The TRACER-X System

Joxan JAFFAR & Rasool MAGHAREH

Department of Computer Science, National University of Singapore
*{joxan,rasool}@comp.nus.edu.sg

April 2018
(*KLEE Workshop 2018*)

TRACER-X

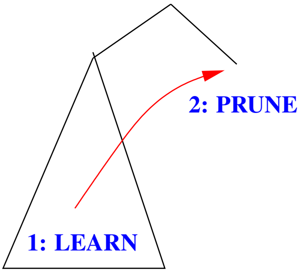
- Introducing *TRACER-X* symbolic execution approach
- Based on the KLEE symbolic virtual machine
- *Interpolation* for search-space reduction
- TRACER-X
- Website: `http://www.comp.nus.edu.sg/~tracerx`
- Github: `https://github.com/tracer-x/`

Outline

1. Mitigating Search-Space Complexity with Interpolation
2. TRACER-X (KLEE with Interpolation)
3. Weakest Precondition Interpolation
4. Memory Bounds Interpolation
5. Symbolic Heap
6. Results & Current Directions

Problem and Solution

- Naive analysis/verification (e.g., standard model checking) $\rightarrow$ huge search space: exponential in the size of the program
- To mitigate the problem we employ *learning*



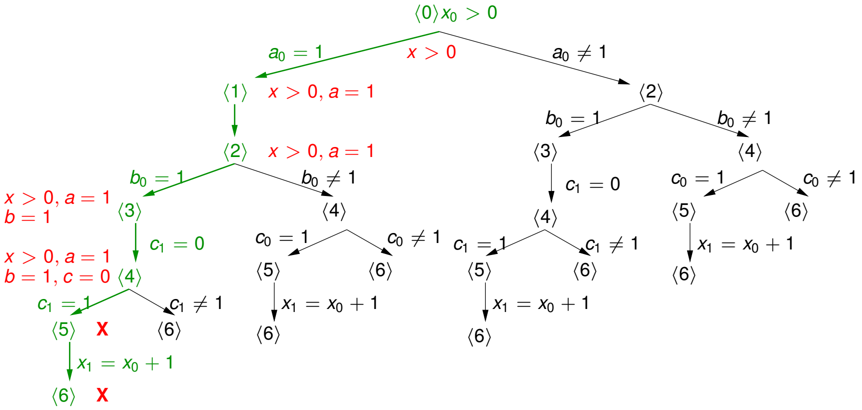
We use information from already traversed (symbolic execution) subtree to prune other subtrees

Example: Proving Safety

Initially $x > 0$
$\langle 0 \rangle$ **if** $(a = 1)$ **then** $\langle 1 \rangle$ **skip endif**
$\langle 2 \rangle$ **if** $(b = 1)$ **then** $\langle 3 \rangle$ $c := 0$ **endif**
$\langle 4 \rangle$ **if** $(c = 1)$ **then** $\langle 5 \rangle$ $x := x + 1$ **endif**
$\langle 6 \rangle$ *assert*$(x > 0)$

Next: The Tree

Symbolic Execution Tree



Constraints with versioned variables for a path in the tree:

$$x_0 > 0 \ \langle 0 \rangle \ a_0 = 1 \ \langle 1 \rangle \ \langle 2 \rangle \ b_0 = 1 \ \langle 3 \rangle \ c_1 = 0 \ \langle 4 \rangle$$
$$c_1 = 1 \ \langle 5 \rangle \ x_1 = x_0 + 1 \ \langle 6 \rangle$$

Interpolation

- HALF Interpolant
  Path-based “weakest precondition”
  (Often easy to compute)
- FULL Interpolant
  Combine half interpolants to become Tree-based
  (Challenge is to obtain compact representation)

Example of the Most Basic Interpolation Method: UNSAT-CORE

$$x_0 > 0 \ \langle 0 \rangle \ a_0 = 1 \ \langle 1 \rangle \ \langle 2 \rangle \ b_0 = 1 \ \langle 3 \rangle \ c_1 = 0 \ \langle 4 \rangle$$
$$c_1 = 1 \ \langle 5 \rangle \ x_1 = x_0 + 1 \ \langle 6 \rangle$$

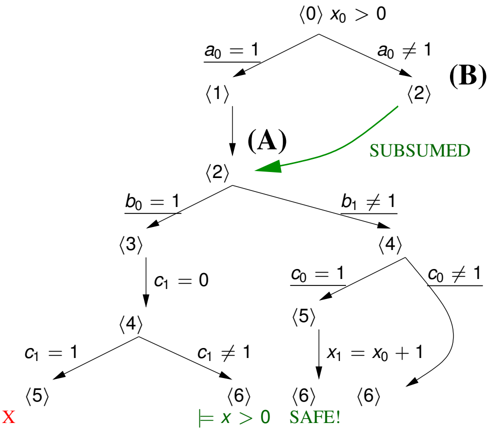
The above constraints are *unsatisfiable*, remove constraints that are not needed to ensure *unsatisfiability*

$$\langle 0 \rangle \ \langle 1 \rangle \ \langle 2 \rangle \ \langle 3 \rangle \ c_1 = 0 \ \langle 4 \rangle \ c_1 = 1 \ \langle 5 \rangle \ \langle 6 \rangle$$

Example: Proving Safety

Initially $x > 0$

$\langle 0 \rangle$ **if** $(a = 1)$ **then** $\langle 1 \rangle$ **skip endif**
$\langle 2 \rangle$ **if** $(b = 1)$ **then** $\langle 3 \rangle$ $c := 0$ **endif**
$\langle 4 \rangle$ **if** $(c = 1)$ **then** $\langle 5 \rangle$ $x := x + 1$ **endif**
$\langle 6 \rangle$ $assert(x > 0)$

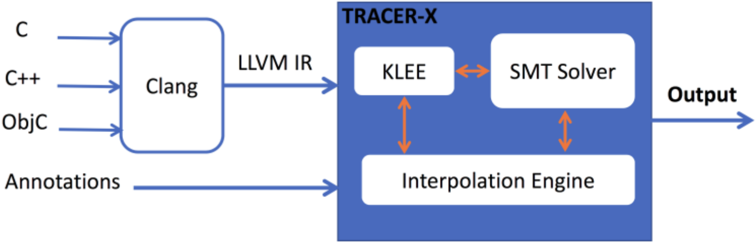


- DFS traversal.
- **W/o interpolation:** The full tree is traversed.
- **W/ interpolation:** (A) is $x > 0$, (B) is $x > 0, a \neq 1$, hence (B) is subsumed by (A), big subtree traversal is avoided.

From KLEE TO TRACER-X

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

Figure: Tracer-X Framework



Weakest Precondition VS Strongest Postcondition

- WP
  Goal-directed and often small formula, per path.
  Unfortunately, not easy to compress individual path WP.
  Biggest disadvantage: agnostic to context.
  (eg: Example above, if x had initial value.)

- SP
  Not goal-directed and often large formula, for all paths.
  Per path reasoning is precise.

- SP with Interpolation
  Can exploit learning from the unsat-core: basic interpolation.
  A remaining disadvantage: interpolation needs to infer new information beyond unsat-core.

Interpolation: Weakest Precondition

- Ideal interpolant is the weakest precondition (WP) of the target
- Unfortunately, WP is intractable to compute

```
x = 0;
if (b1) x += 3 else x += 2
if (b2) x += 5 else x += 7
if (b3) x += 9 else x += 14
{x < 24}
```

Assume $(b1 \wedge \neg b2 \wedge \neg b3)$ is UNSAT.
WP is:
$b1 \longrightarrow (\neg b2 \wedge b3 \wedge x \leq 7) \vee (b2 \wedge x \leq 4)$
$\neg b1 \longrightarrow x < 3$

- Essentially, WP is exponentially disjunctive

Weak-ER Precondition of TRACER-X

First the Easy Cases:

suppose a context of $\tilde{c}$.

- $\mathtt{WP}(t, \omega) = \cdots$ LLVM inverse transition of t
- $\mathtt{WP}(\mathit{assume}(b), \omega) = \omega \wedge b$
- $\mathtt{WP}(\mathtt{if\ (b)\ then\ S1\ else\ S2,}\ \omega) = \omega \wedge b$ where $\tilde{c} \models b$
- Similarly for when $\tilde{c} \models \neg b$

Weak-ER Precondition of TRACER-X

The General Case:

`if (b) then S1 else S2` with postcondition ω where

- the context is $\tilde{c} = c_1, c_2, \cdots, c_n$.
- Neither $\tilde{c} \models b$ nor $\tilde{c} \models \neg b$ holds.
- $wpp(S1, \omega)$ is ω_1 and $wpp(S2, \omega)$ is ω_2

In general, the weakest precondition Ψ is a disjunction:

$$(b \longrightarrow \omega_1) \wedge (\neg b \longrightarrow \omega_2)$$

We want to compute a convex Φ. (Therefore $\tilde{c} \models \Phi \models \Psi$)

Takeaway:

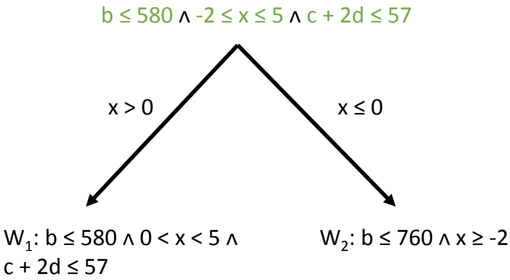
- There is no succinct definition for this convex.
- The above examples show, however, that there are many special cases to exploit.

WP Interpolation Example 1

1. Choose a candidate to generalize: $c = 2 \wedge d = 4$
2. Extract the subset of W_1 and W_2 which share the same variables with $c = 2 \wedge d = 4$:
   - Subset of W_1: $c + 2d \leq 57$
   - Subset of W_2: $\{\}$
3. If one subset is empty, generalize the candidate to the other subset: $c + 2d \leq 57$.

Original Context:

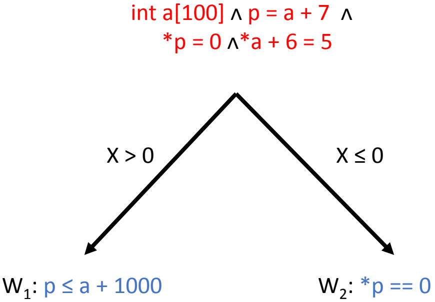
$a > 0 \wedge b = 5 \wedge -1 \leq x \leq 1 \wedge c = 2 \wedge d = 4$



WP Interpolation Example 2 (pointers and elements)

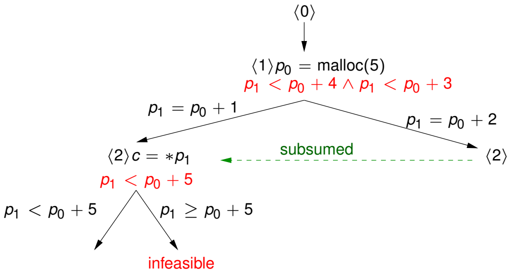
- When generalizing, arrays candidates should be chosen and generalized carefully:
- Candidate: $int\ a[100] \wedge p = a + 7 \wedge *p = 0 \wedge *a + 6 = 5$
- Generalization:

  $int\ a[100] \wedge p \leq a + 100 \wedge *p == 0 \wedge *a + 6 = 5$



Note that the generalization of *p does not include* $p = a + 6$.

$\langle 0 \rangle$ $p = \texttt{malloc}(5)$
$\langle 1 \rangle$ **if** $(\ldots)$ **then**
  $p{+}{+}$
else
  $p{+}{=}2$
endif
$\langle 2 \rangle$ $c := *p$

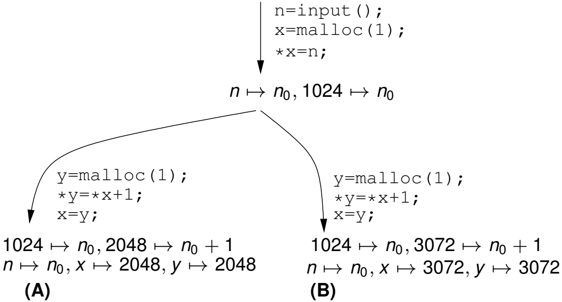


Interpolation: Symbolic Heap

```
#define MAX 18
n = input(); // getting a symbolic input
x = malloc(1); *x = n;
for (int i = 0; i < MAX; i++) {
    if (*) { y = malloc(1); *y = *x+1; }
    else { y = malloc(1); *y = *x+1; }
    x = y;
}
```

- `malloc()` is nondeterministic, but enjoys separation
- Branches (essentially) identical
- Times out using KLEE and LLBMC (30 mins)
- Exponential running time for both KLEE and LLBMC (and potentially Veritesting)

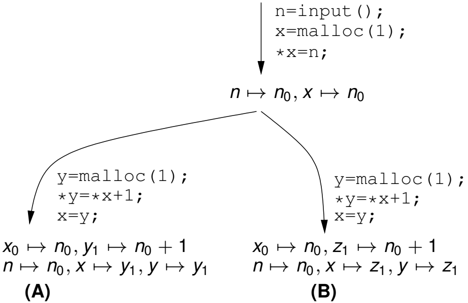
Is (B) Subsumed by (A)?



In dynamic symbolic execution and even LLBMC, different concrete values are returned by each `malloc` call (satisfies separation)

$\rightarrow$ both states cannot be matched

Is (B) Subsumed by (A)?



Our approach:

- We regard dynamically-allocated addresses symbolically: $1024 = x_0$, $2048 = y_1$, $3072 = z_1$.
- Matching: $(y_1, z_1) \rightarrow$ subsumption holds!

Symbolic Heap Interpolation of TRACER-X

$$\exists z_1 . \left(\begin{array}{l} x_0 \mapsto n_0 \wedge z_1 \mapsto n_0 + 1 \wedge \\ n \mapsto n_0 \wedge x \mapsto z_1 \wedge y \mapsto z_1 \end{array} \right) \models$$
$$\exists y_1 . \left(\begin{array}{l} x_0 \mapsto n_0 \wedge y_1 \mapsto n_0 + 1 \wedge \\ n \mapsto n_0 \wedge x \mapsto y_1 \wedge y \mapsto y_1 \end{array} \right)$$

Existentials: z_1, y_1 : some addresses dynamically allocated

Problem: Prove subsumption by eliminating existentials

→ SMT solvers are weak in solving quantified formulas

- General problem is NP-Complete or harder (*conjecture*)
- Must use specialized quantifier elimination techniques

$$\left(\begin{array}{l} x_0 \mapsto n_0 \wedge z_1 \mapsto n_0 + 1 \wedge \\ n \mapsto n_0 \wedge x \mapsto z_1 \wedge y \mapsto z_1 \end{array}\right) \models \exists y_1 . \left(\begin{array}{l} x_0 \mapsto n_0 \wedge y_1 \mapsto n_0 + 1 \wedge \\ n \mapsto n_0 \wedge x \mapsto y_1 \wedge y \mapsto y_1 \end{array}\right)$$

$$\left(\begin{array}{l} x_0 \mapsto n_0 \wedge z_1 \mapsto n_0 + 1 \wedge \\ n \mapsto n_0 \wedge x \mapsto z_1 \wedge y \mapsto z_1 \end{array}\right) \models \left(\begin{array}{l} x_0 \mapsto n_0 \wedge z_1 \mapsto n_0 + 1 \wedge \\ n \mapsto n_0 \wedge x \mapsto z_1 \wedge y \mapsto z_1 \end{array}\right)$$

Procedure:

1. Unquantify antecedent variables: z_1 becomes free
2. Elimination done by traversal from global/local variables and finding matching substitutions that would work. In our case, $[z_1/y_1]$, replacing existentially-quantified y_1 with free z_1.
3. Solve subsumption using SMT solver via entailment without quantification.
4. In general, compute data structure homomorphisms for quantifier elimination
(In general, intractable, but often easy.)

COREUTILS Results 1 (Complete Runs)

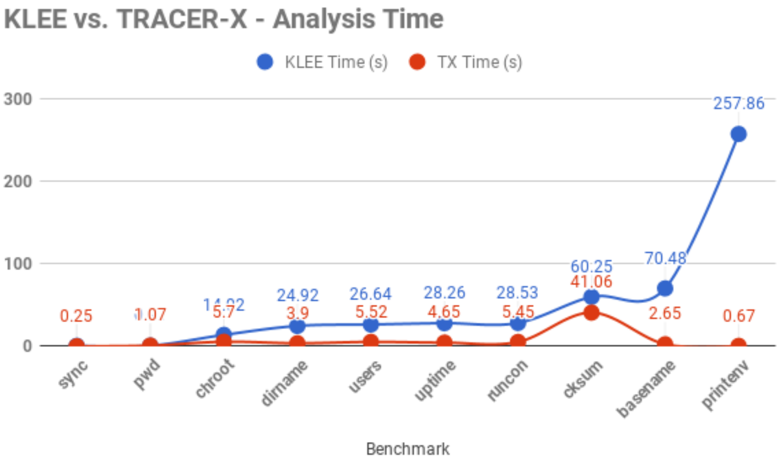


Figure: (Both TRACER-X and KLEE Finish Execution)

COREUTILS Results 2 (Complete Runs)

Table: (TRACER-X Finishes Execution but KLEE does not Finish)

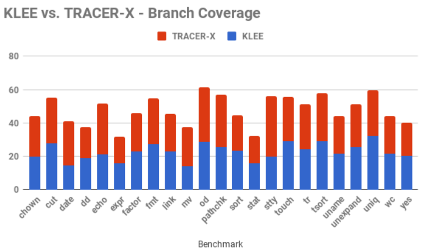
Benchmark	LOC	KLEE (TIMEOUT: 3600 S)			TRACER-X				
		#ERR	#EP	#PEP	T (s)	#ERR	#EP	#SP	#PEP
base64	**401**	0	0	2115667	3327.7	0	0	294256	20551
cat	**339**	1	2761	2729703	1824.5	1	45546	127526	14128
chcon	**604**	0	0	1663596	1927.5	0	0	221628	15722
chgrp	**612**	0	0	778227	2461.7	0	0	150836	34330
comm	**255**	0	0	1860052	2748.1	0	0	67496	39677
df	**547**	2	2108	13114	900.7	2	1456	1531	2321
dircolors	**241**	0	0	1824002	3366.0	0	0	115004	7883
env	**286**	0	0	1846675	63.1	0	0	5078	3124
fold	**98**	0	0	1959113	1292.3	0	0	46899	49494
head	**482**	1	4	1950422	2438.4	1	3	6323	3375
hostid	**175**	0	0	2107218	3494.4	1	1	332198	19060
hostname	**180**	0	0	2323263	968.4	0	0	116020	7876
ln	**497**	0	0	578519	3064.2	0	0	145226	16363
logname	**181**	0	0	2125296	3018.7	0	0	315266	18633
mkdir	**237**	0	0	902244	1964.6	0	0	53072	35288
mkfifo	**206**	0	0	906846	1930.6	1	4	52775	35516
mknod	**597**	2	155	2519413	3300.7	2	2	243958	15422
mktemp	**650**	0	0	2448222	3131.1	0	0	278334	9539
nice	**238**	0	0	2372168	215.4	0	0	16973	1963

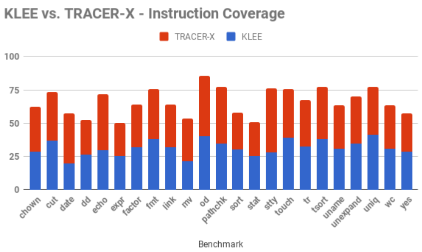
COREUTILS Results 3 (Complete Runs)

Table: (TRACER-X Finishes Execution but KLEE does not Finish)

Benchmark	LOC	KLEE (TIMEOUT: 3600 S)			TRACER-X				
		#ERR	#EP	#PEP	Time (s)	#ERR	#EP	#SP	#PEP
nl	**1293**	0	0	2039298	1527.9	0	0	15508	37601
nohup	**209**	0	0	832849	2351.5	0	0	86765	26321
paste	**135**	0	0	1760657	754.7	0	0	1392	40251
pinky	**514**	0	0	461789	445.9	0	0	4609	1255
pr	**598**	0	0	528712	900.9	0	0	1241	7436
printf	**553**	0	0	2065362	2736.7	0	0	177617	22781
readlink	**301**	0	0	3076444	8.8	0	0	1925	244
rm	**656**	0	0	1330799	1341.9	0	0	33313	30590
rmdir	**180**	0	0	765438	2174.1	0	0	43524	39515
setuidgid	**290**	0	0	1124960	1578.9	0	0	38662	14778
shred	**472**	0	0	22206	397.1	0	0	0	27
sleep	**204**	0	0	926939	1778.6	0	0	61065	41583
tee	**88**	0	0	2000807	67.9	0	0	1058	2989
tty	**176**	0	0	1564733	1160.6	0	0	1116	22012
unlink	**177**	0	0	1596777	970.7	0	0	115903	7899
who	**749**	0	0	2903432	909.3	0	0	1211	21137
whoami	**183**	0	0	2106260	3164.8	0	0	325344	18620

COREUTILS Results 4 (INCOMPLETE Runs)





Note our good performance on coverage.

Current Directions: Testing

- Modified Condition/Decision Coverage (MC/DC): A minimal set of test-cases needed to ensure the safety

- DSE-based approaches: Unguided search for test-cases
- Cannot prove test-case non-existence (not fully traversed SET)

- TRACER-X Approach:
- Guided search to find a path reaching a target test-case
- Proving non-existence of a test-case if not found in the end of search

Current Directions: Incremental Quantitative Analysis

- Quantitative Analysis: Ensure safety of non-functional features in embedded systems and IoT

- Exact Methods: Not Scalable
- Abstraction-based methods: Scalable but Inaccurate

- TRACER-X Approach:
- Given an upper and lower-bound check the mid-point
- If safe: Decrease the upper-bound to the mid-point
- If counter-example found: Increase the lower-bound (unavailable for abstraction-based analyses)

- progressively increasing certified accuracy
- Stop-any-time
- Dynamic Resource Cost Model

Current Directions: Combinatorial Optimization (COP)

- COP is widely applicable in AI
- A good solution is usually good enough

- Traditional methods: Mathematical Programming & Constraint Programming

- TRACER-X Approach:
- Run TRACER-X on a program that check a given solution
- Maintain lower and upper-bounds (similar to Quantitative Analysis)
- Use Interpolation and Symmetry to prune

- progressively walking towards optimal solution
- Stop-any-time

Conclusion

- TRACER-X:
- Website: `http://www.comp.nus.edu.sg/~tracerx`
- Github: `https://github.com/tracer-x/`
- (with Unsat-Core & some Weakest-Precondition interpolation)