Why Naive Program Transformation Systems Are Unlikely to Work

Look how carefully the title has been worded! No developer of a program transformation system need feel offended, for I have given him two escapes. Firstly, I am not arguing an impossibility, but only an unlikeliness — and we know that all startling advances have been made against seemingly overwhelming odds, don’t we? Secondly, he has the option to declare that the program transformation system he is developing is not “naive” in the sense that I shall make more precise below.

I take the position that a serious programmer has at least two major concerns, viz. correctness and efficiency. And from existing software we can deduce that neither of these two concerns is a trivial one.

For years I have argued what I still believe, namely that, when faced with different concerns, we should try to separate them as completely as possible and deal with them in turn. For correctness and efficiency concerns this separation has been achieved up to a point. It is possible to treat the problem of program correctness in isolation from the problem of efficiency in the sense that we can deal with the correctness problem, temporarily even ignoring that our program text also admits the interpretation of executable code. It is also possible to investigate the various cost aspects of program execution independently of the question whether such execution of the program will produce a correct result.

Presented as in the previous paragraph, the separation sought seems to have been found. It is true that the separation is reachable as far as the program text itself is concerned; in the process of composing the text, however, the separation is less marked. There does exist a formal discipline that, when adhered to, cannot lead to an incorrect program. In its application, however, we have a great amount of freedom, and in the choice how to
apply the discipline ensuring correctness, the designer always makes up his mind by considering his other concerns, such as efficiency. In other words, the more rigorous the concerns have been separated with respect to the program text itself, the more schizophrenic the act of program composition becomes: the programmer still remains a jack of many trades, switching all the time — and at a high frequency! — between various rôles, whose differences have only become more and more marked over the last decade.

* * *

Program transformations have been presented as a possible means to overcome the need for such a schizophrenic programmer behaviour. A number of so-called “semantics-preserving program transformations” have been discovered. Each such transformation, when applicable and applied to a program $A$ generates a new program $A'$ that, when executed, will produce the same result as the original program $A$, the difference being that the costs of execution of $A$ and of $A'$ may differ greatly. Program $A'$ may also be derived by successive applications of a sequence of such transformations.

It was the discovery of (sequences of) such transformations that supported the idea of what I call “naive” program transformation systems. When using such a system for the development of a program, this development was envisaged to take place in two successive, clearly and rigorously separated, stages.

In the first stage the programmer would only be concerned with program correctness: unencumbered by efficiency considerations he would write a program, whose correctness could be established as easily as possible. In the ideal case, the program's correctness would be trivial to establish.

In the second stage — which in the dreams of some could or should be conducted by a different person, unfamiliar with the original problem — the correct but inefficient program would be subjected to semantics-preserving transformations from a library, until the program had become efficient as well. (At the moment this dream was dreamt, the available library of acknowledged transformations was admittedly still somewhat small, but it was constantly growing and hopes were high.)

* * *

When such systems were proposed to me I was very sceptical, but I was mainly so for a purely personal reason and accidental circumstance. Their advocates tried to convince me of the viability of their approach by composing according to their proposed method a program I had published myself. In their demonstrations, stage two required about ten pages of formal labour, while stage one had taken them between one day and one week.

It so happened that their demonstrations were not very convincing for me, because, heading schizophrenically towards a correct and efficient solution, I myself had solved the whole problem (without pencil and paper) in fifteen minutes. (It was the evident effectiveness of the heuristics applied