Adaptivity Analysis

1 Related Work

1.1 History of Program Dependency Graph and Relevant Graph

[BBI89]:

1.2 Dynamic Program Graph Analysis

[SHR01]: Support Interprocedural Control dependence analyzing, semantically.
They identified the dependence information between the interactions of among procedures, specifically the control dependence between procedures. Their analysis support the relationship of control and data dependence to semantics dependence.

[AS92]: Dynamic Program Dependency Graph.
They gave the dynamic analysis for the program's dependency, by producing 3 different kinds of graph, including the data flow graph, storage dependence and control dependence graph from program's execution traces.
Then, they constructing dynamic execution graphs by adopting the 3 graph, aims to expose the parallization of the programs

[HGK06]: dynamic path conditions in dependence graphs. They adopting the dynamic information from program trace to the path condition in dependency graph. Then based on these information, they present new approach combining dynamic slicing, which could reveal both dependences holding during program execution as well as why these dependences are holding. Aims to have a finer and preciser analysis of the program.

1.3 Utilization of the Dynamic Program Dependency

[NJ18]: Utilize dependency graph for finding serializability violation.
Combine with the dependency graph of serialization and abstract execution, to statically finding bounded serializability violation. Then reduce the problem of serializability to satisfiability of a formula in FOL. Also reason about unbounded executions.

1.4 Static Program Dependency

[MZ08]: They propose ways of constructing different kinds of program slices, by choose different program dependency. For example, in either syntactic or semantics sense. This abstract dependency is based on properties rather than exact data. Aims to give finer and smaller program slice.

1.5 Utilization Static Program Flow Graph

[APB+12]: Lightweight Static Analysis for GUI Testing. They give the relevant event graph based on black and white Box. To construct finer Event Sequence Graph, they propose new approach to select relevant event sequences among the event sequences generated by black box. This new approach based on static analysis on bytecode of the applications, giving a precisely defined dependency between a fixed number of events in event sequence. Then, they inferred a finer Event Dependency graph, aims to give a better lightweight static analysis on applications.

References

[APB+12] Stephan Arlt, Andreas Podelski, Cristiano Bertolini, Martin Schäf, Ishan Banerjee, and Atif M Memon. Lightweight static analysis for gui testing. In *2012 IEEE 23rd International Symposium on Software Reliability Engineering*, pages 301–310. IEEE, 2012.

[AS92] Todd M Austin and Gurindar S Sohi. Dynamic dependency analysis of ordinary programs. In *Proceedings of the 19th annual international symposium on Computer architecture*, pages 342–351, 1992.

[BBI89] William Baxter and Henry R Bauer III. The program dependence graph and vectorization. In *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 1–11, 1989.

[HGK06] Christian Hammer, Martin Grimme, and Jens Krinke. Dynamic path conditions in dependence graphs. In *Proceedings of the 2006 ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation*, pages 58–67, 2006.

[MZ08] Isabella Mastroeni and Damiano Zanardini. Data dependencies and program slicing: from syntax to abstract semantics. In *Proceedings of the 2008 ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation*, pages 125–134, 2008.

[NJ18] Kartik Nagar and Suresh Jagannathan. Automated detection of serializability violations under weak consistency. *arXiv preprint arXiv:1806.08416*, 2018.

[SHR01] Saurabh Sinha, Mary Jean Harrold, and Gregg Rothermel. Interprocedural control dependence. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 10(2):209–254, 2001.