Toward Optimal MC/DC Test Case Generation

Sangharatna GODBOLEY[†], Joxan JAFFAR[*],
Rasool MAGHAREH[⋆], **Arpita DUTTA**[*]

[*]National University of Singapore, Singapore
{joxan,arpita}@comp.nus.edu.sg
[⋆]Huawei Canada Research Centre, Canada
rasool.maghareh@huawei.com
[†]National Institute of Technology Warangal, India
sanghu@nitw.ac.in

KLEE Workshop - September 2022

Accepted contributions

Accepted Contribution: Technical Track

Sangharatna Godboley, Joxan Jaffar, Rasool Maghareh & Arpita Dutta. "**Toward optimal MC/DC test case generation.**" In Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA), pp. 505-516, 2021, Aarhus, Denmark.
https://dl.acm.org/doi/10.1145/3460319.3464841

Accepted Contribution: Poster Track

Sangharatna Godboley, Joxan Jaffar, Rasool Maghareh & Arpita Dutta. "**Toward optimal MC/DC test case generation.**" In 30th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA), 2021, Aarhus, Denmark.

Artifact Available

Badges obtained: Available, Functional, and Reusable
Sangharatna Godboley, Joxan Jaffar, Rasool Maghareh, & Arpita Dutta. **CUSTOM-Interpolation: ISSTA artifact evaluation.** In 30th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA), Aarhus, Denmark. **Zenodo**: http://doi.org/10.5281/zenodo.4771439

- Website: https://tracer-x.github.io/
- Github: https://github.com/tracer-x/

Outline

What is MC/DC?

Modified Condition/Decision Coverage (MC/DC)

MC/DC is the second strongest coverage criterion for unit testing. It requires linear number of test cases wrt. the number of atomic conditions present in the program. MC/DC requires all the following requirements [1]:

- Each entry and exit point should get invoked.
- Each predicate takes both possible truth values.
- Each atomic condition(AC) in a predicate takes both possible truth values.
- Each AC in a predicate shown as independent.

Why MC/DC?

According to RTCA standard of DO-178B/C[1], it is mandatory to achieve MC/DC for Level A certificate of safety critical application.

Outline

A Survey with Industrial Practitioners

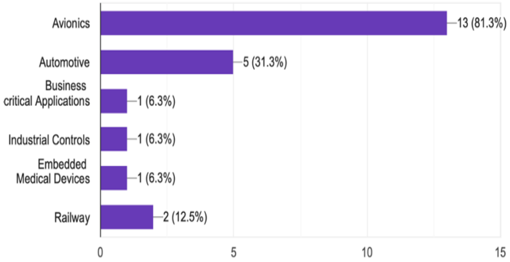

Figure: Which Domain?

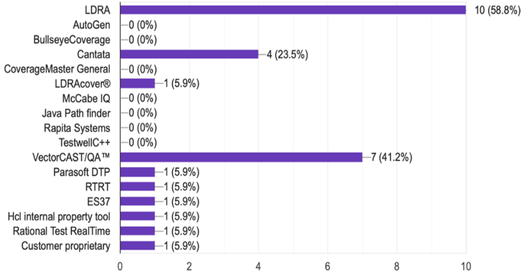

Figure: Which automatic tools?

A Survey with Industrial Practitioners

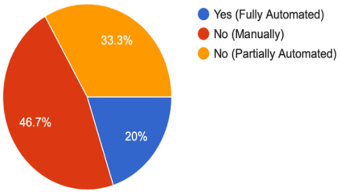

Figure: Automatic vs Manual!

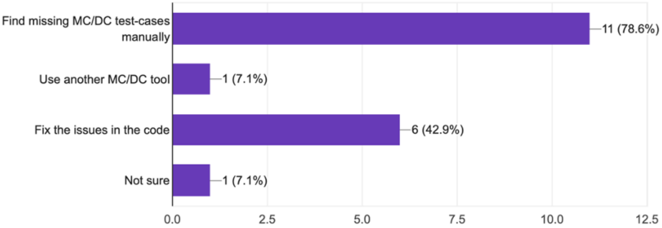

Figure: Which strategy?

Outline

Proposed Idea

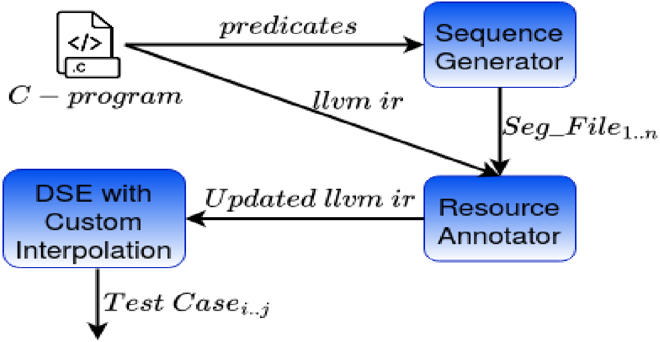

Figure: The Overall Architecture of our Framework

Sequence Generator (SG)

An Example Predicate: if ((b<0 || c<0) && d < 0) { ... }

SG generates five short-circuited sequences $\{S_1 : 101\}$, $\{S_2 : 102\}$, $\{S_3 : 211\}$, $\{S_4 : 212\}$, and $\{S_5 : 220\}$ where 1, 2, 0 represent True, False, and Don't Care for the atomic conditions.

Table: Short-circuit Truth Table for Predicate

	B	C	D	Output
S_1	T	X	T	T
S_2	T	X	F	F
S_3	F	T	T	T
S_4	F	T	F	F
S_5	F	F	X	F

Resource Annotator

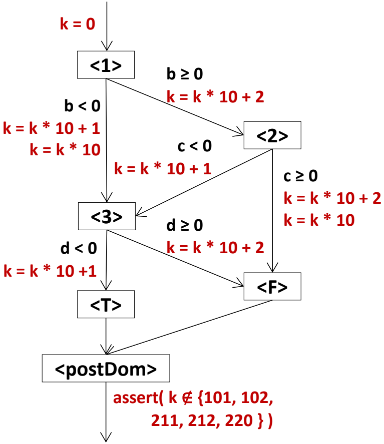

Figure: Annotated CFG for the Predicate

Table: Paths from sequences

Id	Seq	Path
S_1	101	$1 \to 3 \to T$
S_2	102	$1 \to 3 \to F$
S_3	211	$1 \to 2 \to 3 \to T$
S_4	212	$1 \to 2 \to 3 \to F$
S_5	220	$1 \to 2 \to F$

Employing DSE

Algorithm 1 Test Case Generation Using DSE

Input: Upd_LLVM_IR,

Output: Test_Suite

```
Test_Suite ← ∅
errorPaths ← Run_DSE (Upd_LLVM_IR)
for each errorPath in errorPaths do
    Input_Values ← extractInputValues(errorPath)
    Test_Suite ← Test_Suite + Input_Values
end for
```

Taming the Path Explosion

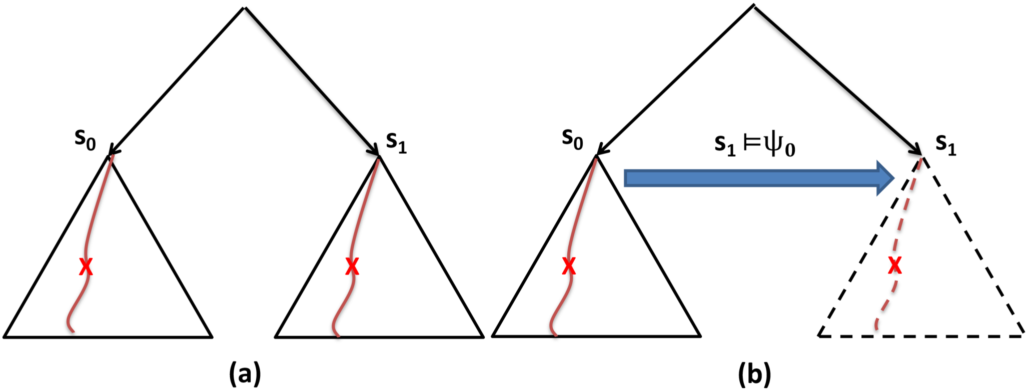

Figure: Exploration of Symbolic Execution Tree in Non-pruning DSE vs. Pruning DSE

Consider the program:

```
x = 0;
if (b1) x += 12;
if (b2) x += 15;
assert(x != 28);
```

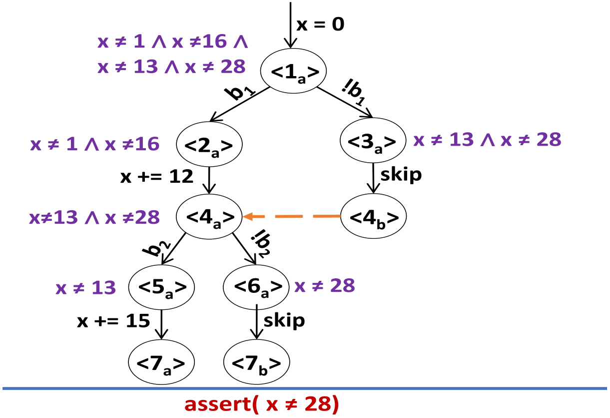

Figure: SET of the program

Custom Interpolation on top of TracerX [5, 7]

Algorithm 2 Custom Interpolation

function PRE(Annotation , ChildInt)
    if BASE_BUG **THEN RETURN** $(\kappa \notin SeqVals - \{Val_\kappa\})$
    END IF
    IF BASE_NO_BUG **THEN RETURN** $(\kappa \notin SeqVals)$
    END IF
    CHILDINT $\equiv (\kappa \notin Set)$
    $ParentSet \leftarrow \{\}$
    FOR EACH s IN Set **DO**
        $ParentSet \leftarrow ParentSet +$ PRECOND(s , ANNOTATION)
    END FOR
    $ParentSet \leftarrow$ REMOVENONINTEGRALS($ParentSet$)
    RETURN $(\kappa \notin ParentSet)$
END FUNCTION
FUNCTION JOIN(PATHINT_1 , PATHINT_2)
    PATHINT_1 $\equiv (\kappa \notin Set\_1)$
    PATHINT_2 $\equiv (\kappa \notin Set\_2)$
    RETURN $(\kappa \notin Set\_1 \cup Set\_2)$
END FUNCTION

Custom Interpolation

Example

$if(a < 0)b = 3;$

$if((b < 0||c < 0)\&\&d < 0)\ \{...\}$

Figure: The Main Example Program

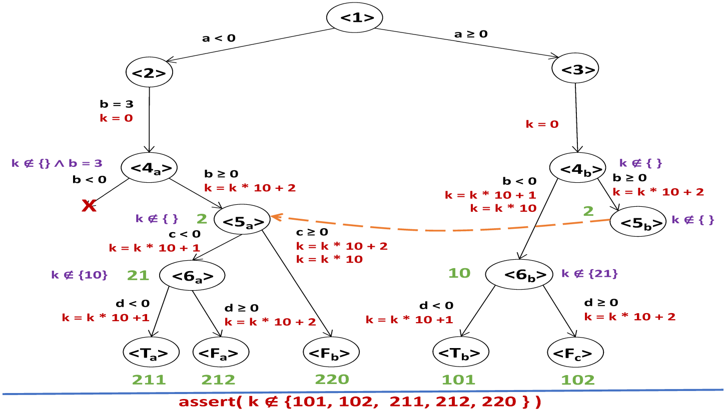

Figure: The Execution Tree of the Main Example Program

Outline

Experimental Setup

Used Setup:

- We experimented on Intel Core i7-6700 3.40 GHz Linux Box with 32GB RAM, and a timeout of 3600 seconds.
- The raw experimental results can be accessed at [28].

Experimental Evaluation:

1. Main Experiment
   - Our Method (CUSTOM) v/s CBMC
2. Supplementary Experiment
   - No Interpolation (KLEE) and Standard Interpolation v/s CUSTOM Interpolation

Used Data set:

Table: Programs Experimented.

Type	psyco	RERS(12-20)	RERS(19-Industry)	zodiac	**Total**
Numbers	14	181	14	1	**210**

KLEE v/s Standard Interpolation v/s CUSTOM Interpolation

No Interpolation (KLEE) v/s Standard Interpolation (TracerX)

- Forward Symbolic Execution to find feasible paths (Similar to KLEE).
- Intermediate execution states preserved (Unlike KLEE).
- Half interpolants are generated by backward tracking and Full interpolants generated by merging half interpolants.
- Full interpolants used for subsumption at similar program points.

Standard Interpolation (TracerX) v/s CUSTOM Interpolation (Paper's Contribution)

- CUSTOM is designed to discover the MC/DC sequences and generate test cases for those sequences unlike TracerX which is used only in case of safety.
- Symbolic execution typically stops the path on witnessing a bug. In contrast, CUSTOM modifies the interpolant and continue the path.
- In CUSTOM, we generate a weakest precondition (WP) interpolant on the ghost variable (κ) alongside the standard interpolant on the rest of the variables.

Main Experiment Results

Groups

1 Both CUSTOM and CBMC terminate.
2 CUSTOM terminates, but CBMC does not
3 CBMC terminates, but CUSTOM does not
4 Neither of the tools terminate

Table: Experimented Programs

Groups	Group1	Group2	Group3	Group4	**#Total**
#Programs	91	71	5	43	**210**

Table: Main Results (Total 710.4K sequences)

Tool	Proved Sequences (Feasible + Infeasible)	UnProven Sequences	Optimal Programs
CBMC	104.7K	605.7K	96/210 (45.71%)
CUSTOM	531.2K	179.2K	**162/210 (77.14%)**

Main Experiment Results

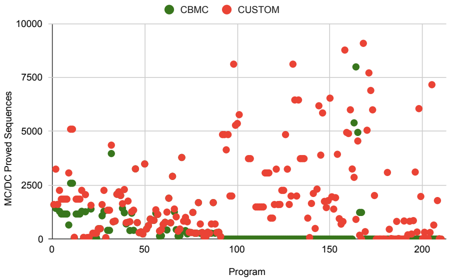

Figure: Scatter chart for MC/DC Proved Sequences

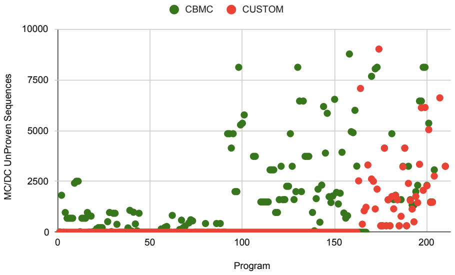

Figure: Scatter chart for MC/DC UnProven Sequences

Supplementary Experimental Results

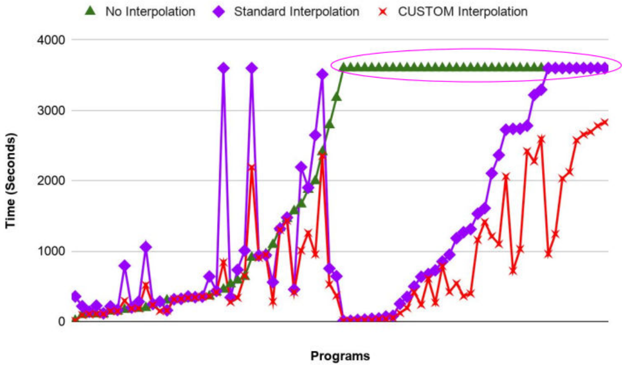

Figure: Comparison of Execution Time in No Interpolation (KLEE) [2], Standard Interpolation [3] vs. **CUSTOM Interpolation**

Major Takeaway

Symbolic Execution(SE) is designed to perform either of these two:

1. Bug Finding
2. Program Verification

In contrast, we used it to discover MC/DC sequences and generate MC/DC specific test cases.

- Our CUSTOM interpolation technique is clever enough to prune the sub trees which contain already discovered MC/DC sequences.
- Our algorithm, if it terminates, generates an optimal set of MC/DC test cases.

Outline

Conclusion

- We have surveyed and found that in industrial practice, automatic MC/DC test generation is woefully inadequate and most practitioners rely on manual effort.
- Our algorithm, if terminates, generates an optimal set of MC/DC test cases.
- We compared CUSTOM against CBMC, the only practical method available which address large programs.
- A comprehensive experimental evaluation shows our implementation to perform at a higher level.

1. Kelly J., Hayhurst and Dan S., Veerhusen and John J., Chilenski and Leanna K., Rierson. A Practical Tutorial on Modified Condition/Decision Coverage. *NASA Langley Technical Report Server* (2001).
2. Cadar, C., Dunbar, D. and Engler, D.R., 2008, December. KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs. In OSDI (Vol. 8, pp. 209-224).
3. Jaffar J., Murali V., Navas J.A., Santosa A.E. (2012) TRACER: A Symbolic Execution Tool for Verification. In: Madhusudan P., Seshia S.A. (eds) Computer Aided Verification. CAV 2012. Lecture Notes in Computer Science, vol 7358. Springer, Berlin, Heidelberg
4. Kroening D., Tautschnig M. (2014) CBMC-C Bounded Model Checker. In: Abraham E., Havelund K. (eds) Tools and Algorithms for the Construction and Analysis of Systems. TACAS 2014. Lecture Notes in Computer Science, vol 8413. Springer, Berlin, Heidelberg
5. Jaffar J., Maghareh R., Godboley S., Ha XL. (2020) TracerX: Dynamic Symbolic Execution with Interpolation (Competition Contribution). In: Wehrheim H., Cabot J. (eds) *Fundamental Approaches to Software Engineering. FASE 2020*. Lecture Notes in Computer Science, vol 12076. Springer, Cham

6. Jaffar J., Godboley S., and Maghareh R. (2019). Optimal MC/DC test case generation. In Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings (ICSE'19). IEEE Press, 288-289. DOI:https://doi.org/10.1109/ICSE-Companion.2019.00118
7. Jaffar J., Maghareh R., Godboley S., Ha XL. (2020) TracerX: Dynamic Symbolic Execution with Interpolation. KLEE 2020 (2nd International KLEE Workshop on Symbolic Execution) Imperial College London, South Kensington Campus
8. SV-COMP Benchmarks: Verification Tasks, https://github.com/sosy-lab/sv-benchmarks/tree/master/c/psyco, Dec 2017
9. RERS: http://rers-challenge.org/, Jun, 2018
10. Artifact Workbook for CUSTOM-Interpolation, https://doi.org/10.6084/m9.figshare.13650242.v1