Then, for $n = 1, 2, 3, \ldots$:

$$\tau_{n+1}(x_1, \ldots, x_n, x_{n+1}, c, u, v) = \tau_1(x_{n+1}, c, (\tau_n(x_1, \ldots, x_{n-1}, x_n', c, u, v))_0, v).$$

When $x_1, \ldots, x_n$ is scanned in standard position with state q_1, and the tape is blank elsewhere, the Gödel number of the situation is $\tau_n(x_1, \ldots, x_n, 1, 1, 1)$. When $x_1, \ldots, x_n, x$ is scanned in standard position with state q_0, the Gödel number of the situation is $\tau_{n+1}(x_1, \ldots, x_n, x, 0, u, v)$ for some u and v; and conversely.

Now, if φ is the partial function of n variables computed by the given machine $\mathfrak{M}$, and $x_1, \ldots, x_n$ is a given n-tuple, then $\varphi(x_1, \ldots, x_n)$ is defined, if and only if there exists a quadruple (z, x, u, v) of numbers such that $\theta(\tau_n(x_1, \ldots, x_n, 1, 1, 1), z) = \tau_{n+1}(x_1, \ldots, x_n, x, 0, u, v)$, in which case x is the value of $\varphi(x_1, \ldots, x_n)$. Accordingly,

$$\varphi(x_1, \ldots, x_n) \simeq (\mu t[\theta(\tau_n(x_1, \ldots, x_n, 1, 1, 1), (t)_0) = \tau_{n+1}(x_1, \ldots, x_n, (t)_1, 0, (t)_2, (t)_3)])_1.$$

Therefore, by Theorem XVIII § 63, φ is partial recursive (or if φ is completely defined, by Theorem III § 57, φ is general recursive).

PROOF FOR $l > 0$. Say e.g. there is one assumed function ψ of one variable (i.e. $l = m_1 = 1$). Now, for each c for which the table entry corresponding to q_c is of the form $1q_d$, we replace "$\rho_{a,c}((w)_0, (w)_3)$ if $(w)_1 = a$ & $(w)_2 = c$ $(a = 0, \ldots, j)$" in the definition of $\rho(w)$ by "$\rho_c(w)$ if $Q_c(w)$", where Q_c is the primitive recursive predicate and ρ_c the function primitive recursive in ψ defined thus:

$$\begin{aligned} Q_c(w) &\equiv (Ey)_{y<w}(Eu)_{u<w}(Ev)_{v<w}[w = \tau_1(y, c, u, v)], \\ \rho_c(w) &= \tau_2(Y, \psi(Y), d, U, V) \text{ where} \\ Y &= \mu y_{y<w}(Eu)_{u<w}(Ev)_{v<w}[w = \tau_1(y, c, u, v)], \\ U &= \mu u_{u<w}(Ey)_{y<w}(Ev)_{v<w}[w = \tau_1(y, c, u, v)], \text{ etc.} \end{aligned}$$

THEOREM XXX (= Theorems XXVIII + XXIX). *The following classes of partial functions are coextensive, i.e. have the same members:* (a) *the partial recursive functions,* (b) *the computable functions,* (c) *the* 1/1 *computable functions. Similarly with l completely defined assumed functions* Ψ.

§ 70. Turing's thesis.

Turing's thesis that every function which would naturally be regarded as computable is computable under his definition, i.e. by one of his machines, is equivalent to Church's thesis by Theorem XXX. We shall now examine the part of the evidence for it

which pertains to the machine concept, i.e. what we listed as (C) in § 62. What we must do is to convince ourselves that any acts a human computer could carry out are analyzable into successions of atomic acts of some Turing machine.

"The behavior of the computer at any moment is determined by the symbols which he is observing, and his 'state of mind' at that moment." The number of symbols which he can recognize is finite. "If we were to allow an infinity of symbols, then there would be symbols differing to an arbitrarily small extent." (Turing 1936-7 pp. 249—250.) The work leading from the problem statement to the answer must be carried out in some "symbol space" (Post 1936), i.e. some systematic arrangement of cells or boxes, each of which may bear (an occurrence of) a symbol. There is a finite bound to the number of occurrences of symbols (or of boxes where a symbol may occur) which he can observe at one moment. He can also remember symbols previously observed, by altering his state of mind. However "the number of states of mind which need to be taken into account is finite. ... If we admitted an infinity of states of mind, some of them will be 'arbitrarily close' and will be confused." (Turing 1936-7 p. 250.) But the computer's action must lead from a quite discrete object, namely the symbol array representing some natural number (or n-tuple of natural numbers) as argument(s), to another such object, namely the symbol array representing the corresponding function value. The possible states of mind are fixed in advance of naming the particular argument(s), as we are considering computation by a preassigned method, and do not allow mathematical invention in the midst of the computer's performance. Each act he performs must constitute a discrete change in the finite system consisting of the occurrences of symbols in the symbol space, the distribution of observed squares in this space, and his state of mind.

These limitations on the behavior of the human computer in computing the value of a number-theoretic function for given arguments, by following only preassigned rules, are of the same kind as enter in the construction of a Turing machine. The tape is the symbol space for the machine, and the machine state corresponds to the computer's state of mind.

The human computer is less restricted in behavior than the machine, as follows: (a) He can observe more than one symbol occurrence at a time. (b) He can perform more complicated atomic acts than the machine. (c) His symbol space need not be a one-dimensional tape. (d) He can choose some other symbolic representation of the arguments and function values than that used in our definition of computability.

We shall examine various possibilities under (a) — (d), and see briefly how each can be reduced to an equivalent in terms of Turing machines. We shall usually speak as though only one were being reduced, but our methods would serve to reduce any combination of them successively.

Under (a), we remark that e.g. 17 and 21 and 100 can each be observed in a single act. But a long sequence of symbols can only be observed by a succession of acts. For example, we cannot tell at a glance whether 157767733443477 and 157767733443477 are the same; "we should have to compare the two numbers figure by figure, possibly ticking the figures off in pencil to make sure of their not being counted twice." (Turing 1936-7 p. 251.)

If 17 and 21 and 100 are not only observed as units but manipulated as though each occupies a single cell of the symbol space, we need only redefine the symbols so that each of these constitutes a single symbol, in order to reduce the compound observation to a simple observation of the kind used by a Turing machine.

In actual computing we sometimes use certain marks (accent, check, movable physical pointer, etc.), which may be placed on a given square in addition to an ordinary symbol. If there are j of the ordinary symbols, and n of these special marks, any subset of which may be placed on a given square, the number of the square conditions is merely increased from $j+1$ to $(j+1)\cdot 2^n$.

As another example of behavior involving compound observation, suppose that the following sequence of symbols is printed,

$$\ldots 4401385789264\ldots,$$

that the observer's attention is centered at the figure 7 near the middle, and that he observes clearly at most five figures centered at this 7; thus the sequence of the five digits 85789 together with his state of mind determine his next act. Some digits further off may be vaguely observed, but without affecting his act. The act shall be of one of the kinds performed by Turing machines, with each separate symbol occurrence (not groups of five) occupying a square. For example, if the next act is $0Lq_d$, the printing becomes

$$\ldots 4401385089264\ldots,$$

with 38508 observed. Such behavior can be reduced to Turing machine behavior as follows. Say that the symbols are the ten Arabic digits. The behavior can be considered as that of a generalized Turing machine, in which the configuration (determining the act) is (e, f, a, g, h, q_c) where e, f, a, g, h are the digits occupying the five squares centered at

the scanned square. Corresponding to each state q_c of this generalized machine, we introduce a set of 10^4 states q_{cefgh} $(e, f, g, h = 0, \ldots, 9)$; and we modify the table so that upon reaching state q_c, a series of Turing machine acts is performed, consisting of inspections of the two adjacent squares on each side, leading to state q_{cefgh} when the four squares in question are occupied by the respective digits e, f, g, h. Not only the states q_{cefgh} must be added, but also some states to be assumed during the action leading from q_c to q_{cefgh}. Details are left to the reader. Now the act the generalized machine performed from the configuration (e, f, a, g, h, q_c) shall be performed from the configuration (a, q_{cefgh}). This reduction is an illustration of the remark that one can remember a finite number of previously observed symbols by having changed one's state of mind when they were observed.

It might be thought that the printing on still other squares may constitute part of the observation, e.g. that on certain specially marked squares (finite in number). If these squares are so located in the symbol space that the computer can find them and return by acts of the kinds performed by Turing machines (cf. the discussion of (c) to follow), this kind of compound observation can be reduced in a similar fashion to the preceding.

Under (b), the computer can alter other squares besides the scanned square. The new observed square need not be adjacent to the original. However there is a finite bound to the complexity of the act, if it is to constitute a single act of the computer. More complicated acts will require renewed motivation by reference to the observed data and the state of mind at intermediate situations between the given and resulting ones. (Indeed it can be argued that the Turing machine act is already compound; and consists psychologically in a printing and change in state of mind, followed by a motion and another change of mind. Post 1947 does thus separate the Turing act into two; we have not here, primarily because it saves space in the machine tables not to do so.)

All simple alterations of the situation, not in the Turing machine form, which are readily proposed, e.g. printing after motion instead of before, are easily expressed as successions of the atomic acts of a Turing machine. (Much more complicated operations, which could hardly be regarded as single acts, have already been so treated in § 68.)

Turning to (c), computing is commonly performed on 2-dimensional paper, and the 2-dimensional character of the paper is sometimes used in elementary arithmetic. Theoretically, we must also consider the possibility of still other kinds of symbol space. The symbol space must be sufficiently regular in structure so that the computer will not become lost in it during the computation.

From a given square or cell of the space, there will be a finite number $m+1$ of ways of moving to the same or an adjacent cell, call them $M_0, \ldots, M_m$ where M_0 is the identical motion. For example, in the plane ruled into squares, $m = 4$ (no motion, left, up, right, down), or if diagonal motions are also allowed, $m = 8$. The computer, whose act from a given situation must be determined by which one of a finite number of configurations is existing, could not use more. We lose no generality in supposing that there are the same number of directions of motion from every cell; in case there are fewer from some cells, the terminus of the rest of the $m+1$ motions may be defined to be the given cell, i.e. these as well as M_0 may be taken to be identical.

The number of cells which can ultimately be reached is therefore countable. The same cell may be reached by different successions of motions, e.g. in the plane, down and then right leads to the same square as right and then down.

We shall suppose that an enumeration without repetitions can be given of all the cells, such that the following is the case. To each of the ways of moving M_i $(i = 0, \ldots, m)$, there is a computable function μ_i such that, if x is the index in the enumeration of the given cell, then $\mu_i(x)$ is the index of the cell reached by the motion M_i. This supposition is realized by any readily imagined symbol space.

Using this enumeration, let the cell numbered x in the enumeration $(x = 0, 1, 2, \ldots)$ correspond to the x-th square counting rightward from a certain square (called the 0-th) on a linear tape.

Using methods from § 68, we can set up a Turing machine which will find the $\mu_i(x)$-th square, when started on the x-th square, if a distinguishing mark is kept on the 0-th (or -1-st) square. The computation for this purpose can be done by marking squares with accents, afterwards erased, without interfering with the printing already on them. This enables us to reduce computation in the given symbol space to computation on the linear tape of a Turing machine.

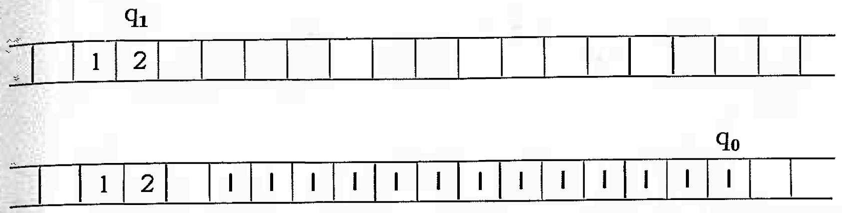
For this reduction, we did not assume that from any cell adjacent to a given cell one of the motions returns us to the given cell, i.e. that every motion in the space has an inverse. This would be the case in any ordinary symbol space. An exception is represented by the computer who receives a signal at intervals by ear.

The symbol space may consist of several disconnected subspaces, each having its own scanned cell, as e.g. in the case of a computer who

simultaneously reads a symbol on a paper by eye, reads another in braille on a tape by hand, and receives a signal by ear. If there are r such subspaces, we can reconstrue the cells to be the r-tuples consisting of a cell from each of those respective subspaces.

In regard to (d), we may argue that a natural number y is given in the original sense (§ 6), only if some sequence of $y+1$ objects, say $y+1$ tallies, is given; and hence that a procedure for computing a function φ from its argument(s), when both are expressed in some other notation, would not solve the computation problem for φ, unless the computer can also proceed from the other notation for a number y to the sequence of $y+1$ tallies, and vice versa.

According to our other arguments, Turing machines could then be built which, given the other notation for y would supply the $y+1$ tallies, and vice versa. Details can be arranged as in the definition of computation within one system of notation. Thus for decimal notation, the first machine started in the first of the following situations would go to the second.



For the familiar systems of notation, such as the dual or decimal, the existence of such a pair of machines can be established.

We have been defending Turing's thesis for number-theoretic functions; but Turing machines apply equally well to expressions in any language having a finite list of symbols. By using them as just illustrated for the case of converting one notation for a natural number into another, we get a direct way of characterizing 'effective' operations on expressions in such languages, as an alternative to requiring a corresponding number-theoretic function under a particular effective Gödel numbering to be general recursive or computable (§ 61). The method extends to languages having an enumerable infinity of symbols, whenever the symbols can be considered effectively as composed in turn from the symbols of some finite list; e.g. to the formal number-theoretic symbolism, by regarding the variables $a, b, c, \ldots$ as $a, a_{|}, a_{||}, \ldots$ (§§ 16, 50).