THE BULLETIN OF SYMBOLIC LOGIC
Volume 11, Number 3, Sept. 2005

TURING COMPUTATIONS ON ORDINALS

PETER KOEPKE

Abstract. We define the notion of *ordinal computability* by generalizing standard TURING computability on tapes of length ω to computations on tapes of arbitrary ordinal length. We show that a set of ordinals is ordinal computable from a finite set of ordinal parameters if and only if it is an element of GÖDEL's constructible universe L. This characterization can be used to prove the generalized continuum hypothesis in L.

§1. Introduction. A standard TURING computation may be visualized as a time-like sequence of elementary *read-write-move* operations carried out by one or more "heads" on "tapes". The sequence of actions is determined by the initial tape contents and by a finite TURING *program*. The specific choice of alphabet, operations and tapes may influence the time or space complexity of calculations; by the CHURCH-TURING thesis, however, the associated notion of TURING *computability* is not affected. So we may assume that TURING machines act on tapes whose cells are indexed by the set ω $(= \mathbb{N})$ of *natural numbers* $0, 1, \dots$ and contain 0's or 1's.

			SPACE								
		0	1	2	3	4	5	6	7	...	...
	0	**1**	0	0	1	1	1	0	0	0	0
TIME	1	0	**0**	0	1	1	1	0	0		
	2	0	0	**0**	1	1	1	0	0		
	3	0	**0**	1	1	1	1	0	0		
	4	0	1	**1**	1	1	1	0	0		
	$\vdots$										
	n	1	1	1	1	**0**	1	1	1		
	$n+1$	1	1	1	1	1	**1**	1	1		
	$\vdots$										

A standard TURING *computation. Head positions are indicated by* ***boldface***.

An obvious generalization from the perspective of transfinite ordinal theory is to extend TURING calculations to tapes whose cells are indexed by

Received October 3, 2004; accepted March 8, 2005.

the class Ord of all *ordinal numbers*. Calculations will become (infinite) sequences of elementary tape operations indexed by ordinals which may be viewed as instances of time. For *successor* ordinals (or times) calculations will basically be defined as for standard TURING machines. At *limit* ordinals we define the tape contents, program states and head positions by appropriate limit operations which may be viewed as *inferior limits*.

		Ordinal Space …													
		0	1	2	3	4	5	6	7	…	…	ω	…	α	…
Ordinal Time …	0	**1**	1	0	1	0	0	1	1	…	…	1	…	1	0
	1	0	**1**	0	1	0	0	1	1			1			
	2	0	0	**0**	1	0	0	1	1			1			
	3	0	**0**	0	1	0	0	1	1			1			
	4	0	0	**0**	0	0	0	1	1			1			
	⋮														
	n	1	1	1	1	0	1	0	**1**			1			
	n+1	1	1	1	1	1	1	**0**	1			1			
	⋮	⋮	⋮	⋮	⋮	⋮									
	ω	0	0	1	0	0	0	1	1	…	…	**1**			
	$\omega+1$	**0**	0	1	0	0	0	1	1			0			
	⋮														
	θ	1	0	0	1	1	1	1	0	…	…	…	…	**0**	…
	⋮			⋮			⋮			⋮	⋮				
	⋮														

An ordinal computation.

The corresponding notion of *ordinal computability* obviously extends TURING computability. By the CHURCH-TURING thesis many operations on natural numbers are ordinal computable. The *ordinal arithmetical* operations (addition, multiplication, exponentiation) and other basic operations on ordinals are also ordinal computable.

Indeed, the recursive properties of the family of ordinal computable functions are so strong that the *bounded truth predicate*

$$\{\, (\alpha, \varphi, \vec{x}) \mid \alpha \in \mathrm{Ord},\ \varphi \text{ an } \in\text{-formula},\ \vec{x} \in L_\alpha,\ L_\alpha \models \varphi(\vec{x}) \,\}$$

for GÖDEL's *constructible hierarchy* $L = \bigcup_{\alpha \in \mathrm{Ord}} L_\alpha$ is ordinal computable given some appropriate coding. As a corollary we obtain the main result characterizing ordinal computability:

THEOREM 1.1. *A set* $x \subseteq \mathrm{Ord}$ *is ordinal computable from finitely many ordinal parameters if and only if* $x \in L$.

The implication from left to right will be immediate from the set-theoretical absoluteness of ordinal computations. The converse requires a careful analysis of the iterative definition of the constructible hierarchy to show that the iteration can be carried out by an ordinal TURING machine.

This theorem may be viewed as an analogue of the CHURCH-TURING thesis: ordinal computability defines a natural and absolute class of sets, and it is stable with respect to technical variations in its definition.

Theories of transfinite computations which extend TURING computability have been proposed and studied for some time. *Higher recursion theory* as described in the monograph [7] of GERALD SACKS culminates in E*-recursion* which defines a computational result $\{e\}(x)$ for programs e built from basic set functions applied to arbitrary sets x. The relation of E-computability to constructibility is analogous to the statement of Theorem 1.1. In computer science various infinitary machines like BÜCHI automata [1] have been defined yielding important applications. The novelty here is in keeping with the original TURING idea of reading and writing on an unrestricted tape while interpreting unrestrictedness in the widest sense as set-theoretical unboundedness.

Our work was inspired by the *infinite time* TURING machines introduced by JOEL D. HAMKINS, JEFF KIDDER and ANDY LEWIS [4]. Infinite time TURING machines use standard tapes indexed by natural numbers but allow infinite computation sequences. At limit times, tape contents are defined as inferior limits of previous contents. Inputs and outputs are characteristic functions on the set ω of all natural numbers and may thus be viewed as *real numbers*. The theory of infinite time TURING machines is naturally related to definability theory over the structure $(\mathbb{R}, \dots)$, i.e., to *descriptive set theory*. In the case of tapes of arbitrary ordinal length one is lead to consider a theory of arbitrarily iterated definitions, i.e., *constructibility theory*.

There is also a link between ordinal computability and SILVER machines which were defined by JACK H. SILVER to — in his words — "eliminate the fine structure from the work of Jensen" [6], [8]. A SILVER machine is not a machine in the sense that it corresponds to some idealized physical device but rather a family of SKOLEM functions and other basic functions which can be used to form SKOLEM hulls similar to the Σ_1-hulls in the fine structure theory of RONALD B. JENSEN [5]. An issue relating the present work to the theory of SILVER machines is SILVER's recursive definition of a bounded truth predicate for L.

§**2. Ordinal Turing Machines.** We give an intuitive description of ordinal computations which will subsequently be formalized. Consider a tape of ordertype Ord, i.e., a sequence indexed by the class Ord of all ordinals. The cells of the tape can be identified with the ordinals, every cell can contain a 0 or a 1 where 0 is the default state. A *read-write* head moves on the tape, starting at cell 0. The computation is steered by a *program* which consists of a finite sequence of *commands* indexed by natural numbers. The indices of the commands can be seen as *states* of the machine.

A computation of the machine is a sequence of machine configurations which are indexed by ordinal "times" $0, 1, 2, \ldots, \omega, \omega+1, \ldots$. At time t the read-write head reads the content of the cell at its position. According to the content and the present machine state the head writes a "0" or a "1" and then moves to the right or to the left. Also the machine changes into a new program state.

So far we have described the computation rules of finitary TURING machines. *Ordinal computations* require the specification of the behaviour at *limit ordinals*; we shall base the limit rules on simple limit operations.

Assume that at time t the head position is $H(t)$. After a *move-right* command we put $H(t+1) = H(t) + 1$. After a *move-left* command we move one cell to the left if possible and otherwise, if $H(t)$ is a limit ordinal or 0, jump to the default position 0:

$$H(t+1) = \begin{cases} H(t) - 1, & \text{if } H(t) \text{ is a successor ordinal;} \\ 0, & \text{else.} \end{cases}$$

The definition of $H(t)$ for t a *limit ordinal* will be given later.

At time t the tape will be identified with a *tape content*

$$T(t) = \big(T(t)_0, T(t)_1, \ldots, T(t)_\omega, T(t)_{\omega+1}, \ldots\big)$$

which is a sequence of *cell contents* $T(t)_\alpha \in \{0, 1\}$. It is determined by previous write operations. For limit times t the content $T(t)_\alpha$ of the αth cell is determined as follows: if the cell content $T(s)_\alpha$ stabilizes at a constant value as s approaches t we let $T(t)_\alpha$ be that value; otherwise we take the default value $T(t)_\alpha = 0$. Formally this is an *inferior limit*:

$$T(t)_\alpha = \liminf_{s \to t} T(s)_\alpha.$$

A lim inf rule will also be used for the program state and the head location at limit times. Let $S(t)$ be the program state at time t. For limit times t set

$$S(t) = \liminf_{s \to t} S(s).$$

Finally the head position $H(t)$ for limit times t is

$$H(t) = \liminf_{s \to t, S(s) = S(t)} H(s).$$

The definitions of $S(t)$ and $H(t)$ can be motivated as follows. Since a TURING program is finite its execution will lead to some (complex) looping structure involving loops, subloops and so forth. This can be presented by pseudo-code like:

```
       ...
17:begin loop
           ...
   21:   begin subloop
              ...
   29:   end subloop
           ...
32:end loop
       ...
```

Assume that for times $s \to t$ the loop (17–32) with its subloop (21–29) is traversed cofinally often. Then at limit time t it is natural to put the machine at the start of the "main loop". Assuming that the lines of the program are enumerated in increasing order this corresponds to the lim inf rule

$$S(t) = \liminf_{s \to t} S(s).$$

The canonical head location $H(t)$ is then determined as the inferior limit of all head locations when the program is at the start of the "main loop". If the head is for example moving linearly towards a limit location, say $H(s_0 + i) = h_0 + i$ for $i < \lambda$, we will have $H(s_0 + \lambda) = h_0 + \lambda$. Note that the limit behaviour of the head position is defined differently for infinite time TURING machines which do not possess limit positions on the tape; there the head simply falls back to 0 at limit times.

The above intuitions are formalized as follows.

DEFINITION 2.1. (a) A **command** is a 5-tuple $C = (s, c, c', m, s')$ where $s, s' \in \omega$ and $c, c', m \in \{0, 1\}$; the natural number s is the **state** of the command C. The intention of the command C is that if the machine is in state s and reads the symbol c under its read-write head, then it writes the symbol c', moves the head left if $m = 0$ or right if $m = 1$, and goes into state s'. States correspond to the "line numbers" of some programming languages.

(b) A **program** is a finite set P of commands satisfying the following structural conditions:

  (i) If $(s, c, c', m, s') \in P$ then there is $(s, d, d', n, t') \in P$ with $c \neq d$; thus in state s the machine can react to reading a "0" as well as to reading a "1".

  (ii) If $(s, c, c', m, s') \in P$ and $(s, c, c'', m', s'') \in P$ then $c' = c'', m = m', s' = s''$; this means that the course of the computation is completely determined by the sequence of program states and the initial cell contents.

(c) For a program P let

$$\text{states}(P) = \{ s \mid (s, c, c', m, s') \in P \}$$

be the set of program states.

DEFINITION 2.2. Let P be a program. A triple

$$S\colon \theta \to \omega, \quad H\colon \theta \to \mathrm{Ord}, \quad T\colon \theta \to ({}^{\mathrm{Ord}}2)$$

is an **ordinal computation** by P if the following hold:

(a) θ is a successor ordinal or $\theta = \mathrm{Ord}$; θ is the **length** of the computation.
(b) $S(0) = H(0) = 0$; the machine starts in state 0 with head position 0.
(c) If $t < \theta$ and $S(t) \notin \mathrm{state}(P)$ then $\theta = t + 1$; the machine **stops** if the machine state is not a program state of P.
(d) If $t < \theta$ and $S(t) \in \mathrm{state}(P)$ then $t + 1 < \theta$; choose the unique command $(s, c, c', m, s') \in P$ with $S(t) = s$ and $T(t)_{H(t)} = c$; this command is executed as follows:

$$T(t+1)_\xi = \begin{cases} c', & \text{if } \xi = H(t); \\ T(t)_\xi, & \text{else;} \end{cases}$$

$$S(t+1) = s';$$

$$H(t+1) = \begin{cases} H(t)+1, & \text{if } m = 1; \\ H(t)-1, & \text{if } m = 0 \text{ and } H(t) \text{ is a successor ordinal;} \\ 0, & \text{else.} \end{cases}$$

(e) If $t < \theta$ is a limit ordinal, the machine constellation at t is determined by taking inferior limits:

$$\forall \xi \in \mathrm{Ord}\ \ T(t)_\xi = \liminf_{r \to t} T(r)_\xi;$$
$$S(t) = \liminf_{r \to t} S(r);$$
$$H(t) = \liminf_{s \to t, S(s) = S(t)} H(s).$$

The computation is obviously recursively determined by the initial tape contents $T(0)$ and the program P. We call it the **ordinal computation by P with input $T(0)$**. If the computation stops, $\theta = \beta + 1$ is a successor ordinal and $T(\beta)$ is the final tape content. In this case we say that P **computes** $T(\beta)$ from $T(0)$ and write $P\colon T(0) \mapsto T(\beta)$.

This interpretation of programs yields associated notions of computability.

DEFINITION 2.3. A partial function $F\colon {}^{\mathrm{Ord}}2 \rightharpoonup {}^{\mathrm{Ord}}2$ is **ordinal computable** if there is a program P such that $P\colon T \mapsto F(T)$ for every $T \in \mathrm{dom}(F)$.

By coding, the notion of ordinal computability can be extended to other domains. We can, e.g., *code* an ordinal $\delta \in \mathrm{Ord}$ by the characteristic function $\chi_{\{\delta\}}\colon \mathrm{Ord} \to 2$, $\chi_{\{\delta\}}(\xi) = 1$ if and only if $\xi = \delta$, and define:

DEFINITION 2.4. A partial function $F\colon \mathrm{Ord} \rightharpoonup \mathrm{Ord}$ is **ordinal computable** if the function $\chi_{\{\delta\}} \mapsto \chi_{\{F(\delta)\}}$ is ordinal computable.

We also consider computations involving finitely many ordinal *parameters*.

DEFINITION 2.5. A subset $x \subseteq \mathrm{Ord}$ is **ordinal computable from finitely many ordinal parameters** if there a finite subset $z \subseteq \mathrm{Ord}$ and a program P such that $P: \chi_z \mapsto \chi_x$.

In view of our intended applications of ordinal computations to models of set theory we note some absoluteness properties:

LEMMA 2.6. *Let $(M, \in)$ be a transitive model of* ZF^-, *i.e., of* ZERMELO-FRAENKEL *set theory without the powerset axiom. Let P be a program and let $T(0)$:* $\mathrm{Ord} \to 2$ *be an initial tape content so that $T(0)\restriction(\mathrm{Ord} \cap M)$ is definable in M. Let $S: \theta \to \omega$, $H: \theta \to \mathrm{Ord}$, $T: \theta \to ({}^{\mathrm{Ord}}2)$ be the ordinal computation by P with input $T(0)$. Then*:

(a) *The ordinal computation by P with input $T(0)$ is absolute for $(M, \in)$ below $(\mathrm{Ord} \cap M)$, i.e.,*

$$S: \theta \cap M \to \omega, \quad H: \theta \cap M \to \mathrm{Ord}, \quad \bar{T}: \theta \cap M \to ({}^{\mathrm{Ord} \cap M}2)$$

with $\bar{T}(t) = T(t)\restriction(\mathrm{Ord} \cap M)$ is the ordinal computation by P with input $T(0)\restriction(\mathrm{Ord} \cap M)$ as computed in the model $(M, \in)$.

(b) *If* $\mathrm{Ord} \subseteq M$ *then the ordinal computations by P in M and in the universe V are equal.*

(c) *Let* $\mathrm{Ord} \subseteq M$ *and $x, y \subseteq \mathrm{Ord}$, $x, y \in M$. Then $P: \chi_x \mapsto \chi_y$ if and only if $(M, \in) \vDash$ "$P: \chi_x \mapsto \chi_y$".*

(d) *Let $x, y \subseteq \mathrm{Ord}$, $x, y \in M$. Assume that $(M, \in) \vDash$ "$P: \chi_x \mapsto \chi_y$". Then $P: \chi_x \mapsto \chi_y$.*

The properties follow from the observation that the recursion in Definition 2.2 is clearly absolute between M and V. Note that the converse of d) is in general false. With the subsequent results on constructibility we could let $M = L_\delta$ be the minimal level of the constructible hierarchy which is a model of ZF^-. If P is a program which searches for the minimal ordinal δ such that L_δ is a ZF^--model then P will stop in V but not in M.

§3. Ordinal algorithms.

We present a number of fundamental algorithms which can be implemented as ordinal computations. Our emphasis is not on writing concrete *programs* as in Definition 2.1 but on showing that programs *exist*. It thus suffices to present basic ideas and algorithms together with methods to combine these into complex algorithms. We shall freely use informal "higher programming languages" to describe algorithms. Algorithms are based on data formats for the representation of input and output values. Again we shall not give detailed definitions but only indicate crucial features of the formats.

The intended computations will deal with ordinals and sequences of ordinals. The simplest way of representing the ordinal $\alpha \in \mathrm{Ord}$ in an ordinal machine is by a tape whose content is the characteristic function of $\{\alpha\}$:

$$\chi_{\{\alpha\}}: \mathrm{Ord} \to 2, \quad \chi_{\{\alpha\}}(\xi) = 1 \text{ if and only if } \xi = \alpha.$$

A basic task is to *find* or *identify* this ordinal α: initially the head is in position 0, it then moves to the right until it stops exactly at position α. This is achieved by the following program:

$$P = \{(0, 0, 0, 1, 0), (0, 1, 1, 1, 1), (1, 0, 0, 0, 2), (1, 1, 1, 0, 2)\}.$$

The program is in state 0 until it reads a 1, then it goes one cell to the right, one cell to the left, and stops because 2 is not a program state. Informally the algorithm may be written as

```
Find_Ordinal:
   if head = 1 then STOP otherwise moveright
```

Similarly one can let the head find (the beginning) of any finite 0-1-*bitstring* $b_0 \dots b_{k-1}$:

```
Find_Bitstring :
A:      if head = ‘b_0’ then moveright otherwise goto C0
        if head = ‘b_1’ then moveright otherwise goto C1
        ...
        if head = ‘b_{k-1}’ then goto B otherwise goto C(k-1)
B:      moveleft
        ...
        moveleft
        moveleft
        stop
C(k-1):moveleft
        ...
C1:     moveleft
C0:     moveright
        goto A
```

In view of this algorithm we may assume that the tape contains arbitrary *symbols* coded by finite bitstrings instead of single bits. Note that the above programs obviously perform the intended tasks on standard TURING machines. The limit rules are designed to lift this behaviour continuously to transfinite ordinals.

Often one has to reset the head to its initial position 0. There are several methods to achieve this. A universal one assumes that there is a unique initial inscription `start` on the tape which indicates the 0-position:

```
Reset_head:
A:      moveleft
        if head reads ‘start’ then STOP otherwise goto A
```

It will be convenient to work with several tapes side-by-side instead of just one. This corresponds to the idea of program *variables* whose values are checked and manipulated. One can simulate an n-tape machine on a 1-tape machine. The contents $(T^i_\xi \mid \xi \in \mathrm{Ord})$ of the ith tape are successively written into the cells of tape T indexed by ordinals $2n\xi + 2i$:

$$T_{2n\xi+2i} = T^i_\xi.$$

The head position H^i on the i-th tape is simulated by writing 1's into an initial segment of length H^i of cells with indices of the form $2n\xi + 2i + 1$:

$$T_{2n\xi+2i+1} = \begin{cases} 1, & \text{if } \xi < H^i; \\ 0, & \text{else.} \end{cases}$$

So two tapes with contents $a_0a_1a_2a_3a_4\ldots$ and $b_0b_1b_2b_3b_4\ldots$ and head positions 3 and 1 respectively are coded as

$$T = a_0 1 b_0 1 a_1 1 b_1 0 a_2 1 b_2 0 a_3 0 b_3 0 a_4 0 b_4 0 \ldots\ldots.$$

We describe operations of machines with several tapes by commands like `move-right2` or `print3 = ' '`, where the number of the active tape is adjoined to the right. There are canonical but tedious translations from programs for n-tape machines into corresponding programs for 1-tape machines. A manipulation of the i-th tape amounts to first finding the head marker at ordinals of form $2n\xi + 2i + 1$; moving left by one cell one obtains the corresponding cell content for possible modification; the subsequent head movement is simulated by moving right again, writing a 0, moving $2n$ cells to the right or left, and printing a 1; if a left-movement goes across a limit ordinal, then a "1" has to be printed into cell $2i + 1$.

The subsequent algorithms will be presented as multiple tape algorithms. One can assume that one or more of the tapes serve as standard TURING tapes on which ordinary TURING recursive functions are computed. Since the usual syntactical operations for a language of set theory are intuitively computable we can assume by the CHURCH-TURING thesis that these operations are ordinal computed on some of the ordinal tapes. This will be used in the ordinal computation of the constructible model L.

Basic operations on ordinals are ordinal computable. Let the ordinals α and β be given on tapes 0 and 1 as their characteristic functions $\chi_{\{\alpha\}}$ and $\chi_{\{\beta\}}$. The following algorithm *compares* the ordinals and indicates the result of the comparison by its "stopping state":

```
Ordinal_Comparison:
       Reset_Head0
       Reset_Head1
A:     if head0='1' and head1='0' then STOP ('alpha>beta')
       if head0='1' and head1='1' then STOP ('alpha=beta')
       if head0='0' and head1='1' then STOP ('alpha<beta')
       moveright0
       moveright1
       goto A
```

Obviously there are ordinal algorithms to *reset* a register containing an ordinal to 0, or to *copy* one ordinal register to another one. The ordinal *sum* $\alpha + \beta$ and *product* $\alpha \cdot \beta$ are computable as follows:

```
Ordinal_Addition:
        Reset_Head0
        Reset_Head1
        Reset_Head2
A:      if head0 = ‘1’ then goto B
        moveright0
        moveright2
        goto A
B:      if head1 = ‘1’ then goto C
        moveright1
        moveright2
        goto B
C:      print2 = ‘1’
        STOP
```

```
Ordinal_Multiplication:
        Reset_Head0
        Reset_Head1
        Reset_Head2
A:      if head1 = ‘1’ then goto C
        if head0 = ‘1’ then goto B
        moveright0
        moveright2
        goto A
B:      Reset_Head0
        moveright1
        goto A
C:      print2 = ‘1’
        STOP
```

The class $\mathrm{Ord}^{<\omega} = \{ s \mid \exists k < \omega \; s : k \to \mathrm{Ord} \}$ of finite sequences of ordinals will be of particular interest for relating ordinal computability to the iterated definability of GÖDEL's constructible universe. We code a sequence $(\alpha_0, \dots, \alpha_{k-1}) : k \to \mathrm{Ord}$ by a tape which starts with an initial symbol "(", followed by k intervals of 0's of lengths $\alpha_0, \dots \alpha_{k-1}$ respectively, which are separated by a separation symbol "," and then a closing ")". So $(1, \omega, \omega + 2)$ is coded as

$$(0, 00 \dots, 00 \dots 00)$$

If the sequence is given on tape 0 and a natural number n on tape 1 then the nth element of the sequence can be output on tape 2 by the following algorithm:

```
Extract:
        Reset_Head0
        Reset_Head1
        Reset_Head2
A:      if head1 = '1' then goto C
        moveright1
B:      if head0 = ',' then goto A
        if head0 = ')' then STOP (no output)
        moveright0
        goto B
C:      moveright0
D:      if head0 = ',' then goto E
        if head0 = ')' then goto E
        moveright0
        moveright2
        goto D
E:      print2 = '1'
        STOP
```

Another important operation on sequences is the *replacement* of the nth element of a sequence s of ordinals by a given ordinal α; if the given sequence is shorter than $n+1$, it is padded by 0's up to length $n+1$. Formally this operation on sequences is defined as $s \mapsto s\frac{\alpha}{n}$ where $\mathrm{dom}(s\frac{\alpha}{n}) = \mathrm{dom}(s) \cup (n+1)$ and

$$s\frac{\alpha}{n}(i) = \begin{cases} s(i), & \text{if } i \in \mathrm{dom}(s) \setminus \{n\}; \\ \alpha, & \text{if } i = n; \\ 0, & \text{else.} \end{cases}$$

Let the original sequence be given on tape 0, the natural number n on tape 1, and the ordinal α on tape 2. The modified sequence $s\frac{\alpha}{n}$ can be output on tape 3 by the following algorithm:

```
Replace:
        print3 = '('
A:      moveright0
        if read1 = '1' then goto C
        if read0 = ',' then goto B
        if read0 = ')' then goto H
        moveright3
        goto A
B:      print3 = ','
        moveright1
        goto A
C:      if read2 = '1' then goto D
        moveright2
```

```
        moveright3
        goto C
D:      moveright0
        if read0 = ‘0’ then goto D
E:      if read0 = ‘,’ then goto F
        if read0 = ‘)’ then goto G
        moveright3
        moveright0
        goto E
F:      print3 = ‘,’
        moveright3
        moveright0
        goto E
G:      print3 = ‘)’
        STOP
H:      print3 = ‘,’
        moveright1
        if read1 = ‘0’ then goto H
I:      if read2 =‘1’ then goto J
        moveright2
        moveright3
        goto I
J:      print3 = ‘)’
        STOP
```

With the subroutine mechanism known from ordinary programming the basic algorithms can be combined into complex algorithms for comparing and manipulating ordinal sequences. We can, e.g., carry out a syntactic manipulation on a standard Turing tape which outputs requests for checking or manipulating elements of ordinal sequences. According to the requests the appropriate elements can be extracted and subjected to some algorithms whose results can be substituted into the original sequences.

§4. Enumerating finite sequences of ordinals.

For X a class let $[X]^{<\omega} = \{ z \subseteq X \mid z \text{ is finite} \}$ and $X^{<\omega} = \{ s \mid \exists k < \omega \; s : k \to X \}$ be the class of all finite *subsets* of X and of all finite *sequences* from X respectively.

Finite sequences of ordinals are finite sets of ordered pairs:

$$\mathrm{Ord}^{<\omega} \subseteq [\omega \times \mathrm{Ord}]^{<\omega}.$$

Well-order $\omega \times \mathrm{Ord}$ by

$$(m, \alpha) \prec (n, \beta) \quad \text{if and only if} \quad \alpha < \beta \text{ or } (\alpha = \beta \wedge m < n).$$

Define a canonical well-order $([\omega \times \mathrm{Ord}]^{<\omega}, \prec^*)$ by largest difference:

$$s \prec^* s' \quad \text{if and only if} \quad \exists x \in s' \setminus s \; \{ y \in s \mid y \succ x \} = \{ y \in s' \mid y \succ x \}.$$

One can show inductively that $([Y]^{<\omega}, \prec^*)$ is a well-order on initial segments Y of $(\omega \times \mathrm{Ord}, \prec)$. So $\prec^*$ well-orders $[\omega \times \mathrm{Ord}]^{<\omega}$ and hence $\mathrm{Ord}^{<\omega}$. We note an important *substitution property* of the well-order:

LEMMA 4.1. *If* $s, t \in \mathrm{Ord}^{<\omega}, m \in \mathrm{dom}(s), \mathrm{dom}(s) \subseteq \mathrm{dom}(t), s \restriction (\mathrm{dom}(s) \setminus \{m\}) = t \restriction (\mathrm{dom}(s) \setminus \{m\}), t(m) < s(m), \forall i \in \mathrm{dom}(t) \setminus \mathrm{dom}(s)\ t(i) < s(m)$ *then*

$$t \prec^* s.$$

So replacing an arbitrary ordinal $s(m)$ of s by possibly many smaller ordinals leads to a descent in $\prec^*$. The substitution property will correspond to the substitution of a bounded variable below some bound by terms with parameters smaller than that bound. This will lead to a recursive definition of bounded truth in L along the $\prec^*$-relation.

We define an enumeration $S\colon \mathrm{Ord} \to \mathrm{Ord}^{<\omega}$ (with repetitions) of $\mathrm{Ord}^{<\omega}$ which is compatible with $\prec^*$ and which can be computed by an ordinal machine. The idea of the construction is to recursively apply the replacement operation $s\frac{\alpha}{n}$ to sequences s which have been enumerated before.

For $(m, \alpha) \in \omega \times \mathrm{Ord}$ define functions $S_{m\alpha}\colon \theta_{m\alpha} \to \mathrm{Ord}^{<\omega}$ such that for $(m, \alpha) \prec (n, \beta)$, $S_{m\alpha}$ is an initial segment of $S_{n\beta}$. Set $S_{00}\colon 1 \to \mathrm{Ord}^{<\omega}$, $S_{00}(0) = \emptyset$. For $\beta > 0$ set

$$S_{0\beta} = \bigcup_{(m,\alpha) \prec (0,\beta)} S_{m\alpha}.$$

Assume that $S_{m\alpha}\colon \theta_{m\alpha} \to \mathrm{Ord}^{<\omega}$ is defined. Then define $S_{m+1,\alpha}\colon \theta_{m\alpha} \cdot 2 \to \mathrm{Ord}^{<\omega}$ by: $S_{m+1,\alpha} \restriction \theta_{m\alpha} = S_{m\alpha}$; for $\xi < \theta_{m\alpha}$ let

$$S_{m+1,\alpha}(\theta_{m\alpha} + \xi) = S_{m\alpha}(\xi)\frac{\alpha}{m}.$$

Finally set

$$S = \bigcup_{(m,\alpha) \in \omega \times \mathrm{Ord}} S_{m\alpha}.$$

LEMMA 4.2. (a) $S\colon \mathrm{Ord} \to \mathrm{Ord}^{<\omega}$ *is a surjection.*
(b) *If* $\xi < \zeta$ *then* $S(\xi) = S(\zeta)$ *or* $S(\xi) \prec^* S(\zeta)$.

PROOF. (a) We show by induction on α that $S_{0\alpha}\colon \theta_{0\alpha} \to \alpha^{<\omega}$ is a surjection. The initial case $\alpha = 0$ and the limit step are easy. Consider $\alpha = \beta + 1$ and some $s \in \alpha^{<\omega}$, $s\colon k \to \alpha$. Let $\bar{s}\colon k \to \alpha$ be the following restriction of s to β:

$$\bar{s}(i) = \begin{cases} s(i), & \text{if } s(i) < \beta; \\ 0, & \text{if } s(i) = \beta. \end{cases}$$

By the inductive assumption there is $\xi < \theta_{0\beta}$ such that $S_{0\beta}(\xi) = \bar{s}$. Then

$$S_{0\alpha}\left(\theta_{0\beta} \cdot \left(\sum_{i<k, s(i)=\beta} 2^i\right) + \xi\right) = s.$$

(b) follows from the substitution property. ⊣

The enumeration S of $\mathrm{Ord}^{<\omega}$ is *ordinal computable* using coding methods from the previous paragraph. We indicate a program which writes the values of S consecutively on a tape:

$$S(0)S(1)S(2)\ldots S(\omega)S(\omega+1)\ldots S(\omega+\omega)\ldots S(\alpha)\ldots\ldots$$

where each $S(\alpha)$ is of the form

$$(0\ldots 0, 0\ldots 0, \ldots\ldots, 0\ldots 0)$$

The algorithm is based on the `Replace`-algorithm from the previous section:

```
counter0 = 0
counter1 = 0
position_of_writing_head = 0
write the empty sequence ‘()’
while true
 while counter1 < ω
  mark = position_of_writing_head
  position_of_reading_head = 0
  while position_of_reading_head < mark
   read sequence
   Replace element at position counter1 by counter0
   write modified sequence at mark
  endwhile
  counter1 = counter1 + 1
 endwhile
 counter1 = 0
 counter0 = counter0 + 1
endwhile
```

The procedure will eventually be extended as to write a bounded truth function for the constructible hierarchy.

§5. The constructible hierarchy. Kurt Gödel [3] defined the inner model L of *constructible sets* as the union of a hierarchy of levels L_α:

$$L = \bigcup_{\alpha \in \mathrm{Ord}} L_\alpha$$

where the hierarchy is defined by: $L_0 = \emptyset$, $L_\delta = \bigcup_{\alpha<\delta} L_\alpha$ for limit ordinals δ, and $L_{\alpha+1}$ =the set of all sets which are first-order definable in the structure

$(L_\alpha, \in)$. The standard reference to the theory of the model L is the book [2] by KEITH DEVLIN.

An element of L is definable over some L_α from parameters which are themselves definable over some L_β, $\beta < \alpha$ in some other parameters and so forth. We therefore introduce a language with definable terms, which in turn may involve definable terms, etc.

Consider a language with symbols $(,), \{, \}, |, \in, =, \wedge, \neg, \forall, \exists$ and variables $v_0, v_1, \ldots$. We define (**bounded**) **formulas** and (**bounded**) **terms** by a common recursion on the lenghts of words formed from these symbols:

- the variables $v_0, v_1, \ldots$ are terms;
- if s and t are terms then $s = t$ and $s \in t$ are formulas;
- if φ and ψ are formulas then $\neg\varphi$, $(\varphi \wedge \psi)$, $\forall v_i \in v_j\ \varphi$ and $\exists v_i \in v_j\ \varphi$ are formulas;
- if φ is a formula then $\{\, v_i \in v_j \mid \varphi \,\}$ is a term.

For terms and formulas of this language define **free** and **bound variables**:

- $\text{free}(v_i) = \{v_i\}, \text{bound}(v_i) = \emptyset$;
- $\text{free}(s = t) = \text{free}(s \in t) = \text{free}(s) \cup \text{free}(t)$;
- $\text{bound}(s = t) = \text{bound}(s \in t) = \text{bound}(s) \cup \text{bound}(t)$;
- $\text{free}(\neg\varphi) = \text{free}(\varphi), \text{bound}(\neg\varphi) = \text{bound}(\varphi)$;
- $\text{free}((\varphi \wedge \psi)) = \text{free}(\varphi) \cup \text{free}(\psi), \text{bound}((\varphi \wedge \psi)) = \text{bound}(\varphi) \cup \text{bound}(\psi)$;
- $\text{free}(\forall v_i \in v_j\ \varphi) = \text{free}(\exists v_i \in v_j\ \varphi) = \text{free}(\{\, v_i \in v_j \mid \varphi \,\}) = (\text{free}(\varphi) \cup \{v_j\}) \setminus \{v_i\}$;
- $\text{bound}(\forall v_i \in v_j\ \varphi) = \text{bound}(\exists v_i \in v_j\ \varphi) = \text{bound}(\{\, v_i \in v_j \mid \varphi \,\}) = \text{bound}(\varphi) \cup \{v_i\}$.

For technical reasons we will be interested in terms and formulas in which

- no bound variable occurs free,
- every free variable occurs exactly once.

Such terms and formulas are called **tidy**; with tidy formulas one avoids having to deal with the interpretation of one free variable at different positions within a formula.

In recursive truth definitions one reduces the truth of formulas to the truth of *simpler* formulas. The **term complexity** $\text{tc}(t)$ and $\text{tc}(\varphi)$ of terms and formulas is defined recursively:

- $\text{tc}(v_i) = 0$;
- $\text{tc}(s = t) = \text{tc}(s \in t) = \max(\text{tc}(s), \text{tc}(t))$;
- $\text{tc}(\neg\varphi) = \text{tc}(\forall v_i \in v_j\ \varphi) = \text{tc}(\exists v_i \in v_j\ \varphi) = \text{tc}(\varphi)$;
- $\text{tc}(\varphi \wedge \psi) = \max(\text{tc}(\varphi), \text{tc}(\psi))$;
- $\text{tc}(\{\, v_i \in v_j \mid \varphi \,\}) = \text{tc}(\varphi) + 1$.

We can define a pre-wellordering $<_{\text{Form}}$ of the set of all bounded formulas by

$$\varphi <_{\text{Form}} \psi \quad \text{if and only if}$$
$$\text{tc}(\varphi) < \text{tc}(\psi) \text{ or } (\text{tc}(\varphi) = \text{tc}(\psi) \wedge \text{length}(\varphi) < \text{length}(\psi)).$$

Obviously the syntactical notions and operations of this language are TURING computable and therefore ordinal computable. Also there is an ordinal computable enumeration of all formulas which is compatible with $<_{\text{Form}}$.

An *assignment* for a term t or formula φ is a finite sequence $a\colon k \to V$ so that for every free variable v_i of t or φ we have $i < k$; $a(i)$ will be the *interpretation* of v_i. The *value* of t or the *truth value* of φ is determined by the assignment a. We write $t[a]$ and $\varphi[a]$ for the values of t und φ under the assignment a.

Concerning the constructible hierarchy L, it is shown by an easy induction on α that every element of L_α is the interpretation $t[(L_{\alpha_0}, \ldots, L_{\alpha_{k-1}})]$ of some *tidy* term t with an assignment $(L_{\alpha_0}, \ldots, L_{\alpha_{k-1}})$ whose values are constructible levels L_{α_i} with $\alpha_0, \ldots, \alpha_{k-1} < \alpha$. This will allow to reduce bounded quantifications $\forall v \in L_\alpha$ or $\exists v \in L_\alpha$ to the substitution of terms of lesser complexity. Moreover, the truth of (bounded) formulas in L is captured by *tidy* bounded formulas of the form $\varphi[(L_{\alpha_0}, \ldots, L_{\alpha_{k-1}})]$.

We shall code an assignment of the form $(L_{\alpha_0}, L_{\alpha_1}, \ldots, L_{\alpha_{k-1}})$ by its sequence of ordinal indices, i.e., we write

$$t[(\alpha_0, \alpha_1, \ldots, \alpha_{k-1})] \quad \text{or} \quad \varphi[(\alpha_0, \alpha_1, \ldots, \alpha_{k-1})]$$

instead of

$$t[(L_{\alpha_0}, L_{\alpha_1}, \ldots, L_{\alpha_{k-1}})] \quad \text{or} \quad \varphi[(L_{\alpha_0}, L_{\alpha_1}, \ldots, L_{\alpha_{k-1}})].$$

The relevant assignments are thus elements of $\text{Ord}^{<\omega}$ and can be handled by the programs of the previous section. Since the bounded language is recursive we can modify the enumeration program so that all assigned tidy formulas $\varphi[(\alpha_0, \alpha_1, \ldots, \alpha_{k-1})]$ occur in the enumeration: for a fixed assigment $a = (\alpha_0, \alpha_1, \ldots, \alpha_{k-1})$ list the pairs (a, φ) where φ is a tidy formula with $\text{free}(\varphi) \subseteq k$ in an order compatible with $<_{\text{Form}}$. The following is a straightforward extension of the enumeration program of the previous section:

```
counter0 = 0
counter1 = 0
position_of_writing_head = 0
write the empty sequence ‘()’
while true
 while counter1 < ω
  mark = position_of_writing_head
  position_of_reading_head = 0
```

```
  while position_of_reading_head < mark
   read sequence
   Replace element at position counter1 by counter0
   form an enumeration of the appropriate tidy formulas
        which is compatible with the pre-wellorder <Form
   for all enumerated formulas
    write the modified sequence and the formula
   endfor
  endwhile
  counter1 = counter1 + 1
 endwhile
 counter1 = 0
 counter0 = counter0 + 1
endwhile
```

§6. A bounded truth function for L. We define a bounded truth function W for the constructible hierarchy on the class

$$A = \{ (a, \varphi) \mid a \in \mathrm{Ord}^{<\omega}, \varphi \text{ is a tidy bounded formula}, \\ \mathrm{free}(\varphi) \subseteq \mathrm{dom}(a) \}$$

of all "tidy pairs" of assignments and formulas. Define the **bounded constructible truth function** $W : A \to 2$ by

$$W(a, \varphi) = 1 \quad \text{if and only if} \quad \varphi[a].$$

The function W has a *recursive* definition along the enumeration of A given by the enumeration algorithm from the preceding section. We explain the principal idea of the recursion with a bounded quantification like $\exists v_i \in v_j\ \varphi[a]$. If $a(j) = \alpha$ then the assigned formula is satisfied if and only if there is a witness for φ in L_α. By the recursive definition of L_α such a witness must be the interpretation $t[b]$ of a term where $b(l) < \alpha$ for every free variable v_l of t. If one chooses t such that it has no variable in common with φ the assignment b can be taken to further satisfy $b(j) < a(j) = \alpha$. The substitution property of Lemma 4.1 leads to the evaluation of $\varphi\frac{t}{v_i}[b]$ for some $b \prec^* a$ which is the basis for the subsequent recursion.

As we want to work with *tidy* formulas a technical problem has to be solved. The variable v_i might occur in φ in several places which renders the straightforward substitution $\varphi\frac{t}{v_i}$ "untidy". We "tidy up" $\varphi\frac{t}{v_i}$ by renaming variables. The assignment of the variables of t has to be modified accordingly.

So consider a formula φ, a variable v_i, a term t, and an assignment a with $\{ i \mid v_i \in \mathrm{free}(\varphi) \} \cup \{v_j\} \subseteq \mathrm{dom}(a)$, where v_j is a further variable thought to be a bound for t as in $\exists v_i \in v_j\ \varphi[a]$. Also assume that φ and t have no common variable and do not contain v_j. Define the **tidy substitution** $(\varphi\frac{t}{v_i})^{\mathrm{tidy}}$ of v_i by t into φ as follows. If $v_i \notin \mathrm{free}(\varphi)$ then let $(\varphi\frac{t}{v_i})^{\mathrm{tidy}} = \varphi$. Otherwise ensure that v_i is not a bound variable of φ by possibly renaming bound variables. Let φ' be the renamed formula. Then rename all occurances of v_i in φ' by *pairwise different new* variables $w_0, \dots, w_{k-1}$, say. Obtain terms $t_0, \dots, t_{k-1}$ from the given term t by renaming all variables with *new* variables so that for $i \neq j$ the terms t_i and t_j do not have common variables; call $t_0, \dots, t_{k-1}$ **copies** of t. Now set

$$\left(\varphi \frac{t}{v_i}\right)^{\text{tidy}} = \varphi' \frac{t_0 \dots t_{k-1}}{w_0 \dots w_{k-1}}.$$

The assignment a has to be extended to an assignment $b \prec^* a$ in line with the various renaming operations.

We define that the assignment b **adequately extends** a **for the tidy substitution** $\left(\varphi \frac{t}{v_i}\right)^{\text{tidy}}$ **bounded by** v_j if

(a) $\forall l \in \text{dom}(a) \setminus \{j\}\ b(l) = a(l)$ and $b(j) < a(j)$;
(b) $\forall l \in \text{dom}(b) \setminus \text{dom}(a)\ b(l) < a(j)$;
(c) if v_l is a variable of t and $v_{l'}$ and $v_{l''}$ are the renamings of v_l in the copies t_i and t_j, resp., then $b(l') = b(l'')$.

Note that b satisfies $b \prec^* a$ by the substitution property Lemma 4.1.

With these preparations we can now carry out a recursive definition of the bounded constructible truth function:

- $W(a, \neg\varphi) = 1$ if and only if $W(a, \varphi) = 0$;
- $W(a, (\varphi \wedge \psi)) = 1$ if and only if $W(a, \varphi) = 1$ *and* $W(a, \psi) = 1$;
- $W(a, \forall v_i \in v_j\ \varphi) = 1$ if and only if *for all* terms t and all assignments b which are adequate for the tidy substitution $(\varphi \frac{t}{v_i})^{\text{tidy}}$ bounded by v_j holds $W(b, (\varphi \frac{t}{v_i})^{\text{tidy}}) = 1$;
- $W(a, \exists v_i \in v_j\ \varphi) = 1$ if and only if *there is* a term t and an assignment b which is adequate for the tidy substitution $(\varphi \frac{t}{v_i})^{\text{tidy}}$ bounded by v_j so that $W(b, (\varphi \frac{t}{v_i})^{\text{tidy}}) = 1$;
- $W(a, v_i \in v_j) = 1$ if and only if $a(i) < a(j)$;
- $W(a, v_i \in \{ v_j \in v_k \mid \varphi \}) = 1$ if and only if $W(a, \exists v_j \in v_k\ (v_i = v_j \wedge \varphi)) = 1$;
- $W(a, \{ v_i \in v_j \mid \varphi \} \in v_k) = 1$ iff *there is* a term t and an assignment b which is adequate for the tidy substitution $\left((\{ v_i \in v_j \mid \varphi \} = v_l) \frac{t}{v_l}\right)^{\text{tidy}}$ bounded by v_k so that $W\big(b, \left((\{ v_i \in v_j \mid \varphi \} = v_l) \frac{t}{v_l}\right)^{\text{tidy}}\big) = 1$;
- $W(a, \{ v_i \in v_j \mid \varphi \} \in \{ v_m \in v_n \mid \psi \}) = 1$ if and only if *there is* a term t and an assignment b which is adequate for the tidy substitution $\left((\{ v_i \in v_j \mid \varphi \} = v_m \wedge \psi) \frac{t}{v_m}\right)^{\text{tidy}}$ bounded by v_n so that $W\big(b, \left((\{ v_i \in v_j \mid \varphi \} = v_m \wedge \psi) \frac{t}{v_m}\right)^{\text{tidy}}\big) = 1$;
- $W(a, v_i = v_j) = 1$ if and only if $a(i) = a(j)$;

– $W(a, v_i = \{ v_j \in v_k \mid \varphi \}) = 1$ if and only if $W(a, \forall v_l \in v_i\, \exists v_j \in v_k\, (\varphi \wedge v_l = v_j)) = 1$ and $W(a, \forall v_j \in v_k\, (\varphi \to \exists v_l \in v_i\, v_l = v_j)) = 1$;
– $W(a, \{ v_i \in v_j \mid \varphi \} = \{ v_m \in v_n \mid \psi \}) = 1$ if and only if $W(a, (\forall v_i \in v_j\, (\varphi \to \exists v_m \in v_n\, (v_i = v_m \wedge \psi))) = 1$ and $W(a, \forall v_m \in v_n\, (\psi \to \exists v_i \in v_j\, (v_m = v_i \wedge \varphi)))) = 1$.

In all clauses the determination of $W(a, \varphi)$ is reduced to values $W(b, \psi)$. where the relevant arguments (b, ψ) are of lesser complexity than (a, φ): either the assignments satisfy $b \prec^* a$ or we have that $a = b$ and $\psi <_{\mathrm{Form}} \varphi$. Therefore W has a recursive definition along the enumeration given by the algorithm of the preceding section.

The programming techniques introduced above allow to incorporate the recursive definition of W into the enumeration algorithm for the class A of admissible pairs. This leads to our main results:

LEMMA 6.1. *The bounded truth function W for the constructible universe is ordinal computable.*

THEOREM 6.2. *A set x of ordinals is ordinal computable from a finite set of ordinal parameters if and only if it is an element of the constructible universe L.*

PROOF. Let $x \subseteq \mathrm{Ord}$ be ordinal computable by the program P from the finite set $\{\alpha_0, \dots, \alpha_{k-1}\}$ of ordinal parameters: $P\colon \chi_{\{\alpha_0,\dots,\alpha_{k-1}\}} \mapsto \chi_x$. By Lemma 2.6 c) the same computation can be carried out inside the inner model L:

$$(L, \in) \models P\colon \chi_{\{\alpha_0,\dots,\alpha_{k-1}\}} \mapsto \chi_x.$$

Hence $\chi_x \in L$ and $x \in L$.

Conversely consider $x \in L$. Choose a tidy term t and an assignment $(\alpha_0, \dots, \alpha_{k-1}) \in \mathrm{Ord}^{<\omega}$ such that $x = t[(\alpha_0, \dots, \alpha_{k-1})]$. An ordinal β can be represented as

$$\begin{aligned} \beta &= \{ \alpha \in L_\beta \mid \alpha \text{ is an ordinal} \} \\ &= \{ v_{k+1} \in v_k \mid v_{k+1} \text{ is an ordinal} \}[(\alpha_0, \dots, \alpha_{k-1}, \beta)]. \end{aligned}$$

Thus

$$\begin{aligned} &\chi_x(\beta) = 1 \\ &\quad \leftrightarrow \beta \in t[(\alpha_0, \dots, \alpha_{k-1})] \\ &\quad \leftrightarrow (\{ v_{k+1} \in v_k \mid v_{k+1} \text{ is an ordinal} \} \in t)[(\alpha_0, \dots, \alpha_{k-1}, \beta)] \\ &\quad \leftrightarrow W((\alpha_0, \dots, \alpha_{k-1}, \beta), \{ v_{k+1} \in v_k \mid v_{k+1} \text{ is an ordinal} \} \in t) = 1. \end{aligned}$$

Using the enumeration algorithm for the truth function W one can turn this equivalence into an ordinal algorithm which sends $\chi_{\{\alpha_0,\dots,\alpha_{k-1}\}}$ to χ_x. Hence x is ordinal computable from the parameters $\alpha_0, \dots, \alpha_{k-1} \in \mathrm{Ord}$. ⊣

§7. The generalized continuum hypothesis in L. Ordinal computability allows to reprove some basic facts about the constructible universe L. The analogue of the axiom of constructibility, $V = L$, is the statement that *every* set of of ordinals is ordinal computable from a finite set of ordinals.

THEOREM 7.1. *The constructible model $(L, \in)$ satisfies that every set of ordinals is ordinal computable from a finite set of ordinals.*

PROOF. Let $x \in L$, $x \subseteq \mathrm{Ord}$. By Theorem 6.2, take a program P and a finite set $\{\alpha_0, \ldots, \alpha_{k-1}\}$ of ordinal parameters such that $P\colon \chi_{\{\alpha_0,\ldots,\alpha_{k-1}\}} \mapsto \chi_x$. By Lemma 2.6 c) the same computation can be carried out inside the inner model L:

$$(L, \in) \models P\colon \chi_{\{\alpha_0,\ldots,\alpha_{k-1}\}} \mapsto \chi_x.$$

So in L, x is ordinal computable from the set $\{\alpha_0, \ldots, \alpha_{k-1}\}$. ⊣

The following theorem is proved by a condensation argument for ordinal computations which is a simple analogue of the usual condensation arguments for the constructible hierarchy.

THEOREM 7.2. *Assume that every set of ordinals is ordinal computable from a finite set of ordinals. Then*:

(a) *Let $\kappa \geqslant \omega$ be an infinite ordinal and $x \subseteq \kappa$. Then there are ordinals $\alpha_0, \ldots, \alpha_{k-1} < \kappa^+$ such that x is ordinal computable from the set $\{\alpha_0, \ldots, \alpha_{k-1}\}$.*
(b) *Let $\kappa \geqslant \omega$ be infinite. Then* $\mathrm{card}(\mathcal{P}(\kappa)) = \kappa^+$.
(c) *The generalized continuum hypothesis* GCH *holds.*

PROOF. (a) Take a program P and a finite set $\{\alpha'_0, \ldots, \alpha'_{k-1}\}$ of ordinal parameters such that $P\colon \chi_{\{\alpha'_0,\ldots,\alpha'_{k-1}\}} \mapsto \chi_x$; let θ be the length of this ordinal computation. Take a transitive ZF^--model $(M, \in)$ such that $\alpha'_0, \ldots, \alpha'_{k-1}, \theta, \kappa, x \in M$. By Lemma 2.6 a), $(M, \in)$ also satisfies that $P\colon \chi_{\{\alpha'_0,\ldots,\alpha'_{k-1}\}} \mapsto \chi_x$. The downward LÖWENHEIM-SKOLEM theorem and the MOSTOWSKI isomorphism theorem yield an elementary embedding

$$\pi\colon (\bar{M}, \in) \to (M, \in)$$

such that $\bar{M}$ is transitive, $\mathrm{card}(\bar{M}) = \kappa$ and $\{\alpha'_0, \ldots, \alpha'_{k-1}, \theta, \kappa, x\} \cup \kappa \subseteq \pi''\bar{M}$. Let $\pi(\alpha_0) = \alpha'_0, \ldots, \pi(\alpha_{k-1}) = \alpha'_{k-1}$. Then $\alpha_0, \ldots, \alpha_{k-1} < \kappa^+$ since $\mathrm{card}(\bar{M}) < \kappa^+$. Observe that $\pi(x) = x$. Since π is elementary $(\bar{M}, \in)$ satisfies that $P\colon \chi_{\{\alpha_0,\ldots,\alpha_{k-1}\}} \mapsto \chi_x$. By Lemma 2.6 d), $P\colon \chi_{\{\alpha_0,\ldots,\alpha_{k-1}\}} \mapsto \chi_x$ in V. Thus x is ordinal computable from the set $\{\alpha_0, \ldots, \alpha_{k-1}\}$ as required.

(b) follows from (a) since there are countably many programs and κ^+ many finite sets of ordinals $< \kappa^+$.

(c) is immediate from (b). ⊣

These two theorems immediately imply GÖDEL's result:

THEOREM 7.3. $(L, \in) \models \mathrm{GCH}$.

Other condensation arguments like the proof of the combinatorial principle $\diamondsuit$ in L can also be translated into the setting of ordinal computability in a straightforward way. It remains to be seen whether arguments involving JENSEN's fine structure theory of the constructible hierarchy [5] can be carried out with ordinal computability. One would hope that the simple concept of ordinal computation allows clear proofs of principles like $\square$ and morasses without definability complications.

REFERENCES

[1] J. RICHARD BÜCHI, *Weak second-order arithmetic and finite automata*, ***Zeitschrift für Mathematische Logik und die Grundlagen der Mathematik***, vol. 6 (1960), pp. 66–92.

[2] KEITH DEVLIN, ***Constructibility***, Perspectives in Mathematical Logic, Springer-Verlag, Berlin, 1984.

[3] KURT GÖDEL, ***The consistency of the continuum hypothesis***, Annals of Mathematics Studies, vol. 3, Princeton University Press, Princeton, 1940.

[4] JOEL DAVID HAMKINS and ANDY LEWIS, *Infinite time Turing machines*, ***The Journal of Symbolic Logic***, vol. 65 (2000), no. 2, pp. 567–604.

[5] RONALD R. JENSEN, *The fine structure of the constructible hierarchy*, ***Annals of Mathematical Logic***, vol. 4 (1972), pp. 229–308.

[6] THOMAS LLOYD RICHARDSON, ***Silver machine approach to the constructible universe***, Ph.D. thesis, University of California, Berkeley, 1979.

[7] GERALD E. SACKS, ***Higher recursion theory***, Perspectives in Mathematical Logic, Springer-Verlag, Berlin, Heidelberg, 1990.

[8] JACK H. SILVER, *How to eliminate the fine structure from the work of Jensen*, handwritten manuscript, 197?

MATHEMATISCHES INSTITUT
RHEINISCHE FRIEDRICH-WILHELMS-UNIVERSITÄT
BERINGSTRAßE 1
D-53115 BONN, GERMANY
E-mail: koepke@math.uni-bonn.de