Detecting Semantic Bugs in Autopilot Software by Classifying Anomalous Variables

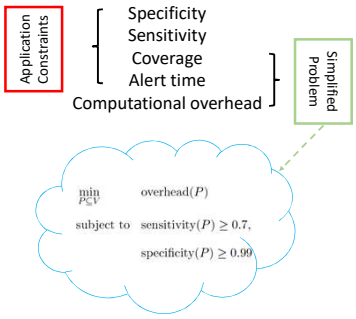
Hu Huang, Liangchun Xu, Samuel Z. Guyer, Jason H. Rife
Tufts University

Abstract

We construct a bug detector to detect semantic bugs, which leverages machine learning models as the method to interpret the data. We formulate the problem of identifying relevant variables formally as an optimization problem, which is to locate a set of variables that minimizes overhead and satisfies our performance constraints. Our experimental results show variables identified in the program slice enable our learning models to perform significantly better compared to variables comprising the system inputs and outputs. Additionally, we implement two methods that select a subset of variables from the program slice, which attempts to retain bug detection performance while reducing overhead. Our results show that we can retain nearly the same bug detection performance as compared to the full slice but reduce the overhead of the bug detector by as much as 80%.

Problem

The bug detector depends on selecting an appropriate set of variables to monitor. The problem of selecting this set of variables is nontrivial because of the need to balance design requirements, which include: overhead, sensitivity, specificity, coverage and alert-time.

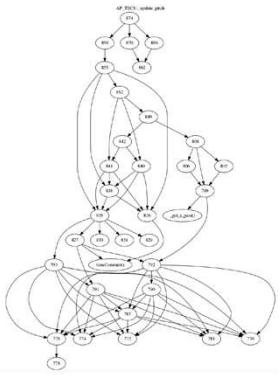


Contribution

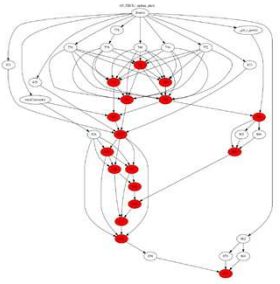
- Adapted conventional program slicing to apply to Cyber-Physical Systems. Adaptations include adding temporal segmentation and generalizing the slicing operations to account for interfaces with other software and physical components.
- Demonstrated effectiveness of program slicing (and specifically of backward slicing) as a basis for improving bug detector performance, by reducing overhead while maintaining other constraints. This demonstration was conducted by applying our bug detection tool to Ardupilot and assessing performance for real and injected bugs.

Approach

- In order to introduce a concept of time into the code, we introduce an operation we call temporal segmentation. The purpose of temporal segmentation is to annotate key points in the software where time progression is relevant.
- In order to create statement groupings with a clear input-output relationship, we introduce an operation we call generalized slicing.



(a). Backward flow graph



(b). Forward flow graph

Figure 1. a). obtained by backward slicing on variable pitch dem starting on line 874 extracted from the update pitch function in the Total Energy Control Systems (TECS) module. Nodes are line numbers and edges represent variable dependencies. In b), a forward flow graph constructed from inverting the edges of the graph shown in a) and adding a "source" node. The set of nodes in the dominance frontier is computed for each node in the forward flow graph and colored red. Notice that all of the nodes in the dominance frontier have multiple in-flow edges.

Methods

- Inject 9 bugs (Bug 0 through 8) and evaluate a real bug (Bug 7062) in TECS module in Ardupilot.
- The variables from the full program slice (SliceFull), the inputs and outputs of the program slice (SliceIO), the nodes in the dominance frontier within the slice (SliceDF), and the sensors and actuators (SysIO) are chosen.
- Both Decision Tree and AdaBoost models used the default starting parameters as in scikit-learn.

Results

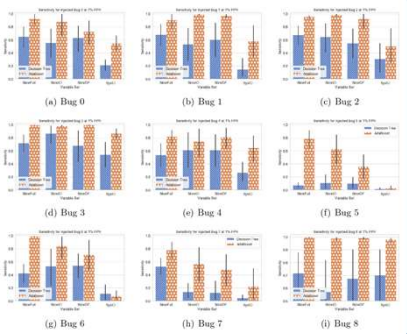


Figure 2. Model sensitivities for Decision Tree and AdaBoost for the four variable selection methods SliceFull, SliceIO SliceDF, and SysIO across all 9 injected bugs at 99% specificity, tolerating 1% false positives.

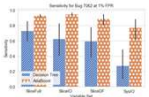


Figure 3. Model sensitivities for bug 7062 in Ardupilot grouped by the four variable selection methods SliceFull, SliceIO SliceDF, and SysIO at 99% specificity, tolerating 1% false positives.

Discussion

All three slice-based methods (SliceFull, SliceIO, and SliceDF) outperform SysIO. In addition, the nonoverlapping confidence intervals for SliceFull and SliceIO compared with SysIO (for AdaBoost) suggests that there is a significant difference between the means. This difference in performance further supports our assertion that local variables are better to detect a bug rather than variables global to the physical system. Furthermore, the three slice-based methods have very similar sensitivity, shown by the overlapping 88 error bars. Though there is a slight reduction for SliceDF, sensitivity is well above 90%. These results provide more support to our assertion that the number of variables (and by extension, overhead) can be reduced without significantly impacting performance.

Conclusion

We used program slicing to identify variables in a particular region of the code and subsequently performed graph-based operations to prune the program slice to a smaller subset to reduce the operational costs for future implementation of an online bug detector. We then demonstrated that reduced variable subsets could achieve high sensitivity for locating injected and real bugs.

Acknowledgement

This research was supported by National Science Foundation under grant CNS-1836942.

Contact

Jason H. Rife
Tufts University
Email: jason.rife@tufts.edu

References

1. Huang, Hu. Detecting Semantic Bugs in Autopilot Software by Classifying Anomalous Variables. PhD thesis.