A Context-Sensitive Structural Heuristic for Guided Model Checking
Neha Rungta and Eric Mercer
Brigham Young University, Provo, UT

1. Introduction
- The use of embedded systems has become ubiquitous in recent times
- Growing complexity makes ad-hoc testing techniques insufficient
- Model checking exhaustively searches program states as shown in Fig. 1
- Computation resources often run out before finishing the search
- Guided search ranks states in order of interest with a heuristic
- Guided search explores interesting states first as in Fig. 2
- The goal is to find an error before running out of computation resources

2. FSM Distance Heuristic
- FSM builds an Interprocedural Control Flow Graph (ICFG) statically
- A depth-first traversal of a program (Fig. 3) builds the ICFG shown in Fig. 4
- It maps the current state to a vertex in the ICFG as shown in Fig. 5
- It computes shortest path from a current state to an error state on the ICFG
- The length of the shortest path is returned as the heuristic estimate

3. Augmented Interprocedural Control Flow Graph (AICFG)
- An AICFG includes a bounded (k) calling context of function calls
- The bound k is specified by the user
- The program now has k slots for return addresses in the runtime stack
- A traversal of the program in Fig. 3 with k=1 creates the AICFG in Fig. 7
- The vertex (0, 0, 0) in Fig. 7 has information that it was called from line 02

Fig. 1: Breadth First Search
Fig. 2: Guided Search

Breadth-first search orders the search frontier in a FIFO as shown in Fig. 1
Guided search orders the frontier in a priority queue as shown in Fig. 2
The heuristic values on the states is the estimated distance to an error
Guided search explores interesting states first as in Fig. 2
The goal is to find an error before running out of computation resources

The ICFG has no context information
The use of embedded systems has become ubiquitous in recent times
Growing complexity makes ad-hoc testing techniques insufficient
Model checking exhaustively searches program states as shown in Fig. 1
Computation resources often run out before finishing the search
Guided search ranks states in order of interest with a heuristic
Guided search explores interesting states first as in Fig. 2
The goal is to find an error before running out of computation resources

3. Augmented Interprocedural Control Flow Graph (AICFG)
- An AICFG includes a bounded (k) calling context of function calls
- The bound k is specified by the user
- The program now has k slots for return addresses in the runtime stack
- A traversal of the program in Fig. 3 with k=1 creates the AICFG in Fig. 7
- The vertex (0, 0, 0) in Fig. 7 has information that it was called from line 02

The length of the shortest path is returned as the heuristic estimate

4. Accuracy depends on k-bound
- The context information in the AICFG is limited by the bound k
- The AICFG for the program in Fig. 3 with a k=1 is shown in Fig. 11
- The reduced call graphs in Fig. 10 shows the lost context in function g
- For large k’s the size of the AICFG can become prohibitive

Fig. 7: An AICFG for the program in Fig 3
Fig. 8: Path computed after the first call to foo on the AICFG - 8 steps

5. Extended FSM Distance Heuristic
- The extended FSM (EFSM) distance heuristic recovers calling context as shown in Fig. 12 by using the return addresses for function calls in the runtime stack
- The EFMS takes the runtime stack and extracts AICFG vertices as shown in Fig. 13 that represent the call trace to the current point of execution
- The EFMS combines the dynamic information on the runtime stack in Fig. 13 with the static information on the AICFG in Fig. 14 to get a better heuristic estimate

Fig. 12: The call trace
Fig. 13: Extracting AICFG vertices based on the values of the runtime stack

Fig. 14: AICFG for Fig. 9; path computed after first call to g by the EFMS is 11 steps

6. Results
- The EFMS heuristic does a forward analysis from the current state (q0) on the AICFG in Fig. 14 to see if an error is reachable within the scope of the function g
- If an error is not reachable, or to find a shorter distance to the error, the EFMS heuristic unnests the call trace by a single call to q1 in Fig. 12
- Overhead in following call trace
- EFMS guided search is slower even though it expands fewer states
- Optimizes space over time
- The EFSM heuristic again does a forward analysis on the AICFG in Fig. 14 at q2 on all unique paths in the function f that might lead to an error
- Repeat the pattern of unnesting the call trace and then doing a forward analysis on the AICFG until the call trace is exhausted
- The EFSM combines the dynamic information on the runtime stack in Fig. 13 with the static information on the AICFG in Fig. 14 to get a better heuristic estimate

Fig. 15: A program with nested functions
Fig. 16: True Path to the error after the first call to foo - 8 steps

A visible decrease in number of states
- FSM guided search degenerates into a random search in some models

7. Conclusions
- EFSM is admissible and consistent
- EFSM results in a better estimate of the distance to the error
- EFSM results in a better estimate of the distance to the error
- EFSM does better in models with densely connected transition graphs
- Extra memory used for building and storing the AICFG is negligible
- Structural heuristics make error discovery more traceable
- EFSM can be improved further by using data flow information

Contact Information
Verification and Validation Laboratory
Computer Science Department
Brigham Young University
Provo, UT 84602
Neha Rungta, neha@byu.edu
Eric Mercer, ermc@cs.byu.edu
http://vv.cs.byu.edu/