Conclusive Proofs of While Loops Using Invariant Relations

Lamia Labed Jilani (ISG, Tunisia), Wided Ghardallou (FST, Tunisia), Ali Mili (NJIT, USA)

Traditional methods of verifying iterative programs rely on invariant assertions to prove partial correctness and on variant functions to prove termination. As such, they suffer from some weaknesses, which we characterize briefly and broadly as follows:

- If we attempt to prove the partial correctness of a while loop with respect to a (pre/post) specification and the proof fails, we have no simple way to tell whether the proof failed because the loop is incorrect or because the invariant assertion is inadequate.
- Even assuming we have determined that the invariant assertion is inadequate, we have no simple way to determine whether we need to adjust it by strengthening it or by weakening it: granted, if the invariant assertion does not imply the post condition or is not implied by the precondition, then the remedy is obvious; but if the invariant assertion does not meet the inductive step, it is not clear how it must be adjusted.
- When one uses variant functions to prove loop termination, one equates the termination of the loop with the condition that the number of iterations is finite, but fails to capture the situation where individual iterations of the loop body fail to terminate normally because they attempt an illegal operation (array reference out of bounds, pointer reference to null, etc).

In this paper, we present an alternative method to analyze loops, which appears to address most of the concerns raised above, and is based on the concept of invariant relations. Our approach can be characterized by the following premises.

- Any invariant relation can be converted into a necessary condition of termination; our definition of termination is comprehensive, in the sense that it encompasses the condition that the number of iterations is finite, as well as the condition that each individual iteration proceeds normally.
- Given an invariant relation, we can use it to generate a necessary condition of correctness of the loop with respect to a relational specification. This condition returns false whenever the invariant relation contradicts the candidate specification.
- Given an invariant relation, we can use it to generate a sufficient condition of correctness of the loop with respect to a relational specification. This condition returns true whenever the invariant relation subsumes the candidate specification.
- We have an automated tool that generates invariant relations from a static analysis of the source code of the loop, using a knowledge base that captures the necessary programming knowledge and domain knowledge that is required for an adequate analysis of the loop. Each invariant relation captures some functional property of the loop. If the tool runs out of invariant relations without capturing all the functional details of the loop, it gives an indication of how to complete the missing knowledge in the knowledge base.
- Using the capabilities discussed above, we generate an algorithm for proving the correctness of a loop with respect to a relational specification that extracts invariant relations until it finds one that contradicts the candidate specification (if the necessary condition is false) or one that subsumes the candidate specification (if the sufficient condition is true) or until it runs out of invariant relations (in which case it indicates what is missing from the knowledge base).