On "almost-invariants"

In multiprogramming, a system invariant X is an assertion satisfying (0) and (1) :

(0) $[\ \text{Pre} \Rightarrow X\ ]$ —where Pre is the program precondition—

(1) X is maintained by each $\{Q\}\ S$ in the multiprogram .

Occasionally, we come across what might be called "almost-invariants" : these satisfy (1) , but not (0) ; they are stable in, but not necessarily established by, the program.

Naturally, if an almost-invariant is not established by the program, it is not very interesting. But it may happen that the program does establish it, just not in its initial state. In this case, the notion of a almost-invariant could still be very useful, for instance, if we only need to know that it holds upon termination.

In order to formalize this, we could introduce a fresh variable to "encode" the difference between initial and final states of the program, embellish the assignment establishing the almost-invariant with an assignment to this variable, etc. But this introduces all sorts of unilluminating assignments and assertions into our code, and the assertions may beget new unilluminating assertions that will beget unilluminating proofs of their correctness, etc. It may be far simpler to establish "textually" , "topologically" , or operationally that the almost-invariant is necessarily established. (For the same reason that the catchword Widening is so greatly preferred to repeating the proof obligations time and time again.)

We should be aware of the difficulties such a strategy faces, however. Each computation evoked by a program has an initial state, hence (0) is indeed a way of guaranteeing that X is established. But in general there is no guarantee that a statement establishing an almost-invariant will be executed, just because it appears in the program text: that statement could, for instance, be guarded by a guard that, in some computations, is never truthified.

So in general, the situation could be quite a hairy face indeed. Fortunately, we don't need a general theory; we should just be on the lookout for almost-invariants when they can be easily exploited. (Similar considerations hold for the concept of a Multibound, for example.)

We shouldn't punish nondeterminism by requiring the values of variables to be completely specified in the program precondition. And we shouldn't punish ourselves as multiprogram designers by denying ourselves the luxury of exploiting stable assertions. The moral of this note is that when only a small dose of operational reasoning is required to meet these aims simultaneously, it is a dose we should be willing to take.

A footnote

I hear from Wim Feijen that almost-invariants are well-known in the multiprogramming community, and are called "stable relations" . They become important in the domain of

progress. For example, to ensure progress of, say, a guarded skip in one component of a system:

$$\{P\}\ \textbf{if}\ B\ \rightarrow\ \mathit{skip}\ \textbf{fi} \qquad ,$$

we have to show that the rest of the system, constrained to P, will converge after a finite number of steps to a state satisfying B. Towards that end, it may be quite helpful to know some almost-invariants (or stable relations) that are established by P. For then, in the context of this progress argument, they become true invariants.

Eindhoven, 7 February 2006

Jeremy Weissmann
11260 Overland Ave. #21A
Culver City, CA 90230
USA

jeremy@mathmeth.com