POSTSCRIPT ABOUT NP-HARD PROBLEMS

D. E. Knuth

The proposal to call a problem NP-hard if every problem in NP reduces to it, as discussed in the January SIGACT newsletter, seems to have gotten widespread support. (For example, I have had favorable reactions from Steve Cook, Dick Karp, Albert Meyer, and several others, and no unfavorable reactions so far.) However, these people have convinced me that I should use reducibility in Cook's sense as opposed to Karp's in the definition of NP-hard. The main reason is that we naturally think of the complement of a problem being just as "hard" as that problem is, when using actual computers.

This led me to think about the terminology a little more, and I think I have found a decent way to avoid the dilemma between Cook's "Turing reducibility" and Karp's "many-one reducibility". The former has a standard meaning in recursive function theory which is different from the meaning we now intend, and the latter appears to be a simple transformation of the data, hence I propose to use different words for the two concepts.

If L_1 and L_2 are problems, let us say

L_1 <u>transforms</u> <u>to</u> L_2

if there is a polynomial-time transformation

$$f\colon \ \mathrm{domain}(L_1) \to \mathrm{domain}(L_2)$$

such that $x \in L_1 \Leftrightarrow f(x) \in L_2$. Let us say

L_1 <u>reduces</u> <u>to</u> L_2

if there is a way to solve L_1 by a deterministic polynomial-time program if that program is allowed to use the operation of computing the characteristic function of L_2 in unit time.

The advantage of the above definitions in lectures (where we have to speak what we mean instead of simply writing it) is obvious. There doesn't seem to be any problem in mentally confusing reducibility with transformability either, since transforming is done by a data transformation, while reducing is done by any construction which makes the efficient solution of one problem reduce to the efficient solution of another.

The terminology "L is NP-hard" means that any problem in NP reduces to L . If we wish to state the stronger condition that any problem in NP transforms to L , we may use stronger language and say "L is NP-hard by transformation".

This terminology has further advantages in that it extends easily to other relations; e.g. " L_1 linearly transforms to L_2 " might mean that the transformation increases the length of input by at most a constant factor. The statement " L_1 transforms to L_2 in linear time" has an obvious meaning, given a particular computational model, as does the statement " L_1 reduces to L_2 in linear time". It seems wisest to single out polynomial time for the words transform and reduce when they are used without qualification, because this makes them independent of the common models of computations.

I proposed this to Meyer and he used it in five lectures at Stanford in January; the terminology consistently worked well.

P.P.S. I have found a fatal error in my supposed proof that reducibility often implies transformability, so please ignore that part of my January note.

****

REGULARITY PRESERVING FUNCTIONS*

S. Rao Kosaraju

The Johns Hopkins University
Electrical Engineering Department
Baltimore, Maryland

A function f: $\{0,1,2,\ldots\} \to \{0,1,2,\ldots\}$ is a regularity preserving function (rpf) if and only if for every regular set R, $\{x \mid (\exists y)\ (xy \in R \ \&\ |y| = f(|x|))\}$ is a regular set. It was shown in [1] that any linear function is an rpf. Some time ago we proved [2] that polynomial and exponential functions are rpf's. In a recent technical report [3] we obtained various results on the class, S, of rpf's:

* This research was supported by the National Bureau of Standards under contract FL13-3-35999