Inferring Loop Invariants Dynamically

Juan Pablo Galeotti and Andreas Zeller
{galeotti, zeller}@cs.uni-saarland.de

Saarland University – Computer Science, Saarbrücken, Germany

There is extensive literature on inferring loop invariants *statically* (i.e. without explicitly executing the program under analysis). We report on a new *dynamic* technique for inferring loop invariants based on the invariant detector Daikon [2]. Unlike InvGen [4], this new technique follows a counter example guided approach for refining candidate loop invariants. Let us consider the following annotated program for multiplying 16 bit integers in the left column:

```
_(requires 0<=x<65535)            // Candidate Loop Invariants
_(requires 0<=y<65535)            #1  x one of { 1, 1316 }
_(ensures \result==x*y) {         #2  y one of { 1, 131 }
  mult = i = 0;                   #3  i >= 0
  while (i<y) {                   ...
    mult+=x; i++;                 #9  i <= y
  }                               #10  i == (mult / x)
  return mult;                    #11  mult == (x * i)
}
```

Our approach starts by finding new test cases using the search-based test suite generator EvoSuite [3]. Then, the dynamic invariant detector collects 11 different loop invariant candidates (excerpt shown on the right), which we feed to the static verifier VCC [1].

Since the conjunction of all candidates under-approximates the loop invariant, the static verifier fails. Then, EvoSuite guides the generation of new test inputs using the static verifier's error model. The invariant detector synthesizes new candidates (ruling some of them out), which are fed to VCC. This refinement continues until VCC successfully verifies the program (using only candidates `#9` and `#11`).

The combination of test case generation and Daikon opens the potential for inferring loop invariants even for nontrivial programs. Current challenges include the static verification itself, as well as refining the candidate loop invariants.

The main challenge, however, will be to find appropriate *patterns* for the most recurrent loop invariants: Daikon itself is limited to at most three related variables, and we will have to expand the search space considerably. Finally, we are also looking for benchmarks such that we can compare against other existing automatic loop invariant detectors, such as InvGen [4].

References

1. Cohen E., Dahlweid M., Hillebrand M., Leinenbach D., Moskal M., Santen T., Schulte W., and Tobies S., *VCC: A Practical System for Verifying Concurrent C*. TPHOLs, 2009.
2. Ernst M., Cockrell J., Griswold W., and Notkin D. *Dynamically discovering likely program invariants to support program evolution*. IEEE TSE, 27(2), 2002.
3. Fraser G., and Arcuri A., *EvoSuite: Automatic Test Suite Generation for Object-Oriented Software*. ESEC/FSE, 2011.
4. Gupta A., Majumdar R., and Rybalchenko A., *From Tests To Proofs*. TACAS, 2009.