Notre Dame Journal of Formal Logic
Volume 23, Number 1, January 1982

Acceptable Notation

STEWART SHAPIRO*

Mechanical devices engaged in computation and humans following algorithms[1] do not encounter numbers themselves, but rather physical objects such as ink marks on paper. Since strings are the relevant abstract forms of these physical objects, algorithms should be understood as procedures for the manipulation of strings, not numbers. Furthermore, mathematical automata, such as Turing machines, which are the abstract forms of computation devices, have only appropriately constituted strings for inputs and outputs. It follows that, strictly speaking, computability applies only to string-theoretic functions and not to number-theoretic functions. That is, a string-theoretic function is said to be computable iff there is an algorithm that computes it.

Throughout the literature, however, computability is said to apply to number-theoretic functions through *notation*. A *notation d* consists of a finite alphabet, a solvable class of strings on this alphabet, called the class of *numerals,* and a convention which assigns to each numeral x a natural number dx, called the *denotation* of x. The following are common notations:

E1. Stroke notation: The alphabet consists of a single character |, called a "stroke". The class of numerals is the entire class of strings on this alphabet, including the null string. That is, a numeral in stroke notation is a finite sequence of strokes. A given numeral is taken to denote the number of strokes it contains. In what follows, for each natural number n, let $\underline{n}$ be the stroke numeral for n.

E2. Arabic notation: The alphabet consists of the following ten characters: $\langle \bar{0}, \bar{1}, \bar{2}, \bar{3}, \bar{4}, \bar{5}, \bar{6}, \bar{7}, \bar{8}, \bar{9} \rangle$. The class of numerals consists of the ten single character strings, together with all multiple character strings which do not

*I would like to thank an anonymous referee and the members of the Buffalo Logic Colloquium for helpful comments on earlier versions of this paper.

Received November 13, 1979; revised May 11, 1981

begin with $\bar{0}$. The convention which assigns denotations to these strings is well-known. In what follows, for each natural number n, let $\bar{n}$ be the Arabic numeral for n.

Let S be the numeral class for a notation d; let $\mathbf{N}$ be the class of natural numbers. Each string-theoretic function $f\colon S \to S$ has a number-theoretic counterpart $f_d\colon \mathbf{N} \to \mathbf{N}$ relative to d, such that $f_d(dx) = df(x)$.

Notice that if the convention of d is not one to one, then f_d may not be well-defined and that if the convention of d is not onto then f_d may not be defined at every number. It is assumed, therefore, that a notation convention is a bijection from S to $\mathbf{N}$. Let $\bar{d}\colon \mathbf{N} \to S$ be the inverse of the convention of d. If $F\colon \mathbf{N} \to \mathbf{N}$ is a number-theoretic function, let $F^d\colon S \to S$ be its string-theoretic counterpart relative to d: $F^d(\bar{d}X) = \bar{d}F(X)$.

Finally, if F is a number-theoretic function, we say that F is *computable relative to d* iff the string-theoretic F^d is computable—iff there is a string-theoretic algorithm that computes F^d.

Section 1 below contains two theorems which indicate that there are different classes of computable number-theoretic functions corresponding to different notations. It is shown, in particular, that the class of number-theoretic functions which are computable relative to every notation is too narrow, containing only rather trivial functions, and that the class of number-theoretic functions which are computable relative to some notation is too broad containing, for example, every characteristic function. It becomes clear, moreover, that not all notations are acceptable. The purpose of Section 2 is to formulate and defend a formal criterion for notation acceptability and, thereby, a general concept of number-theoretic computability.

1 Notice that each number-theoretic constant function is computable relative to every notation and that the number-theoretic identity function is computable relative to every notation. Notice also that each function which differs from one of these at only a finite number of arguments is computable relative to every notation (because a table of the finite exceptions and their values can be included in an algorithm). The first theorem is that this list is complete.

T1 *The only number-theoretic functions which are computable relative to every notation are almost constant functions: functions F where $\exists p \exists m \forall n > m(F(n) = p)$; and almost identity functions: functions G where $\exists m \forall n > m(G(n) = n)$.*

Proof: Let H be a number-theoretic function which is neither an almost constant function nor an almost identity function. The task is to show that there is a notation e such that H^e is not computable. We use the following lemma:

L1 *There is a set P of natural numbers such that*

(i) *the set $H^{-1}(P) - P$ is infinite*
(ii) *the set $N - (P \cup H^{-1}(P))$ is infinite.*

That is, there is a set P such that (i) *the preimage of P contains infinitely many* *numbers not in P and* (ii) *there are infinitely many numbers which are neither in P nor in the preimage of P.*

We first show that L1 entails T1. Let A be any solvable, coinfinite set of Arabic numerals that has the same cardinality as P, and let $a_1, \ldots$ be an enumeration of the elements of A. Let B be a nonsolvable set of Arabic numerals disjoint from A, and let $b_1, \ldots$ be an enumeration of the elements of B. Finally, let $c_1, \ldots$ be an enumeration of the Arabic numerals which are neither in A nor in B. Notice that the latter two enumerations must be infinite (but the first may be finite).

Notation e is formed as follows: The numerals of e are the strings $\bar{0}p$, $\bar{1}p$, $\bar{2}p$, . . . The denotation of a_ip is the i^{th} smallest element of P; the denotation of b_ip is the i^{th} smallest element of $H^{-1}(P) - P$; and the denotation of c_ip is the i^{th} smallest element of $\mathbf{N} - (P \cup H^{-1}(P))$.

Suppose that H^e were computable. The set of e-numerals that denote elements of P is the set of strings $\{a_ip\}$. Concerning computability, this set is equivalent to A, and thus is solvable. By the assumption of H^e, then, the set of e-numerals that denote elements of $H^{-1}(P)$ is also solvable. The latter set of numerals is equivalent to a set D of Arabic numerals such that $B \subset D$ and $D \subset A \cup B$. A decision procedure for D, however, can be combined with a decision procedure for A to produce a decision procedure for B. This contradicts the condition on B. Thus, H^e is not computable. It remains to prove the lemma.

Proof of L1: If there is a natural number n which has an infinite number of preimages under H, then let P be the singleton $\{n\}$. Condition (i) follows from the characterization of n and condition (ii) follows from the fact that H is not almost constant.

Suppose, then, that each natural number has at most a finite number of preimages under H. It follows that the range of H is infinite. Construct the nested sequences P_i, R_i of finite sets of natural numbers as follows:

```
$P_0 = R_0 = \phi$
Stage $n + 1$:
    If  (a) $H^{-1}(n) \not\subset P_n \cup \{n\}$
        (b) $H^{-1}(n) \cap R_n = \phi$
        (c) $H(n) \notin P_n$
    and (d) $n \notin R_n$,
    then set $P_{n+1} = P_n \cup \{n\}$, let $r_n$ be the smallest number not in the
    finite set $P_{n+1} \cup H^{-1}(P_{n+1}) \cup R_n$ and set $R_{n+1} = R_n \cup \{r_n\}$.
    Otherwise, set $P_{n+1} = P_n$ and $R_{n+1} = R_n$.
Finally, let $P = \bigcup_{n \in \mathbf{N}} P_n$ and $R = \bigcup_{n \in \mathbf{N}} R_n$.
```

To show, first, that P is infinite, suppose that it is finite. Let n be the largest element of P. Notice that if $m > n$, then $P = P_m = P_{n+1}$, so $R = R_m = R_{n+1}$. In particular, R is finite. Let q_1 be the largest element of R. Let q_2 be a number such that if $m > q_2$, then $H^{-1}(m)$ is disjoint from $P \cup R$. Let q_3 be a number such that if $m > q_3$, then $H(m) \notin P$ (this is possible because no natural number has infinitely many preimages under H). Finally, let $q = \max(n + 1,$

q_1, q_2, q_3). Since H is not an almost identity, there is a number $s > q$ such that s is in the range of H and $H^{-1}(s) \neq \{s\}$. It follows that conditions (a), (b), (c), and (d) are all satisfied at stage $s + 1$ and, therefore, that $s \in P_{s+1}$ and $s \in P$. Since $s > q > n$, this is a contradiction. Thus, P is infinite.

re (i): Suppose now that $H^{-1}(P) - P$ is finite. Then there must be a number n such that $H^{-1}(P) - P = H^{-1}(P_n) - P$. Let $m > n$ be an element of P. By (a), there is an element s of $H^{-1}(m)$ that is not in $P_m \cup \{m\}$. Since $H(s) = m$ and $m > n$, we have $s \notin H^{-1}(P_n)$ and, a fortiori, $s \notin H^{-1}(P_n) - P$. So $s \notin H^{-1}(P) - P$ (by the characterization of n). However, $s \in H^{-1}(P)$, so we must have $s \in P$. Since, by definition, $s \notin P_m$, we must have $s > m$. But in this case, condition (c) is not satisfied at stage $s + 1$. This contradicts $s \in P$.

re (ii): Since R_n always has the same cardinality as P_n, R is infinite. Notice that conditions (b) and (d) and the definitions of R_n, R, and P entail that R is disjoint from both P and $H^{-1}(P)$. Thus, the infinite set R is contained in $\mathbf{N} - (P \cup H^{-1}(P))$. This completes the proof of T1.

The following corollary is immediate:

C1 *The only sets of natural numbers whose characteristic functions are computable relative to every notation are finite sets and cofinite sets.*

In contrast with this, many (but not all) number-theoretic functions are computable relative to some notation:

T2 *Let F be a number-theoretic function. There is a notation d such that F^d is computable iff there is a permutation T of the natural numbers such that $T^{-1}FT$ is computable relative to stroke notation.*

If one accepts Church's thesis in the form (similar to that given by Turing) that a number-theoretic function is recursive iff it is computable relative to stroke notation, then T2 can be more concisely formulated:

T2′ *There is a notation d such that F^d is computable iff there is a permutation T such that $T^{-1}FT$ is recursive.*

Proof (of T2): (a) Suppose that F is computable relative to notation d. Let $p_0, p_1, \ldots$ be an effective enumeration of the numerals of d. Let T be the permutation of the natural numbers such that $T(n)$ is the denotation in d of p_n. The following is a procedure for computing $T^{-1}FT$ relative to stroke notation:

> Given $\underline{n}$, enumerate the list $p_0, p_1, \ldots p_n$. Apply the algorithm for F^d to p_n, and let q be the result. Then enumerate the list $p_0, p_1, \ldots$ until q appears. Say q is p_m. Output $\underline{m}$.

(b) Suppose now that there is a permutation T such that $T^{-1}GT$ is computable relative to stroke notation. Construct notation e' as follows: The numerals of e' are the strings c, $c|$, $c||$, $c|||$, etc. The denotation in e' of $c\underline{n}$ is $T(n)$. The following is an algorithm for $G^{e'}$:

> Given $c\underline{n}$, apply the algorithm for $T^{-1}GT$ (relative to stroke notation) to $\underline{n}$ and let $\underline{m}$ be the result. Output $c\underline{m}$.

This completes the proof. The following is an easy consequence of T2:

C2a *For each set of natural numbers M there is a notation e such that the characteristic function of M is computable relative to e.*

At this point, the possibility that there are functions which are not computable relative to any notation might be surprising. As an aside, the following corollary is presented:

C2b *There is a number-theoretic function which is not computable relative to any notation.*

Proof (outline): For each number-theoretic function F, let $S(F)$ be the set of natural numbers $\{n | n \neq 0$ and there is a natural number which has exactly n preimages under $F\}$. Notice first that if T is a permutation of the natural numbers then $S(F) = S(T^{-1}FT)$. Notice also (by an extended version of Church's thesis) that if F is computable relative to stroke notation, then $S(F)$ is recursively enumerable in the halting problem. It follows from this and T2 that if $S(F)$ is not recursively enumerable in the halting problem, then F is not computable relative to any notation. A function J with this property is now presented. Let $a_0, a_1, a_2, \ldots$ be an enumeration of a set of positive integers which is not recursively enumerable in the halting problem. Form the sequence $\langle b_i \rangle$ as follows: $b_0 = a_0 - 1$, $b_{n+1} = b_n + a_{n+1}$. Finally, let $J(0) = \ldots = J(b_0) = 0$, $J(b_0 + 1) = \ldots = J(b_1) = 1, \ldots, J(b_n + 1) = \ldots = J(b_{n+1}) = n + 1, \ldots$. Notice that $S(J)$ is the set enumerated by $\langle a_i \rangle$, and, therefore, that $S(J)$ is not recursively enumerable in the halting problem.

It seems that we have not, as yet, developed a general concept of number-theoretic computability. To do this, of course, further restrictions must be placed on notations. Clearly, many of the above notations are not acceptable, but both Arabic notation and stroke notation are. In the next section, a formal criterion for notation acceptability and (thereby) a general concept of number-theoretic computability are developed.

2 Under normal circumstances, a person engaged in computation is not merely following an algorithm. It is usually important, in particular, that the computist know the number-theoretic goal of the algorithm. This suggests two informal criteria on notations employed by algorithms:

(1) The computist should be able to *write* numbers in the notation. If he has a particular number in mind, he should (in principle) be able to write and identify tokens for the corresponding numeral.

(2) The computist should be able to *read* the notation. If he is given a token for a numeral, he should (in principle) be able to determine what number it denotes.[2]

It is admitted that these conditions are, at best, vague and perhaps obscure. Taken literally, for example, the phrase "have a number in mind" seems to involve the possibility of *de re* knowledge of particular natural numbers independent of notation. The phrase "determine a number" involves the possibility of mentally determining a number with a given property, again independent of notation. The following less problematic conditions are sufficient for present purposes. Conditions (1a) and (1b) follow from (1); condition (2a) follows from (2).

(1a) If the computist is given a finite collection of distinct objects, then he can (in principle) write and identify tokens for the numeral which denotes the cardinality of the collection.

(1b) The computist can *count* in the notation. He is able (in principle) to write, in order, tokens for the numerals denoting any finite initial segment of the natural numbers.

(2a) If the computist is given a token for a numeral p and a collection of distinct objects, then he can (in principle) determine whether the denotation of p is smaller than the cardinality of the collection and, if it is, produce a subcollection whose cardinality is the denotation of p.

Let us say that a computist *knows* a notation d iff conditions (1) and (2) hold for her with respect to d. It is easy to see, for example, that most people know Arabic notation and can easily come to know stroke notation.

If a computist does not know a particular notation, then it is hard to see a sense in which she understands the number-theoretic goal of an algorithm which employs the notation. If, for example, an algorithm for addition uses a notation which is not known by a computist and she is given two numerals in the notation, she could not know that the algorithm determines the sum of the denoted numbers. That is, the computist could not *use* the algorithm to add numbers.

The following informal lemmas will be useful. Although the "proofs" rely on conditions (1) and (2), they can easily be reconstructed to rely on only (1a) and (2a).

IL1 *Suppose that a person knows notations d and b. Then he can be taught an effective procedure to translate d into b without substantially increasing his mathematical knowledge. That is, a translation procedure is an immediate consequence of his knowledge.*

Proof: The following is such a procedure: Given p, a numeral in d, read d to find which number p denotes and write that number in notation b.[3]

If it is agreed that every normal person knows (or can easily come to know) stroke notation, then the following is immediate:

IL2 *A person knows (or can easily be taught) notation d iff she knows (or can easily be taught) either an effective procedure to translate d into stroke notation or an effective procedure to translate stroke notation into d.*

For the final informal lemmas, let us say that a person B *can calculate* a string-theoretic function f iff B can be taught an algorithm P for f and B can be taught that B realizes f, both without substantially increasing his mathematical knowledge. That is, B can calculate f iff an algorithm for f is an immediate consequence of his knowledge. Let s be the successor function.

IL3a *If a person knows notation d, then he can calculate* s^d.

Proof: Suppose a person knows d. If he is given a numeral p in d, then (by (2)) he can find the number n denoted by p and (by (1)) he can find the numeral denoting $n + 1$.

IL3b *If a person can calculate s^d and knows the numeral in d which denotes zero, then he knows (or can easily come to know) d.*

Proof: To write any number n in d, the person need only successively apply the algorithm for s^d to the numeral denoting zero $\underline{n}$ times. $\underline{\text{He}}$ can determine the denotation of a numeral p by computing $\bar{d}0$, $s^d(\bar{d}0)$, $s^d(s^d(\bar{d}0))$, etc., until p is produced.

The formal criterion of notation acceptability is obtained by objectifying the above concept of "knows" along the lines of IL3:

Notation d is *acceptable* iff s^d is computable.

The following analogues of IL1 and IL2 are trivial:

T3 *If notations d and b are acceptable, then there is an effective (string-theoretic) procedure which translates d into b.*

C3 *Notation d is acceptable iff either there is an effective (string-theoretic) procedure which translates d into stroke notation or there is an effective (string-theoretic) procedure which translates stroke notation into d.*

Finally, we say that a number-theoretic function F is *computable* iff there is an acceptable notation d such that F is computable relative to d. The following are immediate:

T4 *F is computable iff F is computable relative to every acceptable notation.*

T5 *F is computable iff F is computable relative to stroke notation.*

Theorem T5, together with the above formulation of Church's thesis, entails the general Church's thesis:

CT *F is computable iff F is recursive.*

NOTES

1. The term 'algorithm' is understood here in its intuitive or preformal sense as an effective procedure for computation. That is, it is not meant to be restricted to a particular algorithm formulation, such as the Markov-formulation or the Post-formulation.

2. Following the (nonconstructive) notion of "ability in principle", each of conditions (1) and (2) implies the other. For example, if a computist can read a notation, then he can write it. Indeed, if he has a number in mind, he can determine the corresponding numeral by systematically enumerating and reading the numerals. A similar line of reasoning establishes the converse.

3. Because both this procedure and that in the proof of IL3a essentially refer to numbers, they are not procedures for manipulating strings. Therefore, as presented, they cannot be executed by a machine. Some authors (such as Kreisel) make a distinction between human computability and machine computability on similar grounds. In this case, however, it is possible to give procedures (relying on conditions (1a) and (2a)) which do not involve numbers.

Department of Philosophy
Ohio State University, Newark Campus
Newark, Ohio 43055