WBS: Weighted Backtracking Strategy for Symbolic Testing of Embedded Software

Varsha P Suresh, IIITB, India

Joint work with -
Sujit Kumar Chakrabarti, IIITB, India
Athul Suresh, IIITB, India
Raoul Jetley, ABB, India

34th International Conference on Software Engineering & Knowledge Engineering

July 4, 2022

Motivation

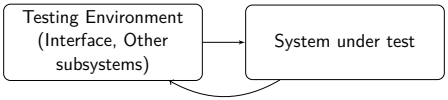
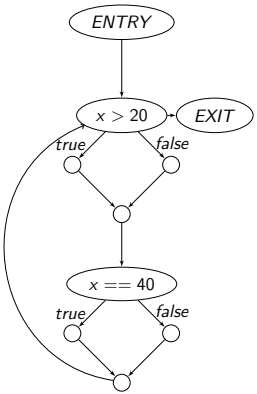


Figure: Software Interaction

- Resource intensive process in test execution: network delays, mechanical movements, memory requirements, storage requirements, human actions etc [3] [4].
- Short test sequence reduces interaction with testing environment.
- Reduces cost of testing.

Short Test Sequence



- Test Sequence: [[19],[21],[41]]
- Test Sequence: [40]

Background

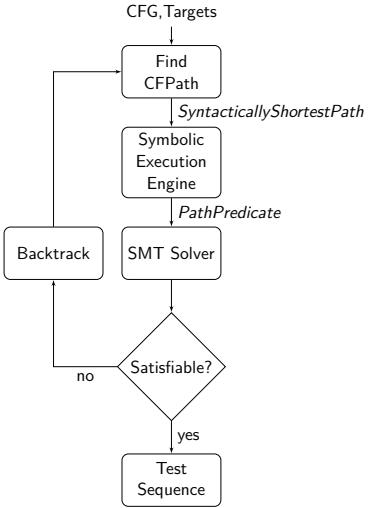


Figure: SymTest Overview [2]

Challenge

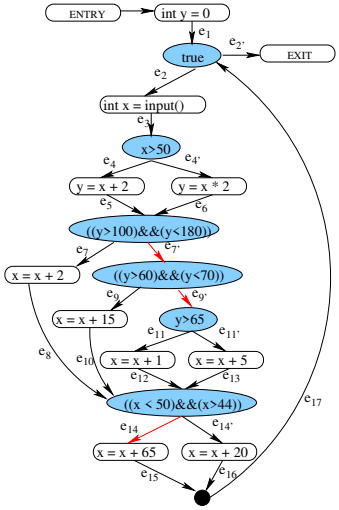


Figure: CFG of program under test

Weighted Backtracking Strategy(WBS)

- Supports backtracking by more than one decision node.

Weighted Backtracking Strategy(WBS)

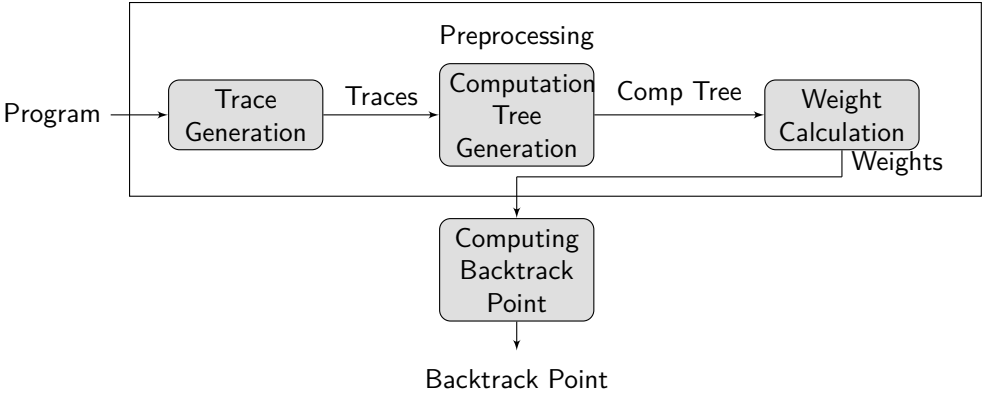
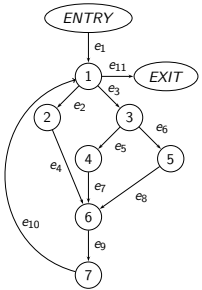


Figure: WBS Architecture

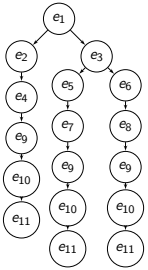
Trace Generation

- Instrument the program under test such that every time a edge is traversed during its execution, its corresponding edge id is printed into a file.
- The instrumented program is executed N number of times to generate the set of traces $\Pi = \{\pi_1, \pi_2, ..., \pi_N\}$.
- Traces are used to generate the computation tree.

Computation Tree Generation



1. $e_1, e_2, e_4, e_9, e_{10}, e_{11}$
2. $e_1, e_3, e_5, e_7, e_9, e_{10}, e_{11}$
3. $e_1, e_3, e_6, e_8, e_9, e_{10}, e_{11}$



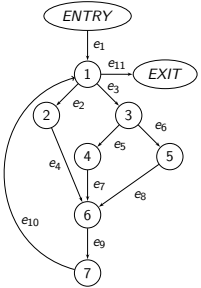
- Compressed representation of all the traces.
- Each trace is represented by a path in the computation tree.

Weight Calculation: Probability

$$W_P(e, e_{t_i}) = \frac{P_{(e,e_{t_i})}}{P} \quad (1)$$

- Previous runs
  1. $e_1, e_2, e_4, e_9, e_{10}, e_{11}$
  2. $e_1, e_3, e_5, e_7, e_9, e_{10}, e_{11}$
  3. $e_1, e_3, e_6, e_8, e_9, e_{10}, e_{11}$
- Probability to reach target edge e_9 from edge e_3,

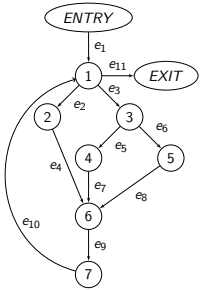
$$W_P(e_3, e_9) = \frac{P_{(e_3,e_9)}}{3} \quad (2)$$



Weight Calculation: Length

$$w_L(\mathbf{n}, e_{t_2}) = \sum_{\mathbf{n}' \in N_2} \frac{\mathbf{n}'.np}{L(\mathbf{n}, \mathbf{n}') + 1} \quad (3)$$

- $\mathbf{n}'.np$: Number of individual tree paths.
- $L(\mathbf{n}, \mathbf{n}')$: Length of the path from $\mathbf{n}$ and $\mathbf{n}'$
- Number of iterations.
- Previous runs
  1. $e_1, e_2, e_4, e_9, e_{10}, e_{11}$ - 1
  2. $e_1, e_2, e_4, e_9, e_{10}, e_3, e_6, e_8, e_9, e_{10}, e_{11}$ - 2



$$W_L(e_1, e_{t_2}) = \sum_{\mathbf{n} \in N_1} w_L(\mathbf{n}, e_{t_2}) \quad (4)$$

Weight Calculation: Composite Weight

- Individual Weight

$$W_I(e', e_{t_i}) = W_P(e', e_{t_i}) + W_L(e', e_{t_i}) \tag{5}$$

- Composite Weight

$$W(e') = \sum_{e_{t_i} \in T'} W_I(e', e_{t_i}) \tag{6}$$

Weight Calculation: Ordering

- Backtracking point selection based on composite weight may lead to longer path if the target edges are not properly ordered.
- Ordered weight is calculated.

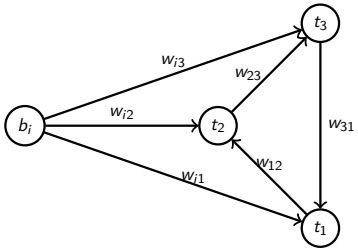


Figure: Weighted Graph

Computing Backtracking Point

1: **function** $\mathrm{BTP}(\mathcal{G},\ stack,\ W,\ T)$
2: $\quad s \leftarrow < flip(e) \forall (e, tf) \in stack >$
3: $\quad D' \leftarrow$ the subset of s such that it consists of upto N elements of s whose composed weights are the maximum.
4: $\quad$ **for all** $d \in D'$ **do**
5: $\quad\quad T' \leftarrow \textsc{pendingTargets}(d,\ T,\ stack)$
6: $\quad\quad EO[d] \leftarrow \textsc{edgeOrdering}(d,\ T')$
7: $\quad$ **return** $\underset{d \in D'}{\operatorname{argmin}}\ EO[d]$

Results

- Implemented WBS in SymTest [1].
- Benchmark programs- RERS challenge, SVCOMP test suite.
- Compared SymTest-WBS with Symtest and KLEE.

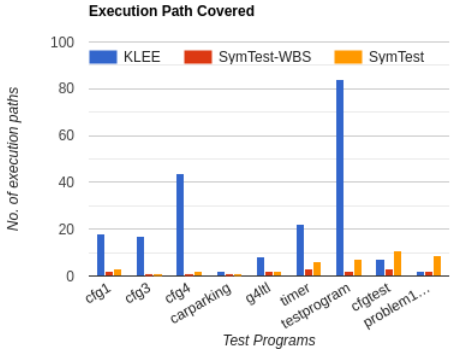


Figure: Result

Conclusion

- We proposed a backtracking heuristic named Weighted Backtracking Strategy to generate short test sequence for embedded software.
- WBS is implemented in SymTest.
- SymTest-WBS requires less number of paths to attain target edge coverage compared to SymTest and KLEE.

References

[1]. https://github.com/sujitkc/symtest.

[2]Sujit Chakrabarti and Ramesh S. "SymTest A Framework for Symbolic Testing of Embedded Software". In: *SymTest*. ISEC, 2016.

[3]Sudhakar M. Reddy Irith Pomeranz. "On generating compact test sequences for synchronous sequential circuits". In: *EURO-DAC '95/EURO-VHDL '95: Proceedings of the conference on European design automation*. December 1995.

[4]I. Pomeranz, L.N. Reddy, and S.M. Reddy. "COMPACTEST: A method to generate compact test sets for combinational circuits". In: *1991, Proceedings. International Test Conference*. 1991.

Thank You!!