Symmetry for transfinite computability

Lorenzo Galeotti[1], Ethan S. Lewis[2], Benedikt Löwe[3,4,5]

[1] Amsterdam University College, Postbus 94160, 1090 GD Amsterdam, The Netherlands; l.galeotti@uva.nl
[2] 1179 W 1200 S, Springville, UT 84663, United States of America; etonlels@gmail.com
[3] Institute for Logic, Language and Computation, Universiteit van Amsterdam, Postbus 94242, 1090 GE Amsterdam, The Netherlands; b.loewe@uva.nl
[4] Fachbereich Mathematik, Universität Hamburg, Bundesstrasse 55, 20146 Hamburg, Germany
[5] Churchill College, Lucy Cavendish College, St. Edmund's College, & Department of Pure Mathematics and Mathematical Statistics, University of Cambridge, Storey's Way, Cambridge, CB3 0DS, England

Abstract. Finite Turing computation has a fundamental symmetry between inputs, outputs, programs, time, and storage space. Standard models of transfinite computational break this symmetry; we consider ways to recover it and study the resulting model of computation. This model exhibits the same symmetry as finite Turing computation in universes constructible from a set of ordinals, but that statement is independent of von Neumann-Gödel-Bernays class theory.

1 Introduction

A fundamental feature of the theory of computation is that the constituents of computability, viz. in-/output, programs, time, and storage space can be considered to be the same type of object: natural numbers (if necessary, via coding). A Turing machine receives a finite string of symbols as input, has a finite string of symbols as its program, and produces a finite string of symbols as output. Moreover, both its tape and its time flow are indexed by natural numbers. Therefore, since finite strings of symbols can be coded as a natural number, all these objects are of the same type.

We shall refer to this feature as *symmetry*. Various aspects of symmetry permeate the general theory of computation: the symmetry between inputs and programs is the reason for the *software principle* (the existence of universal machines) and the *s-m-n* Theorem; the symmetry between programs and time underlies the zigzag method that allows us to parallelise infinitely many computations into one by identifying the cartesian product of the program space and time with $\mathbb{N} \times \mathbb{N}$ and using Cantor's zigzag function.

The oldest model of transfinite computation are the *Hamkins-Kidder machines* or *Infinite Time Turing Machines* (ITTM), defined in [9]. These machines have a *storage space* of order type ω, but allow computation to be of arbitrary transfinite ordinal length, thereby breaking the symmetry between time and space. This asymmetry makes their complexity theory vastly different from ordinary complexity theory, as discussed in [20, 11, 5, 17, 23, 24].

In [13, 14], Koepke symmetrised Hamkins-Kidder machines and defined what is now known as *Koepke machines* or *Ordinal Turing Machines*: Koepke machines have a class-sized tape indexed by ordinals and run through ordinal time, thereby re-establishing the symmetry between *time* and *storage space*.[1] However, Koepke machines do not have the full symmetry that we find in finite Turing computation: while *time* and *storage space* are represented by arbitrary ordinals, programs are still finite objects.

In this paper, we shall provide a general framework for models of computation and computability that allows us to phrase the quest for symmetry in abstract terms; this is done in § 3. In this framework, we shall define the relevant models of computability, i.e., ordinary Turing computability, Hamkins-Kidder computability, Koepke computability, and our new notion called *symmetric computability* in § 4. We study basic properties of symmetric computability in § 5, and finally show that the full symmetry of symmetric computability cannot be proved in von Neumann-Gödel-Bernays class theory (NBGC) in § 6: symmetry holds if and only if the universe is constructible from a set of ordinals.

This paper contains results from the second author's Master's thesis [16] written under the supervision of the first and the third author. These results are cited in [2, 8, 3] and [1, Exercise 3.9.7].

2 Class theories

In this paper, we work in *von Neumann-Gödel-Bernays class theory*.[2] The language is the usual language of set theory $\mathcal{L}_\in$ with a single binary relation symbol $\in$. We define a unary predicate $\mathrm{set}(x) := \exists y(x \in y)$. Using this predicate, we can define the two *set quantifiers* $\exists^{\mathrm{set}} x\varphi := \exists x(\mathrm{set}(x) \wedge \varphi)$ and $\forall^{\mathrm{set}} x\varphi := \forall x(\mathrm{set}(x) \to \varphi)$. A formula is called *set theoretic* if all of its quantifiers are set quantifiers. In this context we denote by AC the *axiom of choice* for sets, i.e., the statement that "Every set x has a choice function" and contrast it with the axiom of *Global Choice* which is the statement "There is a global choice class function". We write NBG for von Neumann-Gödel-Bernays class theory without the axiom of Global Choice [12, p. 70: Axioms A–D] and NBGC for the theory obtained from NBG adding the axiom of Global Choice [12, p. 70: Axioms A–E]. It is a well-known result due to Easton that if NBG is consistent, then NBG + AC does not prove the axiom of Global Choice (cf., e.g., [6, Theorem 3.1]).

We can transform a formula φ in the language $\mathcal{L}_\in$ into a set theoretic formula φ^{set} by recursively replacing all quantifiers with the corresponding set quantifiers.

[1] Carl argues in [1, Chapter 9] that Koepke machines are the natural infinitary analogue for finitary computation and complexity theory and this was explored in detail in [4].

[2] For more details, cf., e.g., [18, Chapter 4].

This allows us to formulate the famous *conservativity theorem for* NBGC (cf., e.g., [6, Corollary 4.1 & Theorem 4.2]):[3]

Theorem 1. If φ is a sentence in $\mathcal{L}_\in$, then $\mathsf{ZFC} \vdash \varphi$ if and only if $\mathsf{NBGC} \vdash \varphi^{\text{set}}$ if and only if $\mathsf{NBG} + \mathsf{AC} \vdash \varphi^{\text{set}}$.

We define the *axiom of constructibility from a set of ordinals* as the statement "there is a set of ordinals x such that V=L$[x]$". This is a set theoretic sentence and implies the axiom of Global Choice; thus Theorem 1 implies the following result.

Theorem 2. If NBGC is consistent, then NBGC does not prove nor disprove the axiom of constructibility from a set of ordinals.

3 The general framework of Turing computation and computability

We shall frame our discussion of symmetry in a general context that makes the relavent models of computability special cases of a general framework. Our general framework will work on the class Ord of all ordinals and refer to the class V of all sets as potential programs for these machines. All models of computation in this paper will be variants of Turing machines: they have a single class-length tape indexed by ordinals,[4] a read/write head that moves on the tape according to a program. We fix a finite alphabet Σ with at least two elements $\mathbf{0}$ and $\mathbf{1}$ for the remainder of the paper.

Turing hardware & computations. At the highest level of abstraction, we deal with the *Turing hardware*: the *tape* and the *head*, including the description of how they work. We assume that the tape is always indexed by ordinals, split up into discrete *cells* in which a symbol from Σ can be written; also, we assume that time is considered as discrete points in time, indexed by ordinals, and that at each point in time, the head is located at one of the cells; finally, we assume that we have discrete *states*, indexed by ordinals.

For classes X and Y, we write $f{:}\,X \dashrightarrow Y$ for "f is a class function with $\text{dom}(f) \subseteq X$ and $\text{ran}(f) \subseteq Y$" and $f(x)\downarrow$ if and only if $x \in \text{dom}(f)$ and $f(x)\uparrow$ otherwise. We represent the tape content by arbitrary partial class functions from Ord to Σ; we write $\Sigma^{(\text{Ord})}$ for the class of these objects. We shall consider a number of relevant subclasses of this class: $\Sigma^{<\text{Ord}} := \{x \in \Sigma^{(\text{Ord})}\,; \text{dom}(x) \in \text{Ord}\}$, $\Sigma^{\text{FS}} := \{x \in \Sigma^{(\text{Ord})}\,; \text{dom}(x) \text{ is finite}\}$, $\Sigma^{\omega} := \{x \in \Sigma^{(\text{Ord})}\,; \text{dom}(x) = \omega\}$, $\Sigma^{*} = \Sigma^{<\omega} := \{x \in \Sigma^{(\text{Ord})}\,; \text{dom}(x) \in \omega\}$, and $\Sigma^{\text{O}} := \{x \in \Sigma^{(\text{Ord})}\,; |\text{dom}(x)| = 1 \text{ and } \text{ran}(x) = \{\mathbf{0}\}\}$.

[3] We refer the reader to [6, p. 242] and [7, p. 381] for more information on the history of this theorem.

[4] For most models of computability, the number of tapes does not matter; however, in the case of Hamkins-Kidder machines, 1-tape machines and 3-tape machines differ (cf. [10]). Since we do not discuss Hamkins-Kidder machines in detail, this is immaterial for our context.

The classes Σ^{O} and Σ^{FS} are our representations of the classes Ord and $\mathrm{Ord}^{<\omega}$, respectively. The classes Ord and Σ^{O} have a canonical bijection; the classes Σ^{FS} and $\mathrm{Ord}^{<\omega}$ can be identified via the Gödel pairing function.[5] Furthermore, the Gödel pairing function yields a definable bijection between Ord and $\mathrm{Ord}^{<\omega}$ and a bijection $(w,v) \mapsto w * v \colon \Sigma^{<\mathrm{Ord}} \times \Sigma^{<\mathrm{Ord}} \to \Sigma^{<\mathrm{Ord}}$.

A *snapshot* of the machine consists of the tape content, a state, and a position of the head, i.e., a tuple from $\mathrm{Snap} := \Sigma^{(\mathrm{Ord})} \times \mathrm{Ord} \times \mathrm{Ord}$.

The behaviour of the head is governed by the *transition rule*, a class function that describes what the head will do given its past behaviour and a program p. For now, we still allow all sets to be programs (we consider this specification to be part of the *software*), so a transition function is a class function $T\colon \mathrm{Snap}^{<\mathrm{Ord}} \times \mathrm{V} \to \mathrm{Snap}$. Once a transition function T is fixed, given a program p and a snapshot $s = (x, \alpha, \beta)$, we define an ordinal-length sequence of snapshots by recursion: $C_{p,s}(0) := s$, and $C_{p,s}(\gamma) := T(C_{p,s}\restriction\gamma, p)$ for $\gamma > 0$. We shall call this *the computation of program p with initial snapshot s*.

In this paper, we shall only consider two different transition functions, the *finite transition function* T_{f} which is used by ordinary Turing machines and Hamkins-Kidder machines and the *transfinite transition function* T_{t} which is used by Koepke machines (for definitions, cf. § 4).

Turing software. A *model of computation* consists of hardware (i.e., a transition function T) and a class of programs P that can be used for computing. Specifying the class of programs identifies which of the computations are computations according to a program in P.

In this paper, we shall only consider two classes of programs, the class of *finite programs* P_{f} and the class of *transfinite programs* P_{t} (for definitions, cf. § 4).

Computability. A model of computation determines a class of computations, but does not yet tell us what they do. To illustrate this, consider the ordinary notion of Turing computation: for each program and snapshot, we get an infinite sequence of snapshots, but there are many ways to interpret these infinite sequences. Following Turing's original seminal definition [21, § 2], we designate *start* and *halt states*, give a definition of *halting computations* and then interpret the computation as producing a partial function (for definitions, cf. § 4).

Abstractly, we say that an *interpretation* consists of a partial class function I that assigns to a transition function T and each program $p \in P$ a partial function $I(T,p)\colon \Sigma^{(\mathrm{Ord})} \dashrightarrow \Sigma^{(\mathrm{Ord})}$ and a class $D \subseteq \Sigma^{(\mathrm{Ord})}$ called the *domain* of the interpretation. A *model of computability* is a model of computation (i.e., a transition function T and a class of programs P) together with an interpretation. We say that $f\colon D \dashrightarrow D$ is *computable* according to this model of computability if there is a $p \in P$ such that $f = I(T,p)\restriction D$.

Note that for a given model of computation and a fixed interpretation function, there is some freedom to choose D. E.g., usually, for ordinary Turing com-

[5] The Gödel pairing function is an absolutely definable class bijection between Ord and Ord^2; cf. [12, pp. 30–31].

putations with the usual textbook interpretation, we let $D = \Sigma^*$ and thus, computability is a property of partial functions $f\colon \Sigma^* \dashrightarrow \Sigma^*$. However, we could consider $D = \Sigma^\omega$, i.e., letting the Turing machine operate on arbitrary tape contents of length ω, obtaining a different model of computability.[6] On the other hand, if you fix the model of computation and the type of interpretation function, D cannot be chosen entirely freely: the class D needs to be closed under the operations $I(T,p)$ for $p \in P$. E.g., if our model of computation is Koepke machines with the usual interpretation, we cannot choose $D = \Sigma^*$ or even $D = \Sigma^\omega$ since there are programs with which a Koepke machine would produce an output that is not in D anymore.

In this paper, we shall consider two types of interpretation function, the *finite interpretation* I_f and the *transfinite interpretation* I_t (for definitions, cf. § 4).

4 Concrete models of computation

Programs. We fix three *motion tokens* $\mathrm{MT} := \{\blacktriangleleft, \blacktriangledown, \blacktriangleright\}$ that represent the instructions for the head movements ("move left", "do nothing", and "move right") and use $\mathbf{m}$ as variable for motion tokens. Among the states (indexed by ordinals), we single out three particular states: the *start state* indexed by 0, the *halt state* indexed by 1, and the *limit state* indexed by 2. We write $\Sigma_\circ := \Sigma \cup \{\circ\}$ where $\circ$ is a special symbol representing an empty cell. All of our programs will be partial functions $p\colon \mathrm{Ord} \times \Sigma_\circ \dashrightarrow \mathrm{Ord} \times \Sigma_\circ \times \mathrm{MT}$. We call a program *finite* if its domain is finite and *transfinite* if its domain is a set. The classes of finite and transfinite programs are denoted by P_f and P_t, respectively.

Via the canonical identification of the classes Ord, $\mathrm{Ord} \times \Sigma_\circ$, and $\mathrm{Ord} \times \Sigma_\circ \times \mathrm{MT}$, we can encode programs as elements of $\Sigma^{(\mathrm{Ord})}$. Under our encoding, we identify P_f with the class Σ^{FS} and P_t with the class $\Sigma^{<\mathrm{Ord}}$.

Transition functions. Given a program p, we shall now define the transition functions T_f ("finite transition function") and T_t ("transfinite transition function"). They are identical on sequences of successor length and coincide there with the ordinary transition function defined by Turing for his machines; they differ for sequences of limit length.

If $\vec{s} = (s_\xi\,;\, \xi < \gamma + 1)$ is a sequence of snapshots of successor length, the transition function will only depend on $s_\gamma = (x, \alpha, \beta)$, the final snapshot in the list. Thus, $x \in \Sigma^{(\mathrm{Ord})}$ is the tape content at time γ, α is the state at time γ, and β is the location of the head at time γ. If $p(\alpha, x(\beta))$ is undefined, we let $T(\vec{s}) := s_\gamma$; otherwise, let $p(\alpha, x(\beta)) = (\delta, \sigma, \mathbf{m})$. Then $T(\vec{s}) = (y, \alpha^+, \beta^+)$ where $\alpha^+ := \delta$,

[6] This is a curious model of computability that exhibits a discrepancy between 1-tape and 3-tape machines (cf. Footnote 4): since only finitely many cells are changed in halting computations, for 1-tape machines the identity function is computable and constant functions are not; in contrast, for 3-tape machines constant functions with value $w \in \Sigma^*$ are computable, but other constant functions or the identity function are not.

$$y(\eta) := \begin{cases} x(\eta) & \text{if } \eta \neq \beta, \\ \sigma & \text{if } \eta = \beta, \end{cases} \quad \text{and } \beta^+ := \begin{cases} \beta - 1 & \text{if } \mathbf{m} = \blacktriangleleft \text{ and } \beta \text{ is a successor,} \\ 0 & \text{if } \mathbf{m} = \blacktriangleleft \text{ and } \beta \text{ is a limit,} \\ \beta + 1 & \text{if } \mathbf{m} = \blacktriangleright, \\ \beta & \text{if } \mathbf{m} = \blacktriangledown. \end{cases}$$

If $\vec{s} = (s_\xi\,;\,\xi < \lambda)$ with $s_\xi = (x_\xi, \alpha_\xi, \beta_\xi)$ is a sequence of snapshots of limit length λ, the two transition functions agree in their definition of the tape content, but disagree in their treatment of the head position and state. Let us write $T_{\mathrm{f}}(\vec{s}) = (y, \alpha_{\mathrm{f}}, \beta_{\mathrm{f}})$ and $T_{\mathrm{t}}(\vec{s}) = (y, \alpha_{\mathrm{t}}, \beta_{\mathrm{t}})$. For the tape content, we assume that we have a total ordering on Σ and define $y(\eta) := \liminf\{x_\xi(\eta)\,;\,\xi < \lambda\}$.

The finite transition function T_{f} moves the head to cell 0, moves to the limit state (indexed by 2), i.e., $\alpha_{\mathrm{f}} := 2$ and $\beta_{\mathrm{f}} := 0$. Note that in any computation using the finite transition function, the head will never reach a cell indexed by an infinite ordinal.

The transfinite transition function T_{t} moves both the head and the cell to the *inferior limit* of the ordinals occurring in the sequence, i.e., $\alpha_{\mathrm{t}} := \liminf\{\alpha_\xi\,;\,\xi < \lambda\}$ and $\beta_{\mathrm{t}} := \liminf\{\beta_\xi\,;\,\xi < \lambda \wedge \alpha_\xi = \alpha_{\mathrm{t}}\}$.

Interpretations. We define our two interpretation functions uniformly for arbitrary tape contents $x \in \Sigma^{(\mathrm{Ord})}$. Both interpretations take a tape content x and a program p, define the initial snapshot $s := (x, 0, 0)$, and produce the computation $C_{p,s}$ of program p with initial snapshot s.

The *finite interpretation* I_{f} considers a computation as *halting* if there is a natural number n such that the state of $C_{p,s}(n)$ is 1 (i.e., the halting state); the *transfinite interpretation* I_{t} considers a computation as *halting* if there is an ordinal α such that the state of $C_{p,s}(\alpha)$ is 1. If it exists, the smallest such number is called the *halting time* of the computation. This implicitly defines the time considered by these models of computability: in general, we say that the *time relevant for a model of computability* is the supremum of its halting times. This is at most ω for models with the finite interpretation and at most Ord for models with the infinite interpretation. We let $\Omega \subseteq D$ be a subclass that is identified with the time relevant of the model, e.g., $\Omega = \Sigma^{\mathrm{O}}$ if the relevant time is Ord.

If a computation is halting, we say that the tape content at its halting time is the *output* of the computation. Finally, for $I = I_{\mathrm{f}}$ or $I = I_{\mathrm{t}}$ and the appropriate notion of halting, we let $I(T,p)(x) := y$ if $C_{p,s}$ is halting and y is its output, and $I(T,p)(x)\uparrow$ otherwise.

Models of computability. Using our finite specifications T_{f}, P_{f}, and I_{f} and our transfinite specifications T_{t}, P_{t}, and I_{t}, we can now recover the known models of computability (and a new one) as special cases.

First observe that if the transition function is finite, then any tape content beyond the cells indexed by natural numbers will be immaterial for the computation since the head never moves to these cells. So, the relevant input domain has to be Σ^* or Σ^ω. Moreover, if the interpretation function is finite, then all computations that go on to ω or beyond will be disregarded in the interpretation, so we can assume, without loss of generality, that the transition function is finite as well. This leads to the models of computability listed in Table 1.

	Transition	Programs	Interpretation	In-/Output	
(a)	Finite	Finite	Finite	Σ^*	ordinary Turing machines
(b)	Finite	Finite	Transfinite	Σ^ω	Hamkins-Kidder machines
(c)	Transfinite	Finite	Transfinite	$\Sigma^{<\mathrm{Ord}}$	Koepke machines
(d)	Transfinite	Transfinite	Transfinite	$\Sigma^{<\mathrm{Ord}}$	symmetric machines; cf. § 5

Table 1. The considered models of computability

We briefly discuss the choice of in-/output for the described models:

Table 1 (a). Since the transition function is finite, only Σ^* and Σ^ω make sense as choice of in-/output. However, since the interpretation is also finite, no halting computation will ever be able to read an entire infinite tape, so Σ^* is the natural choice for in-/output. Choosing Σ^ω leads to the model of computability discussed in Footnote 6. The time relevant for this model of ω.

Table 1 (b). Similarly in this case, the finite transition function means that we can only choose Σ^* or Σ^ω as input; however, Σ^* is not closed under the operation of the interpretation (a Hamkins-Kidder machine can fill the entire tape and then halt), so Σ^ω is the only remaining natural choice. The time relevant for Hamkins-Kidder machines has been investigated in [9, 22].

Table 1 (c). In analogy to the argument given for line (a), a halting Koepke machine will only consider a set of cells on the tape. Thus, the natural choice of in-/output is $\Sigma^{<\mathrm{Ord}}$. Similarly to (a), it makes sense to consider $D = \Sigma^{(\mathrm{Ord})}$ in which case the discussion of Footnote 6 applies. The time relevant for Koepke computability is the class Ord of all ordinals. If $p \in \Sigma^{<\mathrm{Ord}}$, we say that a partial function $f\colon \Sigma^{<\mathrm{Ord}} \dashrightarrow \Sigma^{<\mathrm{Ord}}$ is *Koepke computable with parameter* p if the partial function $p * w \mapsto f(w)$ is Koepke computable. We shall prove in Proposition 4 that this notion is equivalent to the new notion introduced in line (d).

Table 1 (d). The notion of computability introduced in line (d), called *symmetric computability*, corrects the lack of symmetry for transfinite computability. In this model, time, space, and programs are all transfinite. The time relevant for symmetric computability is the class Ord of all ordinals.

5 Symmetric machines

In Table 1, we defined the *model of symmetric computability* to be given by the transfinite transition function, transfinite programs, and the transfinite interpretation, using $\Sigma^{<\mathrm{Ord}}$ as input and output. We call the corresponding model of computation *symmetric machines*. In contrast to Hamkins-Kidder machines (who have considerably more time than space) and Koepke machines (whose programs are tiny compared to the time and space they have available), symmetric machines have set-sized time, space, and programs. They are the model of computability that systematically replaces the word "finite" in ordinary Turing computation with "set-sized".

Proposition 3. If $f : \Sigma^{<\mathrm{Ord}} \dashrightarrow \Sigma^{<\mathrm{Ord}}$ is a set, then f is symmetrically computable.

Proof sketch. Clearly, if $x \in \Sigma^{<\mathrm{Ord}}$, there is a transfinite program p that produces x upon empty input (just explicitly specify the values of x).

Since f is a set, find some ξ and $\Sigma^{(\xi)} := \{w \in \Sigma^{(\mathrm{Ord})} \ \mathrm{dom}(w) \subseteq \xi\}$ such that both $\mathrm{dom}(f) \subseteq \Sigma^{(\xi)}$ and $\mathrm{ran}(f) \subseteq \Sigma^{(\xi)}$. By AC, let g be a bijection between some ordinal λ and $\Sigma^{(\xi)}$. We define $x_f \colon \lambda \cdot \xi \cdot 2 \dashrightarrow \Sigma$ by letting $x_f(\alpha, \beta, 0) := g(\alpha)(\beta)$ and $x_f(\alpha, \beta, 1) := f(g(\alpha))(\beta)$.

Now find a program that writes x_f on the tape. Upon input $w \in \Sigma^{(\xi)}$, we can now determine $f(w)$ as follows: search through the 0-components of x_f until you find w; when you found w at index α, output $\{(\beta, x_f(\alpha, \beta, 1))\,;\, \beta < \xi\}$. Q.E.D.

Proposition 4. A partial function $f \colon \Sigma^{<\mathrm{Ord}} \dashrightarrow \Sigma^{<\mathrm{Ord}}$ is symmetrically computable if and only of there is a $p \in \Sigma^{<\mathrm{Ord}}$ such that f is Koepke computable in parameter p.

Proof sketch. By Proposition 3, any parameter p is symmetrically computable, so if f is Koepke computable in parameter p, it is symmetrically computable as follows: upon input w, first compute p, then $w * p$, then $f(p * w)$.

For the other direction, observe that in terms of hardware and interpretation, Koepke machines are just symmetric machines. Therefore, the universal Koepke machine is also a universal symmetric machine, i.e., there is a Koepke machine u such that for all transfinite programs p and all $w \in \Sigma^{<\mathrm{Ord}}$, we have

$$I_{\mathrm{t}}(T_{\mathrm{t}}, u)(p * w) = I_{\mathrm{t}}(T_{\mathrm{t}}, p)(w).$$

Thus, if a partial function f is symmetrically computed by a program p, it is Koepke computable with parameter p. Q.E.D.

As usual, we can define the *halting problem* by

$$K := \{v * w \in \Sigma^{<\mathrm{Ord}}\,;\, I_{\mathrm{t}}(T_{\mathrm{t}}, v)(w)\downarrow\}$$

(where v is interpreted as a transfinite program). The usual proof shows that K is not symmetrically computable.

We shall now have a closer look at the symmetry properties for symmetric computability and ask whether it is the precise analogue of the symmetry exhibited by ordinary Turing computability. For ordinary Turing machines, time and space are indexed by natural numbers; however, programs and in-/output are not *prima facie* natural numbers; they are finite sequences of elements of a finite set, i.e., via some encoding elements of Σ^*. In this case, the symmetry is given by the fact that there is a computable encoding function that identifies Σ^* and ω. Via such an encoding, we can see all four different parameters of the model of computability as the same type of object.

In the case of symmetric computation, the word "finite" is systematically replaced by "set-sized", so time and space are indexed by ordinals and programs and in-/outputs are (up to encoding) elements of $\Sigma^{<\mathrm{Ord}}$. Alas, in general, we cannot identify $\Sigma^{<\mathrm{Ord}}$ and Ord: The encodings between the classes Ord, $\mathrm{Ord}^{<\omega}$, Σ^{O}, and Σ^{FS} mentioned in § 3 can be performed by Koepke machines (see, e.g., [13, Section 4]), but the class $\Sigma^{<\mathrm{Ord}}$ is a very different type of object: among other things, it contains the entire Cantor space (functions $f:\omega \to \Sigma$), so any computable encoding of elements $\Sigma^{<\mathrm{Ord}}$ as ordinals would yield a computable wellordering of the reals. As a consequence, the existence of such a class function cannot be proved without additional set theoretic assumptions.

6 The symmetry condition

We give definitions of the notions of *semidecidability* and *computable enumerability* within our abstract framework. For the model of ordinary Turing computability, these definitions coincide with the usual definitions.

Definition 5. Suppose that a model of computability is given by a transition function T, a class of programs P, and an interpretation I with domain class D. Let $A \subseteq D$ be a non-empty class and let ψ_A be a function such that $\psi_A(w) = \mathbf{0}$ if $w \in A$ and $\psi_A(w)\uparrow$ otherwise (the *pseudocharacteristic function*). Then A is called *semidecidable* if ψ_A is computable. Fixing some $\Omega \subseteq D$ representing the relevant time of the model, we say that A is *computably enumerable* if there is a program $p \in P$ such that $A = \{I(T,p)(w)\,;\, w \in \Omega\}$.

Theorem 6 (Folklore)**.** For the model of ordinary Turing computability and any non-empty set $A \subseteq \Sigma^*$, the following hold:

(i) the set A is semidecidable if and only if it is the range of a partial computable function $f: \Sigma^* \dashrightarrow \Sigma^*$ and
(ii) the set A is computably enumerable if and only if it is semidecidable.

The equivalence (i) is a classical textbook argument [19, Theorem V]; in equivalence (ii), the forwards direction is a trivial consequence of (i) and the backwards direction uses the computable bijection between Σ^* and the relevant time ω. So, adapting this proof to the case of symmetric computability will preserve the equivalence (i) and the forwards direction of (ii).

Theorem 7. For the model of symmetric computability and any non-empty class $A \subseteq \Sigma^{<\mathrm{Ord}}$, the following hold:

(i) the class A is semidecidable if and only if it is the range of a computable class function $f: \Sigma^{<\mathrm{Ord}} \dashrightarrow \Sigma^{<\mathrm{Ord}}$ and

(ii) if the class A is computably enumerable, then it is semidecidable.

In comparison to Theorem 6, the converse of (ii) is missing in Theorem 7; it turns out that this difference is crucial for our quest for the desired symmetry from § 5. We write SC for the statement "the class $\Sigma^{<\mathrm{Ord}}$ is symmetrically computably enumerable", call this the *symmetry condition*, and note that it is a set theoretic sentence in the sense of § 2.

Proposition 8. The symmetry condition SC is equivalent to the statement "every symmetrically semi-decidable class is symmetrically computably enumerable".

Proof sketch. Clearly, $\Sigma^{<\mathrm{Ord}}$ is semi-decidable, so "$\Leftarrow$" is obvious. For "$\Rightarrow$", let $g : \Sigma^{\mathrm{O}} \to \Sigma^{<\mathrm{Ord}}$ be a computable enumeration and A be any semi-decidable class By Theorem 7, we have a computable surjection $f : \Sigma^{<\mathrm{Ord}} \to A$. Then $f \circ g$ enumerates A. Q.E.D.

The symmetry condition expresses that the classes $\Sigma^{<\mathrm{Ord}}$ and Ord can be identified via the computable listing provided by SC. Therefore, assuming SC, time, space, programs, and in-/outputs can be considered the same type of object, and the model of symmetric computability has the symmetry exhibited by ordinary Turing computability.

We shall now see that SC is independent from NBG and characterise under which circumstances SC holds. A crucial ingredient to prove our characterisation is the following result which is a straightforward relativisation of [13, Theorem 6.2].

Theorem 9 (Koepke)**.** Let x be a set of ordinals. Then any $w \in \Sigma^{<\mathrm{Ord}}$ is in $\mathrm{L}[x]$ if and only if there is a finite program p and $v \in \Sigma^{\mathrm{O}}$ such that the Koepke computation of p with input v and parameter x halts and produces the output w.

Proof sketch. The backwards direction follows from the fact that a Koepke computation from a parameter x is absolutely defined. Thus, if a Koepke machine produces the output w upon input $x * v$, then w lies in every model containing both x and v. Since $v \in \Sigma^{\mathrm{O}} \subseteq \mathrm{L}[x]$, we have $w \in \mathrm{L}[x]$. For the forwards direction, assume $w \in \mathrm{L}[x]$ and let α be an exponentially closed ordinal such that $w \in \mathrm{L}_\alpha[x]$. Then by [15, Theorem 7 (a)], w is α-Koepke computable, and thus Koepke computable from the parameter giving α, i.e., Koepke computable from a parameter in $\Sigma^{\mathrm{O}} \subseteq \Sigma^{\mathrm{FS}}$. Q.E.D.

Lemma 10. If x is a set of ordinals and $\Sigma^{<\mathrm{Ord}} \subseteq \mathrm{L}[x]$, then $\mathrm{V}=\mathrm{L}[x]$.

Proof sketch. Assume that $\Sigma^{<\mathrm{Ord}} \subseteq \mathrm{L}[x]$. Assume by contradiction that $\mathrm{V} \neq \mathrm{L}[x]$. Let A be an $\in$-minimal set not in $\mathrm{L}[x]$, i.e., $A \notin \mathrm{L}[x]$, but $A \subseteq \mathrm{L}[x]$. There is a bijection $G : \mathrm{Ord} \to \mathrm{L}[x]$ definable from x, (cf., e.g., [12, p. 193]). Define $w(\alpha){\downarrow} = \mathbf{0}$ if and only if $G(\alpha) \in A$; then $w \in \Sigma^{<\mathrm{Ord}} \subseteq \mathrm{L}[x]$. But then $A = \{G(\alpha) \in \mathrm{L}[x]; w(\alpha) = \mathbf{0}\}$ and therefore $A \in \mathrm{L}[x]$. Q.E.D.

Theorem 11. The symmetry condition SC holds if and only if the universe is constructible from a set of ordinals.

Proof sketch. For "(ii)⇒(i)", use the (computable) Gödel pairing function to get a computable bijection $C : \Sigma^{\mathrm{O}} \to \Sigma^{\mathrm{O}} \times \Sigma^{\mathrm{FS}}$ and identify the finite programs with Σ^{FS}. If $u \in \Sigma^{\mathrm{O}}$, let $C(u) = (v, p)$, and let $F(u)$ be the result of running the finite program p on input v. By Theorem 9, F enumerates $\Sigma^{<\mathrm{Ord}}$.

For "(i)⇒(ii)", assume SC. Let p be the program of the computable enumeration of $\Sigma^{<\mathrm{Ord}}$ (which can be encoded as a set of ordinals). By Lemma 10, it is enough to show that $\Sigma^{<\mathrm{Ord}} \subseteq \mathrm{L}[p]$. But this follows from the fact that p defines a class surjection from Ord onto $\Sigma^{<\mathrm{Ord}}$. Q.E.D.

It follows from Theorems 2 & 11 that SC is independent from NBGC. We note that in the special case of V=L, Koepke computability and symmetric computability are equivalent (cf. [1, Exercise 3.9.7 (d)]); however, if we take any nonconstructible set of ordinals z, then $\mathrm{L}[z]$ is a model of SC by Theorem 11, the set z is symmetrically computable (by Proposition 3), but not Koepke computable by Theorem 9 (letting $x = \varnothing$), so the two models of computability are different. This also answers [16, Question 5.12] about separating the stronger versions SC_κ ("$\Sigma^{<\mathrm{Ord}}$ is computably enumerable by a program of size $<\kappa$"): e.g., if x is a non-constructible real, then $\mathsf{SC}_{\aleph_1}$ holds in $\mathrm{L}[x]$, but not $\mathsf{SC}_{\aleph_0}$.

References

1. M. Carl. *Ordinal Computability. An Introduction to Infinitary Machines*, volume 9 of *De Gruyter Series in Logic and Its Applications*. De Gruyter, 2019.
2. M. Carl. Space-bounded OTMs and REG^∞. *Computability*, 11:41–56, 2022.
3. M. Carl, L. Galeotti, and R. Paßmann. Realisability for infinitary intuitionistic set theory. *Ann. Pure Appl. Log.* to appear, arxiv:2009.12172.
4. M. Carl, B. Löwe, and B. Rin. Koepke machines and satisfiability for infinitary propositional languages. In J. Kari, F. Manea, and I. Petre, editors, *Unveiling Dynamics and Complexity, 13th Conference on Computability in Europe, CiE 2017, Turku, Finland, June 12-16, 2017, Proceedings*, volume 10307 of *Lecture Notes in Computer Science*, pages 187–197. Springer, 2017.
5. V. Deolalikar, J. D. Hamkins, and R. Schindler. $\mathbf{P} \neq \mathbf{NP} \cap \mathbf{co\text{-}NP}$ for infinite time Turing machines. *J. Log. Comput.*, 15(5):577–592, 2005.
6. U. Felgner. Choice functions on sets and classes. In G. H. Müller, editor, *Sets and Classes: On The Work by Paul Bernays*, volume 84 of *Studies in Logic and the Foundations of Mathematics*, pages 217–255. Elsevier, 1976.
7. J. Ferreirós. *Labyrinth of Thought: A History of Set Theory and Its Role in Modern Mathematics*. Birkhäuser Basel, Basel, 2007.
8. L. Galeotti. Surreal Blum-Shub-Smale machines. In F. Manea, B. Martin, D. Paulusma, and G. Primiero, editors, *Computing with Foresight and Industry, 15th Conference on Computability in Europe, CiE 2019, Durham, UK, July 15–19, 2019, Proceedings*, volume 11558 of *Lecture Notes in Computer Science*, pages 13–24. Springer, 2019.
9. J. D. Hamkins and A. Lewis. Infinite time Turing machines. *J. Symb. Log.*, 65(2):567–604, 2000.

10. J. D. Hamkins and D. E. Seabold. Infinite time Turing machines with only one tape. *Mathematical Logic Quarterly*, 47(2):271–287, 2001.
11. J. D. Hamkins and P. D. Welch. $\mathbf{P}^f \neq \mathbf{NP}^f$ for almost all f. *Math. Log. Q.*, 49(5):536–540, 2003.
12. T. S. Jech. *Set Theory*. Springer Monographs in Mathematics. Springer-Verlag, third millenium edition, 2003.
13. P. Koepke. Turing computations on ordinals. *Bull. Symb. Log.*, 11(3):377—397, 2005.
14. P. Koepke. Ordinal computability. In K. Ambos-Spies, B. Löwe, and W. Merkle, editors, *Mathematical Theory and Computational Practice, 5th Conference on Computability in Europe, CiE 2009, Heidelberg, Germany, July 19-24, 2009. Proceedings*, volume 5635 of *Lecture Notes in Computer Science*, pages 280–289. Springer, 2009.
15. P. Koepke and B. Seyfferth. Ordinal machines and admissible recursion theory. *Annals of Pure and Applied Logic*, 160(3):310–318, 2009.
16. E. S. Lewis. Computation with infinite programs. Master's thesis, Universiteit van Amsterdam, 2018. ILLC Publications MoL-2018-14.
17. B. Löwe. Space bounds for infinitary computation. In A. Beckmann, U. Berger, B. Löwe, and J. V. Tucker, editors, *Logical Approaches to Computational Barriers, Second Conference on Computability in Europe, CiE 2006, Swansea, UK, July 2006, Proceedings*, volume 3988 of *Lecture Notes in Computer Science*, pages 319–329, 2006.
18. E. Mendelson. *Introduction to Mathematical Logic*. Textbooks in Mathematics. CRC Press, 6 edition, 2015.
19. H. Rogers. *Theory of Recursive Functions and Effective Computability*. MIT Press, 1987.
20. R. Schindler. $\mathbf{P} \neq \mathbf{NP}$ for infinite time Turing machines. *Monatsh. Math.*, 139:335–340, 2003.
21. A. M. Turing. On computable numbers, with an application to the *Entscheidungsproblem*. *Proceedings of the London Mathematical Society*, 42:230–265, 1937.
22. P. D. Welch. Characteristics of discrete transfinite time Turing machine models: halting times, stabilization times, and normal form theorems. *Theor. Comput. Sci.*, 410:426–442, 2009.
23. J. Winter. Space complexity in infinite time Turing machines. Master's thesis, Universiteit van Amsterdam, 2007. ILLC Publications MoL-2007-14.
24. J. Winter. Is $\mathbf{P} = \mathbf{PSPACE}$ for infinite time Turing machines? In M. Archibald, V. Brattka, V. Goranko, and B. Löwe, editors, *Infinity in Logic and Computation, International Conference, ILC 2007, Cape Town, South Africa, November 3–5, 2007, Revised Selected Papers*, volume 5489 of *Lecture Notes in Artificial Intelligence*, pages 126–137, 2009.