Predicated Bug Signature Mining via HI (HIMPS) User Manual

Zhiqiang Zuo
Department of Computer Science, National University of Singapore
zhiqiangzuo@comp.nus.edu.sg

1 Description

Predicated bug signature mining via hierarchical instrumentation (called HIMPS) [3] is an efficient approach to discover predicated bug signatures from execution traces. HIMPS employs the novel *hierarchical instrumentation* technique [2] to the predicated bug signature mining (MPS) [1] . It performs selective instrumentation so as to significantly enhance the efficiency of MPS.

Specifically, HIMPS mainly consists of two functional modules: boosting and pruning. Given the coarse-grained profiles, the boosting module selects a set of functions which will be instrumented for boosting a threshold. Then the pruning module takes the boosted threshold to return a set of prospective functions.

2 Requirements

In order to finally obtain bug signatures in an efficient way, the following packages or tools are required:

- Java l.6 or above: the runtime environment for running the respective functional modules (e.g., himps.jar –boost and –prune, preprocess.jar)
- sampler-cc: a predicate-based C program instrumentor for performing the instrumentation at both function and predicate levels
  `http://research.cs.wisc.edu/cbi/`
- mbs: a bug signature miner for discovering top-k bug signatures from mining dataset
  `http://www.comp.nus.edu.sg/~specmine/suncn/mps-artifacts/mps.tar.gz`
- preprocess.jar: a package for preprocessing execution profiles to generate mining dataset
  `http://www.comp.nus.edu.sg/~specmine/himps/himps.jar`
- himps.jar: himps package containing two functional modules, –boost and –prune
  `http://www.comp.nus.edu.sg/~specmine/himps/preprocess.jar`

3 Running Procedure

The following gives the procedure of running HIMPS. The inputs are a buggy program P, a test suite containing failing and passing test cases T, the number of signatures mined k and the percentage of predicates instrumented for boosting γ. Users will obtain the top-k suspicious bug signatures at the end of the following procedure.

1. instrument all function entries in the entire program P
   ```
   sampler-cc -fsampler-scheme=function-entries -fno-sample c_source_file gcc_compiler_flag
   ```
2. run the test suite T to collect coarse-grained profiles CP

初始值

3. select a set of functions *boost* whose enclosing predicates will be instrumented for boosting
   java -jar himps.jar cg_sites_file cg_profiles_folder fg_sites_file --boost γ [boost_output_file]
4. instrument all predicates in *boost*
   sampler-cc -fsampler-scheme=branches -fsampler-scheme=returns -fsampler-scheme=scalar-pairs -fno-sample [-finclude-function=function_in_boost_output]$^+$ -fexclude-function=* c_source_file gcc_compiler_flag
5. run the test suite T to collect fine-grained profiles BP 旧的
6. preprocess BP
   java -jar preprocess.jar fg_profiles_folder fg_sites_file dataset_output_folder 输出
7. mine top-k bug signatures to obtain kth top suspiciousness value 输入
   ./mbs -k k -n 0.5 -g --refine 2 --metric 0 --dfs --merge --cache 9999 --up-limit 2 mps-ds.pb [-o output_file] 输出
8. generate a set of prospective functions *prune*
   java -jar himps.jar cg_sites_file cg_profiles_folder fg_sites_file --prune θ [prune_output_file]
9. instrument all predicates in *prune*
   sampler-cc -fsampler-scheme=branches -fsampler-scheme=returns -fsampler-scheme=scalar-pairs -fno-sample [-finclude-function=function_in_prune_output] -fexclude-function=* c_source_file gcc_compiler_flag
10. run the test suite T to collect fine-grained profiles PP 新的+旧的
11. preprocess PP
    java -jar preprocess.jar fg_profiles_folder fg_sites_file dataset_output_folder
12. return top-k bug signatures
    ./mbs -k k -n 0.5 -g --refine 2 --metric 0 --dfs --merge --cache 9999 --up-limit 2 mps-ds.pb [-o output_file]

两次的总和。

References

[1] C. Sun and S.-C. Khoo. Mining succinct predicated bug signatures. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2013, pages 576–586, New York, NY, USA, 2013. ACM.

[2] Z. Zuo. Efficient statistical debugging via hierarchical instrumentation. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, ISSTA 2014, pages 457–460, New York, NY, USA, 2014. ACM.

[3] Z. Zuo, S.-C. Khoo, and C. Sun. Efficient predicated bug signature mining via hierarchical instrumentation. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, ISSTA 2014, pages 215–224, New York, NY, USA, 2014. ACM.

1 instrument
2 run
3 preprocess
4 mine

1 function
2 boost
3 prune

左边每一项数据是右边三步的汇总