Software correctness verification

It is important to note that in most cases, software correctness verification, as described by Richard Linger is informed by correctness questions where mathematical verification follows (if necessary). A popular approach in software correctness verification is the translation of a specific program as well as its criteria of correctness into a mathematical logical formula and in this regard allow a theorem provider (automatic) confirm the formula’s validity.

In this case, it is commonplace to use proof of correctness that are essentially mathematically-based to carry out design specification verification. However, according to Laski & Stanley (2009), software correctness verification can tend to be a little bit radical for utilization in actual development of software essentially due to its extreme emphasis on not only mathematical approaches but theoretical approaches as well.

It is also important to note that as things are currently, there lacks substantive (even practical) means in regard to automatic support when it comes to correctness verification undertaken in large scale in relation to behavior (intended). Thus in that regard, there is significant dependence on testing as well as inspection undertakings and this essentially means that the correctness evidence presented in such a case is largely limited.

On the other hand, software correctness verification as brought out by Richard Linger can be taken to be a reasonable goal for software development in the real world given that there is significant flexibility left out for the tool designer in regard to the specific conditions to be subjected to checks. Further, it can be noted that there is a provision that allows the underlying theorem provider discretion as to the reasoning of a given program’s various aspects.

References