(\langle 0,1\rangle)$, $P(\langle 1,0\rangle)$, $P(\langle 1,1\rangle)$ in some order, by running p 4 times, and then running c to take the conjunction of truth values. Of course, this is a very inefficient way to hypercompute the truth of $(\forall x \in 2^2)P(x)$, but the call to an enumeration function is no longer a call to an oracle, rather to a particular well-ordering. Moreover, essentially the same approach will hypercompute the truth value of $(\forall x \in 2^{\aleph})P(x)$, where $2^{\aleph}$ is the set of all binary sequences of length cardinal $\aleph$ for any $\aleph$, with some handling of limit ordinals as hypercomputing limits.

In summary, using an infinitely long program with sufficiently many registers to hold each value of the domain of quantification and a sufficiently long computation length, it possible to evaluate the truth of any proposition about sets that does not involve proper classes.

There is no data associated with this paper because the results are purely theoretical and used only for classifying the strength of hypercomputers.

3. Results

3.1. Definitions of Hypercomputing

Definition 1. *A $\langle \aleph, \beth, \daleth \rangle$-hypercomputer, for cardinals $\aleph$ and $\beth$ and ordinal $\daleth$, where $\beth \leq \daleth \leq \aleph$, comprises the following elements:*

- *$\aleph$-many Registers for storage of inputs, outputs and workings of a computation. For ease of exposition there will be disjoint sets of registers for inputs, outputs and workings. Input registers are read-only and contain inputs in the hypercomputer's initial state. Separate*

input, working and output registers are not essential, as registers can always be moved around and working space created, but I hope their use makes the exposition easier to follow. Working registers are read-write and receive a copy of the inputs when the program starts. Output registers receive a copy of the content of the working registers, are write-only by the program and contain the outputs of the program in the hypercomputer's halting state (see below). A register consists of an ordinal identifier and a data field, written R_α for $\alpha < \aleph$, which can contain 0 or 1. By default, all registers are initialized with the value 0 (representing "empty"). Input registers will be written I_α, working registers W_α, and output registers O_α. It is convenient to allow multiple disjoint sets of working registers, $W_{\beta,\alpha}$, to facilitate operations on data set and it will be assumed in this paper that working registers are partitioned into disjoint sets. Disjoint sets of registers can be reproduced by coding the set of disjoint sequences $\{\langle a_{1,\alpha}, a_{2,\alpha} \cdots, a_{i<\aleph,\alpha}, \cdots\rangle : \alpha < \aleph\}$ by the concatenation $\langle a_{1,1}, a_{2,1}, \cdots, a_{i<\aleph,1}, \cdots\rangle\langle a_{1,2}, a_{2,2}, \cdots, a_{i<\aleph,2}, \cdots\rangle \cdots \langle a_{1,\alpha}, a_{2,\alpha}, \cdots, a_{i<\aleph,\alpha}, \cdots\rangle \cdots$. This coding depends on each sequence having a fixed length, $\aleph$ for each sequence of registers in this case. To avoid complexities associated with the computability of functions that jump between registers, registers perform like infinite linear tapes of length $\aleph$ terminated on the left, with R_1 being the register with lowest ordinal and only registers $R_{\alpha+1}$ and $R_{\alpha-1}$, where they exist, being accessible from R_α. $W_{0,0}$ is treated as a special register as it is set to 0 by default and set to 1 if a program (or subprogram) runs to completion, after $o(\aleph)$ steps, where $o(\aleph)$ is the least ordinal of cardinality $\aleph$. This register can be used as a "flag" to capture the output of the program;

- *Symbols 0* and *1*;
- $\beth$-many *States* which determine which action the hypercomputer takes and any output it produces. A state can be identified by an ordinal, which is general will not have a finite notation. There are at least two special states, an *initial state,* identified by the ordinal *0*, where a program (see below) starts and a *halting state* where a program stops. The hypercomputer enters the halting state, i.e., stops, when none of the instructions (see below) applies, when an instruction is executed that puts the program in the halting state, or when the computation maximum length is reached (when the contents of $W_{0,0}$ are set to 1). Ordinary states are like line numbers in a hypercomputer program (see Koepke [4]), so, from the initial state, the program will enter the first ordinary state, *1* say, and as the number of instructions executed (i.e., the length of the computation) increases towards limit ordinal α, the program jumps to state α unless there is a state with a smaller least upper bound. (It is of course possible to become stuck in a particular state and for the program not to output given a particular set of register values, but equally it is possible to loop back to the same state if the register value is *0* say, and then at the next limit ordinal for the program to read a *1*, when the program may move to a different state.) It makes sense not to be able to jump past a limit ordinal, so, for successor ordinal state α, only states with ordinal $prevlim(\alpha) \leq \beta < nextlim(\alpha)$ are accessible from α, where $prevlim(\alpha)$ is the preceding limit ordinal $\leq \alpha$ and $nextlim(\alpha)$ is the next limit ordinal $> \alpha$;
- An *initial configuration*, comprising data loaded into the input registers, an initial state and an initial current register (I_1 by default and likewise $W_{\beta,1}$ and O_1 when these sets of registers are accessed);
- A *program* of length $\beth$, which is a (in general transfinite) sequence of 5-tuples $\langle$*Current State, Register Set, Symbol, Action, Next, State*$\rangle$, called *program instructions*, read as "if the hypercomputer is in Current State and the current register in the Register Set contains Symbol then do Action and move into Next, State", where an Action may be to do nothing, write a 0 or 1 to a current register, R_α, in any set of registers, to move left or right where possible, i.e., from R_α to $R_{\alpha-1}$ or $R_{\alpha+1}$ if α is a successor ordinal and from R_α to $R_{\alpha+1}$ otherwise, or set the current register to the 0-th register, i.e., R_0. The instructions can be grouped by state into a table of instructions. For ease of exposition, the program length will refer to the number of state entries in

the table. As these operations apply to each disjoint set of registers, I, W_β, O, there are 11 instruction types (as "do nothing" applies to all registers and I cannot be written to). For definiteness, "do nothing" is represented by 0, "write a 0" to the current register of W_β by $\langle 1, \beta \rangle$, "write a 0" to the current register of O by 5, "write a 1" to the current register of W_β by $\langle 2, \beta \rangle$, "write a 1" to the current register of O by 6, "move left" by 3 (for I), $\langle 7, \beta \rangle$ (for W_β) and 9 (for O), "move right" by 4 (for I), $\langle 8, \beta \rangle$ (for W_β) and 10 (for O), and "reset register" by 11 (for I), $\langle 12, \beta \rangle$ (for W_β) and 13 (for O). Each program comprises a standard introduction which copies the input registers to working registers (i.e., a set of 5-tuples with source set of registers I and destination set of registers W_1), a program that manipulates the working registers, and a standard conclusion which copies working registers to output registers (i.e., a set of 5-tuples with source registers W_β and destination registers O). A program to copy the registers from W_β and destination registers O has one ordinary state, 1, and comprises the instructions $\langle 0, W_\beta, 0, 0, 1 \rangle$, $\langle 0, W_\beta, 1, 0, 1 \rangle$, $\langle 1, W_\beta, 0, 5, 1 \rangle$, $\langle 1, W_\beta, 1, 6, 1 \rangle$, $\langle 1, W_\beta, 0, \langle 8, \beta \rangle, 1 \rangle$, $\langle 1, W_\beta, 1, \langle 8, \beta \rangle, 1 \rangle$, $\langle 1, W_\beta, 0, 9, 1 \rangle$, $\langle 1, W_\beta, 1, 9, 1 \rangle$. The sequence $\langle 1, W_0, 0, \langle 12, 0 \rangle, 1 \rangle, \langle 1, W_0, 1, \langle 12, 0 \rangle, 1 \rangle, \langle 1, W_0, 1, 0, 2 \rangle$ will move the computation to the halting state, 2, when it completes copying. A program to copy the registers from I and destination W_1 has one ordinary state, 1, and comprises the instructions $\langle 0, I, 0, 0, 1 \rangle$, $\langle 0, I, 1, 0, 1 \rangle$,$\langle 1, I, 0, \langle 1, 1 \rangle, 1 \rangle$, $\langle 1, I, 1, \langle 2, 1 \rangle, 1 \rangle$, $\langle 1, I, 0, 4, 1 \rangle$, $\langle 1, I, 1, 4, 1 \rangle$, $\langle 1, I, 0, \langle 8, 1 \rangle, 1 \rangle$ $\langle 1, I, 1, \langle 8, 1 \rangle, 1 \rangle$. The sequence $\langle 1, W_0, 0, \langle 12, 0 \rangle, 1 \rangle, \langle 1, W_0, 1, \langle 12, 0 \rangle, 1 \rangle, \langle 1, W_0, 1, 0, 2 \rangle$ will move the computation to the halting state, 2, when it completes copying. It is not possible for humans to write down infinitely long programs, but it is possible to write *program schemas*. An example is a program schema for the logical conjunction of a set of registers of cardinality $\aleph$ given by a finite program could be written $\langle 1, W_1, 1, \langle 2, 2 \rangle, 1 \rangle$, $\langle \alpha, W_1, 1, \langle 8, 1 \rangle, \alpha + 1 \rangle$, $\langle \alpha, W_1, 0, \langle 1, 2 \rangle, o(\aleph) \rangle$, where $\alpha < o(\aleph)$ is an ordinal parameter for the state and $o(\aleph)$ is the halt state. Program schemas are concise, but finite programs suffice in the example of logical conjunction (the state number does not have to be increased every time the head moves right). For example, take a program which has two states (other than the standard special states), 1 and 2, the standard introduction and conclusion for input and output being ignored for simplicity. In state 1, if the program reads a register $W_{1,\alpha}$ containing a 1, it writes a 1 in register $W_{2,1}$ and stays in state 1. In state 1, if the program reads a register $W_{1,\alpha}$ containing a 1, it moves right to $W_{1,\alpha+1}$ and stays in state 1, while, if $W_{1,\alpha}$ contains a 0, it writes a 0 in register $W_{2,1}$ and terminates by moving to the halting state, 2. When reading registers $W_{1,\lambda}$ with limit ordinal λ, the program will be in the highest state achieved (i.e., 1 in practice) when reading registers $W_{1,\alpha<\lambda}$ and the value of any register $W_{\beta,\alpha\leq\lambda}$ after limit ordinal λ steps of the program will be the value of an eventually constant sequence $W_{\beta,\alpha}$ for $< \lambda$ steps or 1, otherwise. It can be seen that $W_{2,1}$ contains 1 if and only if every $W_{1,\alpha}$ for ordinal $\alpha < \aleph$ contains 1. The program implements infinite logical conjunction (i.e., infinite logical "and") of propositions with truth values stored in $W_{1,\alpha}$. This program can be written formally as follows in the notation of this paper: $\langle 1, W_1, 1, \langle 2, 2 \rangle, 1 \rangle$, $\langle 1, W_1, 1, \langle 8, 1 \rangle, 1 \rangle$, $\langle 1, W_1, 0, \langle 1, 2 \rangle, 2 \rangle$, $\langle 1, W_0, 0, \langle 12, 0 \rangle, 1 \rangle$, $\langle 1, W_0, 1, \langle 12, 0 \rangle, 1 \rangle$, $\langle 1, W_0, 1, 0, 2 \rangle$. The last three instructions implement the flag set to 1 in $W_{0,0}$ when the program completes, and moves the program to the halt state. Infinite "or" can be done similarly with the two state machine: in state 1, if the program reads a register $W_{1,\alpha}$ containing a 0 it writes a 0 to $W_{2,1}$, moves to $W_{1,\alpha+1}$ and stays in state 1; if it reads a register $W_{1,\alpha}$ containing a 1 it writes a 1 to $W_{2,1}$ and moves to halting state 2. It is also worth explaining how *transfinite recursion* such as $Fgf0 := f$ and $Fgf\beta := g\beta(\lambda\gamma < \beta)Fgf\gamma$, where λ is abstraction in the lambda calculus, for ordinal $\beta < \alpha$ for arbitrary ordinal α if f, g are hypercomputable functions, can be implemented. If we assume that F uses (a set of states including) state 1, f uses (a set of states called by state 1 including) state 2 and g uses (a set of states called by state 1 including) state 3, we can represent a transfinite recursion for F by using one working register set, W_1 say, as a counter (up to cell α, preset to *1* on the input tape), a second set, W_2 say, to contain (a binary code) of the

output to f, a third set, W_3 say, to contain the output to g, and a fourth set of registers, W_4 say, to contain the output of F. Aside from coding and decoding from the binary representation on the tape, the program works until the counter hits the α-th cell when it copies the output of W_4 to the output tape (in state 1), starting by copying f from the input, incrementing the counter, copying those values in W_2 to the input of g (state 2), running g, copying the output of g in W_3 to the input of g, appending the output of g at the end of the output of F in W_4, and incrementing the counter, at limits copying the output of F in W_4 to the input of g in W_3 (state 3). To enable markers (1s) to be put at the end of outputs, we can code the working registers by leaving every second cell of 0 value and marking a 0 marker cell with a 1 when the end of the output is reached;
- $\leq \daleth$ many *steps* in the computation;
- *Output* is the contents of the output registers when the program is in a halting state.

A hypercomputer will read a program, which will start in the initial state, run through its computation and terminate when it reaches a halting state. The output of the program is the contents of the hypercomputer's output registers.

Definition 2. *A hypercomputation is a sequence of steps of length* $\daleth$ *that results in output given a specific input.*

To make this characterization precise, a hypercomputation can be considered to take place in discrete time intervals indexed by ordinals. Following Koepke [4], a "step" can be taken to have three components: the current state at time α, written $S_\alpha(R)$, a pointer to the ordinal index of the current register, $H_\alpha(R)$, and the contents of all the registers (a "snapshot" of the computation), $C_\alpha(R) : \daleth \to \{0,1\}$, where R is a set of registers I, W_β, O. Limit ordinal "steps" are special, as the principle (see Hamkins and Lewis [1]) will be adopted that, if $S_\alpha(R)$, $H_\alpha(R)$ or $C_\alpha(R)$ are eventually constant for $\alpha < \lambda$, where λ is a limit ordinal, then, by default $S_\lambda(R)$, $H_\lambda(R)$ or $C_\lambda(R)(\zeta)$ for $\zeta < \aleph$ will take those constant values or else will take the limit of the least upper bounds, which will be $C_\lambda(R)(\zeta) = 1$ if $C_{\alpha<\lambda}(R)(\zeta)$ is not eventually constant and $S_\lambda(R) = \lambda$ and $H_\lambda(R) = R_\lambda$ if $S_\lambda(R)$ and $H_\lambda(R)$ are otherwise unbounded. This is the "lim sup" construction (i.e., the limit of the least upper bounds). Recursive definitions for $S_\alpha(R)$, $H_\alpha(R)$ and $C_\alpha(R)$ are given as follows (again based on Koepke [4]). Koepke uses "lim inf" rather than "lim sup" because the programs he considers are finite, and it makes no sense to jump to an infinite limit ordinal state.

If $\langle \beta, R, b, a, \gamma \rangle$ is the instruction such that $S_\alpha(R) = \beta$ and $C_\alpha(R)(H_\alpha) = b$, then:
- $S_0(R) = 0$;
- $S_{\alpha+1}(R) = \gamma$ where $prevlim(\alpha) \leq \gamma < nextlim(\alpha)$;
- $H_0(R) = 0$;
- $H_{\alpha+1}(I) = H_\alpha(I) - 1$ if $a = 3$ and $H_\alpha(I)$ is a successor ordinal;
- $H_{\alpha+1}(W_\beta) = H_\alpha(W_\beta) - 1$ if $a = \langle 7, \beta \rangle$ and $H_\alpha(W_\beta)$ is a successor ordinal;
- $H_{\alpha+1}(O) = H_\alpha(O) - 1$ if $a = 9$ and $H_\alpha(O)$ is a successor ordinal;
- $H_{\alpha+1}(I) = H_\alpha(I) + 1$ if $a = 4$;
- $H_{\alpha+1}(W_\beta) = H_\alpha(W_\beta) + 1$ if $a = \langle 8, \beta \rangle$;
- $H_{\alpha+1}(O) = H_\alpha(O) + 1$ if $a = 10$;
- $H_{\alpha+1}(I) = 0$ if $a = 11$;
- $H_{\alpha+1}(W_\beta) = 0$ if $a = \langle 12, \beta \rangle$;
- $H_{\alpha+1}(O) = 0$ if $a = 13$;
- $H_{\alpha+1}(R) = H_\alpha(R)$ otherwise;
- $C_0(I)(\zeta) = I_\zeta$ for all $\zeta < \aleph$;
- $C_{\alpha+1}(W_\beta)(\zeta) = 0$ if $a = \langle 1, \beta \rangle$ and $\zeta = H_\alpha(W_\beta)$;
- $C_{\alpha+1}(O)(\zeta) = 0$ if $a = 5$ and $\zeta = H_\alpha(O)$;
- $C_{\alpha+1}(W_\beta)(\zeta) = 1$ if $a = \langle 2, \beta \rangle$ and $\zeta = H_\alpha(W_\beta)$;
- $C_{\alpha+1}(O)(\zeta) = 1$ if $a = 6$ and $\zeta = H_\alpha(O)$;

- $C_{\alpha+1}(\zeta) = C_\alpha(\zeta)$ otherwise for all $\zeta < \aleph$;
- $S_\lambda(R) = \limsup_{\alpha\to\lambda} S_\alpha(R)$ if λ is a limit ordinal;
- $H_\lambda(R) = \limsup_{\alpha\to\lambda} H_\alpha(R)$ if λ is a limit ordinal;
- $C_\lambda(R)(\zeta) = \limsup_{\alpha\to\lambda} C_\alpha(R)(\zeta)$ if λ is a limit ordinal.

Definition 3. *A serial $\langle \aleph, \beth, \daleth \rangle$-hypercomputer, for cardinals $\aleph$, $\beth$ and ordinal $\daleth$, where $\beth \leq \daleth \leq \aleph$, is a hypercomputer in which there are $\aleph$ many input, working and output registers which each can store 0 or 1 and which supports programs with $\beth$ states, with $\beth$ instructions (5-tuples), and which supports a maximum of $\daleth$ steps.*

Definition 4. *A non-deterministic serial $\langle \aleph, \beth, \daleth \rangle$-hypercomputer, for cardinals $\aleph$, $\beth$ and ordinal $\daleth$, where $\beth \leq \daleth \leq \aleph$, is a serial $\langle \aleph, \beth, \daleth \rangle$-hypercomputer in which the computer chooses a shortest computation of a program with with $\beth$ states among all $\beth^{|\daleth|}$ possible computation paths of length $\daleth$ on given input (using cardinal exponentiation), where $|\daleth|$ returns the cardinal corresponding to ordinal $\daleth$.*

Definition 5. *A parallel $\langle \aleph, \beth, \daleth \rangle$-hypercomputer, for cardinals $\aleph$, $\beth$ and ordinal $\daleth$, where $\beth \leq \daleth \leq \aleph$, is a hypercomputer that can store data in the registers and process data from the registers in parallel. For the purposes of this paper, such a parallel hypercomputer will comprise $\aleph-$many serial $\langle \aleph, \beth, \daleth \rangle$-hypercomputers running independently in step but with the ability to use common read-only input registers and the capability of writing outputs to a set of registers through a second management program. The general case is where the $\langle \aleph, \beth, \daleth \rangle$-hypercomputers are not independent of one another, but, even in the general case, the dependency can be made explicit by taking the output of a parallel $\langle \aleph, \beth, \daleth \rangle$-hypercomputer as an input to a serial $\langle \aleph, \beth, \daleth \rangle$-hypercomputer or to another parallel $\langle \aleph, \beth, \daleth \rangle$-hypercomputer. To be precise, there are $\aleph$ sets of registers $\{\langle I_{\alpha,\gamma}, W_{\beta,\alpha,\gamma}, O_{\alpha,\gamma}\rangle\}$, where $\gamma < \aleph$ is an index of the set of registers and in fact an index of the overall parallel program, and the working and output registers are disjoint, i.e., $\bigcup_{\beta<\aleph,\alpha<\aleph} W_{\beta,\alpha,\gamma} \cap \bigcup_{\beta<\aleph,\alpha<\aleph} W_{\beta,\alpha,\delta} = \emptyset$ if $\gamma \neq \delta$ and $O_{\alpha,\gamma} \neq O_{\alpha,\delta}$ if $\gamma \neq \delta$. For each $\langle I_{\alpha,\gamma}, W_{\beta<\aleph,\alpha,\gamma}, O_{\alpha,\gamma}\rangle$, there is a program, P_γ, of length $\beth$ which runs disjoint computations based on input registers $I_{\alpha,\gamma}$ for $\leq \daleth$ steps and produces any output in $O_{\alpha,\gamma}$ for $\alpha < \aleph$. Instructions in a parallel hypercomputer have the form $\langle$Index of Serial hypercomputer,Current State, Current Set of Registers, Symbol, Action, NextState$\rangle$, so that a program to copy input registers $I_{\alpha,\gamma}$ to working register $W_{1,\alpha,\gamma}$ (without the sequence to move the program into the halting state) is $\langle \gamma, 1, I, 0, \langle 1, 1\rangle, 1\rangle$, $\langle \gamma, 1, I, 1, \langle 2, 1\rangle, 1\rangle$, $\langle \gamma, 1, I, 0, 4, 1\rangle$, $\langle \gamma, 1, I, 1, 4, 1\rangle$, $\langle \gamma, 1, I, 0, \langle 8, 1\rangle, 1\rangle$, where α is the current register in the input registers and in the set $W_{1,\gamma}$ in γ-th hypercomputer in the parallel set). There may be a separate management program M(Q) that copies the contents of all registers $O_{\alpha,\gamma}$ to the registers in the initial state of a separate parallel $\langle \aleph, \beth, \daleth \rangle$-hypercomputer and then runs a given program Q ($= P_\gamma$) that in the halting state contains the output of Q (if any). For ease of computation, it is assumed that parallel hypercomputers can be chained, the output from one parallel hypercomputer being the input to other parallel hypercomputers, and such a chain of hypercomputers is also a parallel hypercomputer. Allowing chains of parallel programs does not change the set of computable functions, but can be useful in practice.*

3.2. Definitions of Computing

Definition 6. *A Turing machine (see Turing [24]) is a serial $\langle < \aleph_0, < \aleph_0, < \aleph_0\rangle$-hypercomputer as it has $\aleph_0$ many registers but with only finitely many registers addressed in the program, and each program having finitely many states and instructions. Although a finite program may not stop, a function is usually considered computable if there are $< \aleph_0$ steps.*

Definition 7. *A computation can be considered to take place in discrete time interval steps indexed by natural numbers. Following Koepke [4], a "step" can be taken to have three components: the current state at time n, written $S_n(R)$, a pointer to the natural number index of the current register, $H_n(R)$, and the contents of all the registers (a "snapshot" of the computation), $C_n(R) : N_{<\omega} \to \{0, 1\}$, where R is a set of registers I, W_β, O and $N_{<\omega}$ is an unbounded but a finite set of natural*

numbers. If $\langle m, R, b, a, p\rangle$ *is the instruction, where* m, p *are natural number indices of the current and next state, such that* $S_n(R) = m$ and $C_n(R)(H_n) = b$ *then:*

- $S_0(R) = 0$;
- $S_{n+1}(R) = p$;
- $H_0(R) = 0$;
- $H_{n+1}(I) = H_n(I) - 1$ if $a = 3$ and $H_n(I)$ is a successor ordinal;
- $H_{n+1}(W_m) = H_n(W_m) - 1$ if $a = \langle 7, m\rangle$ and $H_n(W_m)$ is a successor ordinal;
- $H_{n+1}(O) = H_n(O) - 1$ if $a = 9$ and $H_n(O)$ is a successor ordinal;
- $H_{n+1}(I) = H_n(I) + 1$ if $a = 4$;
- $H_{n+1}(W_m) = H_n(W_m) + 1$ if $a = \langle 8, m\rangle$;
- $H_{n+1}(O) = H_n(O) + 1$ if $a = 10$;
- $H_{n+1}(I) = 0$ if $a = 11$;
- $H_{n+1}(W_m) = 0$ if $a = \langle 12, m\rangle$;
- $H_{n+1}(O) = 0$ if $a = 13$;
- $H_{n+1}(R) = H_n(R)$ otherwise;
- $C_0(I)(\zeta) = I_\zeta$ for all $\zeta < \aleph$;
- $C_{n+1}(W_m)(\zeta) = 0$ if $a = \langle 1, m\rangle$ and $\zeta = H_n(W_m)$;
- $C_{n+1}(O)(\zeta) = 0$ if $a = 5$ and $\zeta = H_n(O)$;
- $C_{n+1}(W_m)(\zeta) = 1$ if $a = \langle 2, m\rangle$ and $\zeta = H_n(W_m)$;
- $C_{n+1}(O)(\zeta) = 1$ if $a = 6$ and $\zeta = H_n(O)$;
- $C_{n+1}(\zeta) = C_n(\zeta)$ otherwise for all $\zeta < |N_{<\omega}|$.

Definition 8. *A serial computer is a serial hypercomputer in which there are finitely many input, working and output registers which each can store 0 or 1 and which supports programs with finitely states, with finitely instructions (5-tuples), and which supports an unbounded finite number of steps.*

Definition 9. *A non-deterministic serial* computer *is a serial computer in which the computer chooses a shortest computation of a program with* $|N_{<\omega}|$ *states among all* $\beth^{|N_{<\omega}|}$ *possible computation paths of finite length* $\beth$ *on given input.*

Definition 10. *A parallel computer is a parallel hypercomputer that can store data in the registers and process data from the registers in parallel.*

3.3. Hypercomputation and Trees

A tree is a way of visualizing sequences, where the branches of the trees are sequences and sets of branches correspond to subtrees. In the case of a hyper-computation, the sequences are $\beth$-sequences of states that depend on the content of the input registers, see Clarkson and Schneider [25]. It is more accurate to decorate each node in the hypercomputation tree with the snapshot of the computation as a whole, represented by C_α at the $\alpha < \beth$ node in the computation. It is easily seen that, in the case of a serial computer, the hypercomputation tree comprises one branch; in the case of a parallel computer, the hypercomputation tree has $\beth^{|\beth|} = 2^{|\beth|} \leq 2^{\aleph}$ branches if $\beth \geq \aleph_0$) in the case of an indeterministic serial computer the tree comprises a single branch selected by a function acting on a tree of $\beth^{|\beth|} = 2^{|\beth|} \leq 2^{\aleph}$ branches if $\beth \geq \aleph_0$. The same is true with computations, albeit the trees have finite branches.

Let us state some theorems about hypercomputation and complexity that can be proved using trees. For this, we need to know that the *run-time complexity* of an algorithm is the number of steps needed to produce the output given the input as a function of the length of the input in bits. These theorems are not deep in any sense, but they do show the insights that hypercomputational trees provide.

Theorem 1. *The run-time complexity of a program running on a serial* $\langle \aleph, \beth, \daleth \rangle$*-hypercomputer has complexity* $< \left| \beth^{K(\aleph)} \times K(\aleph) \right|$*, where complexity* $K(\aleph)$ *measures the cardinality of the number of steps in the computation and must be* $\leq |\daleth|$*.*

Proof. Given that for complexity measure $K(\aleph)$ there are (a cardinal number of) $\beth^{K(\aleph)}$ execution paths for the number of states $\beth$ of the program (excluding the halting state as the halting state will terminate the $\daleth$-sequence), treating each execution path as the execution path of a program running on a serial $\langle \aleph, \beth, \daleth \rangle$-hypercomputer, we can see that the total run time (for all execution paths) is of the form $\left| \beth^{K(\aleph)} \times K(\aleph) \right|$. Hence, the run time for any one computation path is $< \left| \beth^{K(\aleph)} \times K(\aleph) \right|$, where the total number of steps $K(\aleph) \leq |\daleth|$. □

Corollary 1. *The run-time complexity of a program running on a serial computer– that has polynomial run-time is strictly less complex than exponential run-time.*

Theorem 2. *The run-time complexity of a program running on a parallel* $\langle \aleph, \beth, \daleth \rangle$*-hypercomputer has complexity* $= \left| \beth^{K(\aleph)} \times K(\aleph) \right|$*.*

Proof. A parallel computer will execute all possible state transitions, i.e., $\beth^{K(\aleph)}$ execution paths for the number of states $\beth$ of the program. As the run time is $K(\aleph)$, for a single execution path, the total complexity is $\left| \beth^{K(\aleph)} \times K(\aleph) \right|$. □

Corollary 2. *The run-time complexity of a program running on a parallel computer that has polynomial run time on each execution path has exponential run time.*

Theorem 3. *The run-time complexity of a program running on an indeterministic serial* $\langle \aleph, \beth, \daleth \rangle$*-hypercomputer has complexity* $\leq \left| \beth^{K(\aleph)} \times K(\aleph) \right|$*, where complexity* $K(\aleph)$ *measures the number of steps in the computation and must be* $\leq |\daleth|$*.*

Proof. In the case of an indeterministic serial $\langle \aleph, \beth, \daleth \rangle$-hypercomputer, because a selection of a single computation path is made on the whole hypercomputation tree, in general, the run-time complexity of a program is $\leq \left| \beth^{K(\aleph)} \times K(\aleph) \right|$. □

We now move on to consider diagonal functions of classes of hypercomputing algorithms, which are functions that are constructed to have values outside a class of such algorithms.

Theorem 4. *If a class of algorithms, A, contains functions* $\aleph^{\aleph} \to \aleph^{\aleph}$ *for a cardinal hypercomputed by a serial* $\langle 2^{\aleph}, 2^{\aleph}, 2^{\aleph} \rangle$*-hypercomputer that has run-time complexity* $\aleph^{\kappa}$ *for* κ*, a variable cardinal run-time complexity measure over* $\kappa < \aleph$*, then the diagonal function of the class A is* $\aleph^{sup_z(z>k)}$ *where* $sup_z(z > k)$ *is the least* $z > \kappa$*.*

Proof. Note that $\aleph^{\aleph} = 2^{\aleph}$ in cardinal terms for infinite cardinal $\aleph$ and that a function of cardinal complexity $|x|^{|x|}$ dominates the complexity of each member of the $|x|^{|x|<\aleph}$ complexity class and is the least cardinal to do so, for x a sequence of ordinals (state labels) $< \aleph$ and therefore having length $\aleph$. If there were an ordinal $\alpha < \aleph^{\aleph}$ such that $\{\alpha\}(y) = \{y\}(y) + 1$ for y variable over ordinals $< \aleph^{\aleph}$, $\{\alpha\}$ is a hypercomputable function with ordinal program code α, $\{y\}$ enumerates the class of all programs of the $|x|^{|x|<\aleph}$ complexity class for different values of y, then substituting α for y would result in the contradiction $\{\alpha\}(\alpha) = \{\alpha\}(\alpha) + 1$. Here, as $\{y\}(y) + 1 < \aleph^{\aleph}$, then it is true that $\{y\}(y) + 1 \in \aleph^{\aleph}$ (and *vice versa* using the standard set representation of an ordinal). It follows that $\alpha \geq \aleph^{\aleph}$, and we have seen that a class of complexity $\alpha = \aleph^{\aleph}$ suffices to enumerate all complexity classes

of cardinal complexity $\aleph^\kappa$. This is a diagonal construction originating from Kleene [26], although the use of diagonalization to show that the complexity of the class of computable functions is due to Peter [27]. □

Corollary 3. *Polynomial run-time computing algorithms of order n have diagonal functions of order n + 1 for natural number n, where the order of a polynomial is the greatest exponent value of the variable (which is a consequence of the Time Hierarchy Theorem; see Arora and Barak [28], for example).*

Proof. Follow the proof of Theorem 4 with $\aleph := n + 1$ and $\kappa \leq n$. □

Corollary 4. *Polynomial run-time computing algorithms of unbounded finite order running on a serial or indeterministic serial computer have an exponential run-time diagonal function.*

Proof. Follow the proof of Theorem 4 with $\aleph :=< \aleph_0$, so that $\aleph^\aleph = 2^{<\aleph_0} =< \aleph_0$ and technically $\kappa < |x|$ to obtain the diagonal of functions of run-time complexity $|x|^\kappa$ for natural numbers x, κ as $|x|^{|x|}$. □

Corollary 5. *It is not possible to differentiate a serial or indeterministic serial computer for polynomial run-time programs by means of a diagonal function (see Fortnow [29]).*

Remark 1. *Finally, in this section, it is worth noting that, in general, the computational resources of a hypercomputer do not collapse to those of a computer when used for proof. The case of an $2^{\aleph_0}$-hypercomputer in the next section illustrates this conclusion well. However, it is always possible to represent mathematical objects that require greater computational resources and to (hyper) prove that certain computations always, sometimes or never succeed. There is a priori no upper bound on this representation process (of ordinals or types for example), but the hyper-proof resources of any serial $\langle \aleph, \aleph, \aleph \rangle$-hypercomputer will cover those of a serial $\langle 2^\aleph, 2^\aleph, 2^\aleph \rangle$-hypercomputer almost nowhere in the sense that $2^\aleph - \aleph = 2^\aleph$ in terms of number of objects for infinite $\aleph$.*

3.4. A $2^{\aleph_0}$-Computer

Definition 11. *A particularly nice hypercomputer is simply a hypercomputation tree consisting of binary valued ω-sequences as branches, called a $2^{\aleph_0}$-hypercomputer, because the set of all real numbers is representable. Each branch corresponds to an extensionally unique property of the natural numbers since an ω-sequence is simply a function from $N \to \{0, 1\}$. It is assumed that branches can depend on the computations in other branches, so there are serial computations as well as parallel ones.*

Definition 12. *The arithmetical hierarchy is the hierarchy of propositions of the form $(\exists x \in N)P_n(x)$ for $n \in N$, known as Σ_n^0, and $(\forall x \in N)Q_n(x)$ for $n \in N$, known as Π_n^0, where P_n is a property defined by a formula in Π_{n-1}^0, Q_n is a property defined by a formula in Σ_{n-1}^0, and $P_0$and Q_0 are computationally decidable propositions.*

Definition 13. *The analytical hierarchy is the hierarchy of propositions of the form $(\exists r : N \to 2)P_n(r)$ for $n \in N$, known as Σ_n^1, and $(\forall r : N \to 2)Q_n(r)$ for $n \in N$, known as Π_n^1, where P_n is a property defined by a formula in Π_{n-1}^1, Q_n is a property defined by a formula in Σ_{n-1}^1, and $P_0$and Q_0 are propositions in the arithmetical hierarchy with real number parameters.*

Definition 14. *The lightface analytical hierarchy is the hierarchy of propositions of the form $(\exists r : N \to 2)(\forall m \in N)P_n(\langle r(0), \ldots, r(m-1)\rangle)$ for $n \in N$, known as Σ_n^1, and $(\forall r : N \to 2)(\forall m \in N)Q_n(\langle r(0), \ldots, r(m-1)\rangle)$ for $n \in N$, known as Π_n^1, where P_n is a property defined by a formula in Π_{n-1}^1, Q_n is a property defined by a formula in Σ_{n-1}^1, and P_0 and Q_0 are propositions in the arithmetical hierarchy.*

Definition 15. *Formulas in a formal language that include quantifiers are called first-order if quantifiers apply to terms (including variables) of the theory. Formulas are called second-order if quantifiers apply to all properties of the terms and more generally over all relations and all functions with the terms in their domain.*

Theorem 5. *Any proposition in the arithmetical hierarchy can be decided by a $2^{\aleph_0}$-computer by means of computable operations on countably many branches of a hypercomputation tree.*

Proof. We proceed by mathematical induction. A single branch representing a particular property $P : N \to 2$ is sufficient to decide a proposition of the kind $(\forall x : N)(P(x))$ if $P(n)$ is the constant ω-sequence of 1s or $(\exists x : N)(P(x))$ if $P(n)$ is not the constant sequence of 0s for a P computable and quantifier-free. For the induction step, $\neg P$ can be decided by considering the branch that decides P, $P \vee Q$ and $P \wedge Q$ can be decided by considering the branches that decide P and Q, while, if $Q(n)$ can be decided by a $2^{\aleph_0}$-computer by means of computable operations on countably many branches of a hypercomputation tree for all $n \in N$, then $(\exists x \in N)Q(x)$ and $(\forall x)Q(x)$ can be decided considering the branch comprising truth values of $Q(n)$ in increasing order of $n \in N$, that is, by considering countably many branches. □

Theorem 6. *The truth-value of any proposition P of the form $(\exists f : N \to 2)Q(f)$ for formula Q of first-order arithmetic containing no real number quantifiers such that $\neg P$ has the form $(\exists f : N \to 2)R(f)$ for formula R of first-order arithmetic containing no real number quantifiers can be decided by a $2^{\aleph_0}$-computer.*

Proof. Either P or $\neg P$ is true, so, by enumerating the set of all real numbers (using the Axiom of Choice and transfinite recursion up to $2^{\aleph_0}$) and applying Theorem 5 to compute the truth value of $Q(f)$ and $R(f)$ for particular real numbers f because they are in the arithmetical hierarchy with real number parameters by definition, we see that P or $\neg P$ will be decided as true. □

Theorem 7. *The truth-value of any proposition P in the analytical hierarchy can be decided by a $2^{\aleph_0}$-computer.*

Proof. Proceed by mathematical induction on the complexity of the quantifiers in the formula P, the basis case being proved in Theorem 6. If Q is a formula that starts $(\exists f : N \to 2) \ldots R(f, g)$, for real number parameter g that, by assumption, the truth value of which can be hypercomputed, then the truth value of $(\forall g : N \to 2)(\exists f : N \to 2) \ldots R(f, g)$ can be hypercomputed by enumerating all real numbers g and hypercomputing the truth value of $(\exists f : N \to 2) \ldots R(f, g)$. If $(\exists f : N \to 2) \ldots R(f, g)$ is false for some g, then we can conclude that $(\forall g : N \to 2)(\exists f : N \to 2) \ldots R(f, g)$ is false; otherwise, it is true. It can be seen that the computation depends on $\leq 2^{\aleph_0}$ branches of the hypercomputation tree. If Q is a formula that starts $(\forall f : N \to 2) \ldots R(f, g)$, for real number parameter g that, by assumption, the truth value of which can be computed, then the truth value of $(\exists g : N \to 2)(\forall f : N \to 2) \ldots R(f, g)$ can be hypercomputed by hypercomputing $(\forall f : N \to 2) \ldots R(f, g)$ and, if it is true for some g, then we can conclude that $(\exists g : N \to 2)(\forall f : N \to 2) \ldots R(f, g)$ is true; otherwise, it is false. □

Theorem 8. *The truth-value of any proposition P in the lightface analytic hierarchy can be decided by a $2^{\aleph_0}$-computer.*

Proof. In the lightface analytic hierarchy (see Martin [30], for example), all formulas are computable on all finite initial subsequences of an ω-sequence. In this case, the proof of Theorem 7 goes through in exactly the same way as quantification over the length of sequences, which is the same as quantification over the natural numbers. □

Remark 2. *A $2^{\aleph_0}$-hypercomputer will not be able to decide the truth of propositions that involve arbitrary sets of real numbers or quantification over arbitrary sets of real numbers. For that, we would need to move to a $2^{2^{\aleph_0}}$-hypercomputer. In terms of the traditional classification of formal number theory, we can say that an $2^{\aleph_0}$-hypercomputer is sufficient to decide all the propositions of formal first and second-order (natural) number theory, but not third order number theory.*

3.5. A Hierarchy of Hypercomputers by Strength

The following section cites a couple of known results from other authors and shows that the hypercomputer described in Section 2 will hypercompute the truth of all propositions in the standard model of set theory, the von Neumann hierarchy of pure sets, V.

Definition 16. *The number of bits in a sequence, set, proposition or function is the least number of bits to which the sequence, set enumerated as a sequence, characteristic function of the proposition or function expressed as a sequence can be losslessly compressed. Lossless compression of a sequence of bits of length $\aleph$, an $\aleph$-sequence for short, here is simply a binary sequence of codes each representing a repeated $< \aleph$-sequence of bits and a code for the number of repetitions, in general there being a shortest sequence which can be obtained by repeated lossless compressions. A code is a finite binary sequence that can be used to represent other finite binary sequences in such a way that the representation is unique and the original finite binary sequence can be recovered from its code. A sequence, set, proposition or function is said to require $\leq \aleph$ bits of information to define if the number of bits in the sequence, set, proposition or function is $\leq \aleph$.*

Theorem 9. *An infinite run time Turing machine that is a computer with finitely many register values and a finite program, but allowing for infinite run times, can decide propositions which extend up the lightface analytical hierarchy to those defined by a Π^1_1 formula, i.e., $(\forall f : N \to 2)(\forall n \in N)P(\bar{f}(n))$ for real number variable f and P a computable predicate on finite initial subsequence of f of length n, $\bar{f}(n) := \langle f(0), ..., f(n-1)\rangle$, and every infinite run time decision problem can be defined by a Δ^1_2 formula, i.e., a formula of the form $(\forall f)(\exists g)(\forall n \in N)P(\bar{f}(n), \bar{g}(n))$ and $(\exists f)(\forall g)(\forall n \in N)Q(\bar{f}(n), \bar{g}(n))$, see Hamkins and Lewis [1].*

Theorem 10. *Using an ordinal Turing machine, with a finite program and a set of registers indexed by all bounded sets of ordinals, the class of all ordinal-computable sets of ordinals that can be computed from finitely many ordinal parameters is Gödel's constructible set universe L (see Koepke [4,11]).*

Theorem 11. *A serial $\langle 2^{\aleph}, 2^{\aleph}, 2^{\aleph}\rangle$-hypercomputer can hypercompute (a) the truth of first-order computably decidable propositions with quantification over sets that require $\leq \aleph$ bits of information to define, (b) the truth of first-order computably decidable propositions like (a) but with the addition of allowing set membership of sets that require $\leq 2^{\aleph}$ bits of information to define, and (c) a serial $\langle 2^{2^{\aleph}}, 2^{2^{\aleph}}, 2^{2^{\aleph}}\rangle$-hypercomputer can compute the truth of second-order propositions about sets that require $\leq \aleph$ bits of information to define.*

Proof. (a) To start, the truth of computable relations involving finitely many sets that require $\leq \aleph$ bits of information to define (including the standard logical operators $\wedge$, $\vee$,$\rightarrow$, $\leftrightarrow$ and $\neg$) can be decided by a program with finitely many instructions in $\leq \aleph$ steps because the computable relation generates a finite program and $\leq \aleph$ steps are needed, one for each bit. Then, to decide $(\forall x)R(x)$ for x a set that requires $\leq \aleph$ bits of information to define and R recursive, loop through the set of all sets that require $\leq \aleph$ bits of information to define by enumerating the set (applying transfinite recursion to the selection of elements by the Axiom of Choice to define a well-ordering), run the program for *R(x)* in disjoint register sets in series, and then copy the results (*0* or *1*, i.e., false or true) to another disjoint set of registers, the program having $\leq 2^{\aleph}$ instructions to avoid the use of the Axiom of Choice as an oracle and the computation having $\leq 2^{\aleph}$ steps. Any set that requires $\leq \aleph$ bits of

information to define can be either be a member or not a member of the set of such sets; hence, the cardinality of the set of all sets that require $\leq \aleph$ bits of information to define, X say, is the same as the set of all functions $\aleph \to 2$, i.e., $2^{\aleph}$. Hence, the total number of steps to loop through every member of X is $\aleph \times 2^{\aleph} = 2^{\aleph}$. To "loop through" the quantification domain, coding can be used to detect in finitely many instructions which registers have been accessed by the program, and the least unaccessed member of the set can be accessed next, which acts as a label for the start of the loop and which is the next state for instructions in the loop after the program for $R(x)$ has run.). Looping requires one new state. If a sequence $\langle a_1, a_2, \cdots, a_{i<\aleph}, \cdots \rangle$ of length $2^{\aleph}$, where a_i is a member of the quantification domain and a binary sequence of length $< \aleph + 1$, is coded as $\langle a_1, 1, a_2, 1, \cdots, 1, a_{i<2^{\aleph}}, 1, \cdots \rangle$, by placing a 1 marker after every successor and limit member of the sequence, then the *1* can be replaced with *0* if the previous register has been accessed by the program. The program can proceed until it finds a register succeeded by a *1*. To create and load all sets that require $\leq \aleph$ bits of information to define requires a program of length $\leq 2^{\aleph}$ because there are $\leq 2^{\aleph}$ such sets to be computed, each requiring $\leq \aleph$ instructions. The conjunction ("and") of the truth values of *R(x)* is then computed by a finite program (see Definition 1 for the outline of a finite program to compute the truth value of a conjunction), and $(\forall x)R(x)$ is true if and only if the conjunction has value *1* (true). $(\exists x)R(x)$ can be decided similarly using disjunctions ("or") rather than conjunctions. By induction on quantifier complexity, the truth of any first-order proposition about sets that require $\leq \aleph$ bits of information to define (with a recursive quantifier free formula) can be decided by a $\langle 2^{\aleph}, < \aleph_0, 2^{\aleph} \rangle$-hypercomputer given an enumeration of a set of sets that require $\leq \aleph$ bits of information to define. If the loading of the input to be looped through is included or the input needs to be enumerated, a serial $\langle 2^{\aleph}, 2^{\aleph}, 2^{\aleph} \rangle$-hypercomputer suffices to compute the truth of any first-order quantified proposition about sets that require $\leq \aleph$ bits of information to define.

(b) To show that a first-order quantified proposition with quantification over sets that require $\leq \aleph$ bits of information to define and with the addition of specific sets that require $\leq 2^{\aleph}$ bits of information to define can also be computed by a serial $\langle 2^{\aleph}, 2^{\aleph}, 2^{\aleph} \rangle$-hypercomputer, we note that a serial $\langle 2^{\aleph}, 2^{\aleph}, 2^{\aleph} \rangle$-hypercomputer can compute any set that requires $\leq 2^{\aleph}$ bits of information to define by starting with a blank tape (i.e., all *0*s) and running a program of length $2^{\aleph}$ to write a value (*0* or *1*) to each register. Membership of a set, $x \in X$, where each x must take $\leq \aleph$ bits to define to be consistent with (a) (since $x \in 2^{\aleph}$ as x takes $\leq \aleph$ bits to define), can therefore be computed by a serial $\langle 2^{\aleph}, 2^{\aleph}, 2^{\aleph} \rangle$-hypercomputer by looping through the set X with current value $y \in X$ and checking whether $y = x$. The inductive argument in (a) above can then be applied to show that a serial $\langle 2^{\aleph}, 2^{\aleph}, 2^{\aleph} \rangle$-hypercomputer can compute the truth of any first-order proposition with quantification over sets that require $\leq \aleph$ bits of information to define and which have set membership of sets that require $\leq 2^{\aleph}$ bits of information to define.

(c) The truth of a second-order proposition of set theory with quantification over sets that require $\leq 2^{\aleph}$ bits of information and sets of sets that require $\leq \aleph$ bits of information can be decided by "looping through" every set of sets that require $\leq \aleph$ bits of information, which requires $2^{2^{\aleph}}$ registers and $2^{2^{\aleph}}$ steps with a finite program and which depends on $2^{2^{\aleph}}$ instructions to create and loop through the data, i.e., the set of sets that require $\leq \aleph$ bits of information. □

Theorem 12. *A parallel $\langle 2^{\aleph}, \aleph, \aleph \rangle$-hypercomputer can hypercompute (a) the truth of first-order computably decidable propositions with quantification over sets that require $\leq \aleph$ bits of information to define, (b) the truth of first-order computably decidable propositions like (a) but with the addition of allowing set membership of sets that require $\leq 2^{\aleph}$ bits of information to define, and (c) a parallel $\langle 2^{2^{\aleph}}, 2^{\aleph}, 2^{\aleph} \rangle$-hypercomputer can compute the truth of second-order propositions about sets that require $\leq \aleph$ bits of information to define.*

Proof. (a) Note that a parallel $\langle 2^{\aleph}, \aleph, \aleph \rangle$-hypercomputer can write $2^{\aleph}$ sets that require $\leq \aleph$ bits of information to define into the registers in parallel. Proceed by induction with the

hypothesis that a parallel $\langle 2^{\aleph}, \aleph, \aleph \rangle$-hypercomputer can compute the truth of first-order quantified propositions of sets that require $\leq \aleph$ bits of information to define, noting that, for the basis case of a computationally decidable relationship between finitely many sets that require $\leq \aleph$ bits of information to define, it takes $\leq \aleph$ instructions and $\leq \aleph$ steps to write finitely many sets that require $\leq \aleph$ bits of information to define to a set of registers and then finitely many instructions and $\leq \aleph$ steps to compute the recursive relationship for those sets. For the induction step, note that, for $(\forall x)R(x)$ or $(\exists x)R(x)$, $2^{\aleph}$ sets that require $\leq \aleph$ bits of information to define can be loaded by a parallel $\langle 2^{\aleph}, \aleph, \aleph \rangle$-hypercomputer across $2^{\aleph}$ disjoint sets of $2^{\aleph}$ registers and the quantification can be parallelized by running a (finite) program for deciding *R(x)* in parallel in $\aleph$ steps, for $(\forall x)R(x)$ writing *1* to an output register of the management program initially and then writing *0* to the output register if any of the *R(x)* computes as false, while, for $(\exists x)R(x)$, writing *0* to an output register initially and then writing *1* to the output register if any of the *R(x)* computes as true.

(b) If we add propositions involving membership of $\leq 2^{\aleph}$ specific sets, assumed for consistency (a) to consist of members which have $\leq \aleph$ bits to define, then to write a specific set requires a parallel $\langle 2^{\aleph}, \aleph, \aleph \rangle$-hypercomputer if each disjoint set of $2^{\aleph}$ registers contains one set that requires $\leq \aleph$ bits of information to define. It is assumed that the $\leq \aleph$ bits are presented serially and cannot be parallelized, for example by a recursive relationship. Testing membership of a specific set of sets that require $\leq \aleph$ bits of information to define, r, requires matching r against $2^{\aleph}$ disjoint sets of registers which contain one set that requires $\leq \aleph$ bits of information to define, $s_{\alpha<2^{\aleph}}$, which can be done in parallel with a finite program in $\aleph$ steps as follows. Use r and s_{α} from the input registers and create a set of working registers, $W_{\alpha<2^{\aleph}}$, with one register each in one step and with a finite program writing 1 to each W_{α} in parallel. For r and each s_{α}, for ordinal $\beta < \aleph$, perform the operation $(r)_{\beta} \leftrightarrow (s_{\alpha})_{\beta}$, that is, $((r)_{\beta} \wedge (s_{\alpha})_{\beta}) \vee ((\neg r)_{\beta} \wedge (\neg s_{\alpha})_{\beta})$, in parallel, which returns *1* if $(r)_{\beta} = (s_{\alpha})_{\beta}$ and 0, otherwise; and, if the result is *0*, write *0* to W_{α} and then halt the program; otherwise, write *1* to W_{α} and then move right one register along r and s_{α} to $(r)_{\beta+1}$ and $(s_{\alpha})_{\beta+1}$. At limit ordinals λ, proceed as normal by performing the operation $(r)_{\lambda} \leftrightarrow (s_{\alpha})_{\lambda}$. To implement the pseudo-code as a program, it is possible to use a hypercomputer with three ordinary states, 2,3,4, an initial state, *1*, a halting state, 5, with the following instructions, assuming that the program starts in state *1* that two sets that require $\leq \aleph$ bits of information to define are for simplicity stored in $W_{1,\alpha<\aleph,\gamma}$ and $W_{2,\alpha<\aleph,\gamma}$, the result of bit-wise comparison of the sets that require $\leq \aleph$ bits of information to define is stored in $W_{3,1}$. A suitable program is $\langle \gamma, 1, W_1, 0, \langle 2, 3 \rangle, 4 \rangle$, $\langle \gamma, 1, W_1, 1, \langle 2, 3 \rangle, 4 \rangle$, $\langle \gamma, 4, W_1, 0, \langle 8, 1 \rangle, 3 \rangle$, $\langle \gamma, 4, W_1, 1, \langle 8, 1 \rangle, 2 \rangle$, $\langle \gamma, 3, W_2, 1, \langle 1, 3 \rangle, 5 \rangle$, $\langle \gamma, 2, W_2, 0, \langle 1, 3 \rangle, 5 \rangle$, $\langle \gamma, 3, W_2, 0, \langle 8, 1 \rangle, 4 \rangle$, $\langle \gamma, 2, W_2, 1, \langle 8, 1 \rangle, 4 \rangle$, $\langle \gamma, 1, W_0, 0, \langle 12, 0 \rangle, 1 \rangle$, $\langle \gamma, 1, W_0, 1, \langle 12, 0 \rangle, 1 \rangle$, $\langle \gamma, 1, W_0, 1, 0, 5 \rangle$. The reason that the state in the main loop is the highest ordinary state of 4 is to allow the program to start in the main loop at limit ordinals. It can be seen that the program will either halt in state 5 with output 0 or in state 4 with output *1* when the computation runs to completion (i.e., at step $o(\aleph)$).

(c) Each of a maximum of $2^{2^{\aleph}}$ sets of sets that require $\leq 2^{\aleph}$ bits of information to define can be represented as specific sets when computing the truth of first-order quantified propositions involving such sets. Put more formally, since a parallel $\langle 2^{\aleph}, \aleph, \aleph \rangle$-hypercomputer can compute the truth of a first-order quantified proposition with quantification over sets of sets that require $\leq \aleph$ bits of information to define with the addition of membership of specific sets that require $\leq 2^{\aleph}$ bits of information to define, if *R(X)*, for X, a set of sets that require $\leq \aleph$ bits of information to define, is a formula of set theory with free variable X, then $(\forall X)R(X)$ can be computed in parallel across $2^{2^{\aleph}}$ disjoint sets of $2^{\aleph}$ registers by writing *1* to an output register of the management program initially and then writing *0* if any of *R(X)* is false; and for $(\exists X)R(X)$ by writing *0* to an output register initially and then writing *1* if any of *R(X)* is true. By induction on quantifier complexity of a second-order predicate *A(X)*, since the parallel computation adds two steps and needs a finite program to implement *A(X)* on each parallel hypercomputer, it can be seen that a parallel $\langle 2^{2^{\aleph}}, 2^{\aleph}, 2^{\aleph} \rangle$-

hypercomputer can hypercompute the truth of second-order quantified propositions about sets that require $\leq 2^{\aleph}$ bits of information to define. □

Remark 3. *It is possible to take the union of all serial $\langle 2^{\aleph}, 2^{\aleph}, 2^{\aleph} \rangle$-hypercomputers, for example, to define a universal hypercomputer, which does hypercompute the truth of properties in the von Neumann hierarchy of pure sets, V. It is possible to continue by specifying hypercomputers that decide the truth of propositions that involve quantification over proper classes (such as those in Morse–Kelley set theory, see Monk [31]).*

4. Discussion

The main purpose of this paper is to show that computation and hypercomputation work are linked, and have similarities in terms of complexity classes and differences in terms of computational power. One of the key differences is that the set of natural numbers is closed under cardinal exponentiation, while infinite cardinals strictly increase under cardinal exponentiation (until the first strongly inaccessible cardinal if such exists). Thus, there is a proper class of hypercomputers once the proper class of all ordinals is allowed as resources (which is true for Koepke's ordinal register machines and ordinal Turing machines of Koepke [4], Koepke and Koerwien [5], Koepke and Siders [6], Koepke [11] as well as for the $\langle 2^{\aleph}, 2^{\aleph}, 2^{\aleph} \rangle$-hypercomputers in Section 3 above).

Given that computation defines a mechanical, finitary process, in light of Gödel's Incompleteness Theorem (see Gödel [32], Smorynski [33]) indicating the existence of true propositions (of the form $(\forall x \in N)P(x)$ for P quantifier-free and having only natural number constants) not provable in sufficiently rich formal number theory (such as weak fragments of first-order Peano Arithmetic), hypercomputation shows that there are formal theories that have a naturally infinitary semantics in the sense that truth of a proposition in the language of the theory can be decided hypercomputationally but not computationally. The author believes that principles of mathematics (including axioms of set theory, type theory and class theory) that are not computably decidable should be judged on whether they are true or at least plausible from the perspective of an infinitary semantics. It is, of course, very difficult to judge whether a principle of mathematics is true from an infinitary point of view, but there are principles that are true even if humans cannot effectively use them. An example of a true principle from an infinitary perspective is the Axiom of Choice, see Jech [34], which was used extensively in this paper, ether as an oracle or in writing infinitely many instructions. In the context of class theory, the Axiom of Global Choice is true because a choice function on the von Neumann hierarchy of non-empty sets can be extended to a choice function on the universe of sets V minus the empty set (see Fraenkel et al. [35]).

Funding: This research received no external funding.

Data Availability Statement: Not applicable.

Acknowledgments: Many thanks to Tim Button and Filip Hrdlicka for correspondence on related ideas.

Conflicts of Interest: The author declares no conflict of interest.

References

1. Hamkins, J.; Lewis, A. Infinite Time Turing machines. *J. Symb. Log.* **2000**, *65*, 567–604. [CrossRef]
2. Koepke, P. *Infinite Time Register Machines*; CiE 2006; Beckmann, A., Ed., University of Athens: Athens, Greece, 2006; Volume LNCS 3988, pp. 257–266.
3. Koepke, P.; Miller, R.G. An enhanced theory of infinite time register machines. In *Logic and the Theory of Algorithms*; Lecture Notes in Computer Science; Beckmann, A., Ed.; Springer: Berlin, Germany, 2008; Volume 5028,
4. Koepke, P. Turing Computations on Ordinals. *Bull. Symb. Log.* **2005**, *11*, 377–397. [CrossRef]
5. Koepke, P.; Koerwien, M. Ordinal computations. *Math. Struct. Comput. Sci.* **2006**, *16*, 1–18. [CrossRef]
6. Koepke, P.; Siders, R. Register Computations on Ordinals. *Arch. Math. Log.* **2008**, *47*, 529–548. [CrossRef]

7. Carl, M.; Fischbach, T.; Koepke, P.; Miller, R.; Nasif, M.; Weckbecker, G. The basic theory of infinite time register machines. *Arch. Math. Log.* **2010**, *49*, 249–273. [CrossRef]
8. Carl, M.; Ouazzani, S.; Welch, P.D. Taming Koepke's Zoo. In Proceedings of the Sailing Routes in the World of Computation—14th Conference on Computability in Europe, Kiel, Germany, 30 July–3 August 2018; Lecture Notes in Computer Science; Manea, F., Miller, R.G., Nowotka, D., Eds.; Springer: Berlin, Germany, 2018; Volume 10936, pp. 136–145.
9. Carl, M. *Ordinal Computability: An Introduction to Infinitary Machines*; De Gruyter Series in Logic and Its Applications; De Gruyter: Berlin, Germany, 2019.
10. Carl, M. Taming Koepke's Zoo II: Register machines. *Ann. Pure Appl. Log.* **2022**, *173*, 103041. [CrossRef]
11. Koepke, P. Ordinal Computability. In *Mathematical Theory and Computational Practice*; Number 5635 in Lecture Notes in Science; Ambos-Spies, K., Ed.; Springer: Berlin, Germany, 2009; pp. 280–289.
12. Hamkins, J.; Miller, R. Post's Problem for ordinal register machines: An explicit approach. *Ann. Pure Appl. Log.* **2009**, *60*, 302–309. [CrossRef]
13. Welch, P.D. Characteristics of discrete transfinite time Turing machine models: Halting times, stabilization times, and Normal Form theorems. *Theor. Comput. Sci.* **2009**, *210*, 426–442. [CrossRef]
14. Welch, P.D. Discrete transfinite computation models. In *Computability in Context: Computation and Logic in the Real World*; Cooper, S.B., Sorbi, A., Eds.; World Scientific: Singapore, 2011; pp. 375–414.
15. Blanchetti, M. Weaker variants of infinite time Turing machines. *Arch. Math. Log.* **2020**, *59*, 335–365. [CrossRef]
16. Rin, B. The computational strengths of alpha-tape infinite time Turing machines. *Ann. Pure Appl. Log.* **2014**, *1165*, 1501–1511. [CrossRef]
17. Welch, P.D. Characterisations of variant transfinite computational models: Infinite time Turing, ordinal time Turing, and Blum–Shub–Smale machines. *Computability* **2021**, *10*, 159–180. [CrossRef]
18. Koepke, P.; Seyfferth, B. Ordinal machines and admissible recursion theory. *Ann. Pure Appl. Log.* **2009**, *160*, 310–318. [CrossRef]
19. Schindler, R. P [not =] NP infinite time Turing machines. *Monatshefte für Math.* **2003**, *139*, 335–340. [CrossRef]
20. Hamkins, J.D.; Welch, P.D. Pf not =NPf for almost all f. *Math. Log. Q.* **2003**, *49*, 536–540. [CrossRef]
21. Deolalikar, V.; Hamkins, J.D.; Schindler, R. P [not =] NP [intersect] co-NP for infinite time turing machines. *Log. Comput.* **2005**, *15*, 577–592. [CrossRef]
22. Carl, M. Space and time complexity for infinite time Turing machines. *Log. Comput.* **2020**, *30*, 1239–1258. [CrossRef]
23. Hellman, G. *Mathematics without Numbers. Towards a Modal-Structural Interpretation*; Clarendon Press: Oxford, UK 1989.
24. Turing, A. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proc. Lond. Math. Soc.* **1937**, *s2-42*, 230–265. [CrossRef]
25. Clarkson, M.; Schneider, F. Hyperproperties. In Proceedings of the 21st IEEE Computer Security Foundations Symposium, Pittsburgh, PA, USA, 23–25 June 2008; IEEE Computer Society: Washington, DC, USA, 2008; pp. 51–65.
26. Kleene, S. Recursive predicates and quantifiers. *Trans. Am. Math. Soc.* **1943**, *53*, 41–73. [CrossRef]
27. Peter, R. *Recursive Functions*; Academic Press: New York, NY, USA, 1967.
28. Arora, S.; Barak, B. *Computational Complexity: A Modern Approach*; Cambridge University Press: Cambridge, UK, 2009.
29. Fortnow, L. Diagonalization. In *Current Trends in Theoretical Computer Science*; Chapter Diagonalization; Păun, B., Rozenberg, G., Salomaa, A., Eds.; World Scientific: Singapore, 2001; Volume 102–114.
30. Martin, D.A. Descriptive set theory: Projective sets. In *Handbook of Mathematical Logic*; Studies in Logic and the Foundations of Mathematics; Barwise, J., Ed.; North-Holland: Amsterdam, The Netherlands, 1977; Volume 90, Chapter C.8, pp. 783–818.
31. Monk, J.D. *Introduction to Set Theory*; McGraw Hill: New York, NY, USA, 1969.
32. Gödel, K. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme, I. *Monatshefte für Math. Und Phys.* **1931**, *38*, 173–198. [CrossRef]
33. Smorynski, C. The Incompleteness Theorems. In *Handbook of Mathematical Logic*; Studies in Logic and the Foundations of Mathematics; Barwise, J., Ed.; North-Holland: Amsterdam, The Netherlands, 1977; Volume 90, Chapter D.1, pp. 821–865.
34. Jech, T. About the axiom of choice. In *Handbook of Mathematical Logic*; Studies in Logic and the Foundations of Mathematics; Barwise, J., Ed.; North-Holland: Amsterdam, the Netherlands, 1977; Voume 90, Chapter B.2, pp. 345–370.
35. Fraenkel, A.; Bar-Hillel, Y.; Levy, A. *Foundations of Set Theory*; Studies in Logic and the Foundations of Mathematics; North-Holland: Amsterdam, Netherlands, 1973; Volume 67.