

PHIL 280
Assignment 1 Key¹

We've seen in class and in handouts that in order for a list of instructions A (describing or implementing a process) to be an algorithm, then A must be:

1. **General** (work for all inputs)
2. **Finite**
3. **Unambiguous**
4. **Deterministic**

(As discussed in Handout II-1.) Moreover, aside from some of the non-trivial tasks for checking for determinacy (4.) for *all possible cases* concerning a particular class of inputs, there are inherent conceptual ambiguities (circularities resulting from context-dependence) linked up with non-ambiguity (3.) which the notion of a Turing Machine TM seeks to resolve. Items 1. – 4. above can be also be specified more sharply according the criteria (handout "Algorithms"):

- A1) A is exactly specified (**precise**)
- A2) The instructions & auxillary storage of A must be of finite length (**finite space**)
- A3) A must produce its output O in a finite number of steps (**finite time**) such that:
- A4) For any legal input I : (**completeness**)
- A5) A gives the correspondingly correct output² O (**correctness**)

- Note how A2), A3) refine notion 2. (finiteness), how A1) refines 3.) (no ambiguity) and most importantly how A4) and A5) refine 1. & 4. (generality and determinacy): Generality, restricted to the case of a legal input should always produce a definite output (A should be complete) such that A consistently gives the *same* (unique) answer (A should be deterministic) which must be correct.³

The article mentions finiteness (2.) but doesn't explicitly spell this out in terms of A2), A3). Instead, an allusion is made to *definiteness* and *effectiveness*. The latter, we may assume, corresponds to A3) as *effective procedure* is briefly discussed a few paragraphs later. However, in the former case, whatever the technical parlance, it is misleading to mention 'definiteness' without more explanation, since one naturally associates this idea with (3.) and (4.) (or, respectively, with A4) and A5)).

The article qualifies innate ambiguities associated with the concept of algorithm to 'what a computer can do.' "[A]lgorithms are thought of as recipes, methods, ...[1] for getting *computers* to do something, ...[2] when restricted to computers, the term 'algorithm' becomes more precise...because [this]...means 'a computer can do it.'"(11) Though Dietrich later explains [2] in more detail in his discussion on virtual machines (how this avoids the homunculus connotations (11-12)), the ambiguities in notions such as 'recipe,' 'methods' are swept aside (or under the rug) in computer science parlance. However, as Dietrich would no doubt agree, the notion of algorithm logically precedes computer science notions. ("The connection with computers is not necessary..."(11)) Thus, for example, he doesn't address how to precisify inputs (A1) in a meaningful way (logically prior to introducing technical notions like

¹ In the form of remarks, not in a report or philosophy paper fashion

² In the handout, the output corresponded to the specific question of determining whether or not an element x belongs to a set S or not.

³ Consider, in this vein, measuring the weight of something on a scale. (Assume that the scale functions properly). The scale's output is *deterministic* insofar as it will always give a unique weight for a given object or objects (i.e. one or more objects, i.e. the inputs for the scale) can yield the same weight, but one object can never yield more than one weight when measure on the scale. The scale's readings obey *completeness* insofar as a non-zero weight is produced for any *legal* input. (The latter is determined by the precision and capacity of the scale...for instance, a feather and a grand piano are illegal inputs in the case of a common bathroom scale.) **Note:** in this example concerning the scale, leave technical worries aside concerning the inherent vagueness (continuous error-bars) of all physically measured quantities (recall the discussion of the 'mug of coffee' in the MIT article.) (OVER →)

TMs) as contrasted, for instance, in the “Nature of Computation” handout. For example, the Euclidean GCD algorithm suffers the same shortcomings as that of the STE-GCD: how is the name/value dichotomy resolved in the input command “Given two integers?” It is as ambiguous as “guess” is in Step1 of STE-GCD.

In the article’s favor, mention is made concerning the infinite number of ways to *intensionally* specify an algorithm given its (only one) *extensional* specification. That is to say, informally, there exists a non-unique number of ways to solve a problem with one unique input and output. Consider, for example, the two different versions of Euclid’s GCD quasialgorithms discussed in the handout.

More fully algorithmic specifications of Euclid’s GCD include non-recursive and the recursive variety (which take about the same number of steps to compute)⁴: (comment lines in “[...]” format)

I. Euclidean Algorithm (non-recursive) EA[integer,integer]

[Input non-zero m, n]

[DIV means ‘get the divisor of’. Example: 5DIV2 = 2]

[MOD means ‘get the remainder of.’ Example: 5MOD2 = 1]

[Output $d = \text{GCD}(m,n)$, and integers s, t such that $d = sm + tn$]

[Auxiliary variables: integers q,a,a',s,s',t,t']

$a \rightarrow m, a' \rightarrow n, s \rightarrow 1, s' \rightarrow 0, t \rightarrow 0, t' \rightarrow 1$

while $a' \neq 0$ **do**

$q \rightarrow a\text{DIV}a'$

$(a,a') \rightarrow (a', a - qa')$

$(s,s') \rightarrow (s', s - qs')$

$(t,t') \rightarrow (t', t - qt')$

$d \rightarrow a$

return d,s,t

II. Euclidean Algorithm (recursive) EA[integer,integer]

[Input non-zero m, n]

[Output $d = \text{GCD}(m, n)$]

if $n = 0$ **then**

return m

else

return EA[$n, m\text{MOD}n$]

Note how II renders precise the recursive procedure informally discussed in pages 4-5, in “The Nature of Computation” handout.

-William Kallfelz
Grader PHIL 280
301 405-5841
wkallfel@umd.edu
SKN 1120C

Office hours Tuesday 11:00am-1:00pm

⁴ Borrowed and modified from Ross & White (2003) *Discrete Mathematics* 5th ed, Prentice-Hall. Don’t worry about the specifically technical details here, the important point is take note of the necessary steps one needs to make to ‘precisify’ *before* introducing any specific computer scientific notions. The algorithm above is merely a more carefully and logically spelled-out from informal (standard English) versions.