Symstra

A Framework for Generating Object-Oriented Unit Tests using Symbolic Execution

Tobias Kaiser, Délia Cheminot, May 2024

UNIT TESTS

- Unit tests test classes.
- Sequence of method invocations with arguments
- Branch coverage, intra-method path coverage
- Concrete representation mostly used

EXISTING TOOLS

Random

Repeating sequences, not covering

Concrete states

User has to choose domains

Symbolic execution

No method sequences

SYMSTRA

- Symbolic sequence exploration
- Symbolic state comparison
- Show a real implementation
- Faster generation and better branch coverage

TODAY'S PLAN

01
Proposed solution
How Symstra works

02
Evaluation
Does it work well?

03
Discussion
Future works

01

Proposed solution

SYMBOLIC EXPRESSIONS

x

Variables

Each symbolic variable has a corresponding type

c

Constants

A Java constant of type T is a symbolic expression of type T

o

Operators

Symbolic expressions connected with an operator are also symbolic expressions

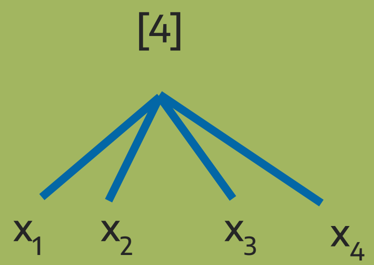
SYMBOLIC STATE

- Symbolic expr. rather than concrete values
- Pair of constraints and heap *{C, H}*
- Heap is viewed as a graph
  - Nodes represent Objects
  - Edges represent Object Fields

SYMBOLIC STATE

Example:

- Constraint: $x_1 = x_2$ && $x_3 < x_4$
- Heap:



HEAP ISOMORPHISM

- Represent identical method behaviors
- Renaming $\tau : V \to V$ extended as $\tau : U \to U$
- $\tau(p) = p$ for all $p \in P$, $\tau(\odot u1,\ldots,un) = \odot\tau(u1),\ldots,\tau(un)$ for all $u1, \ldots, un \in U$ and operations $\odot$.

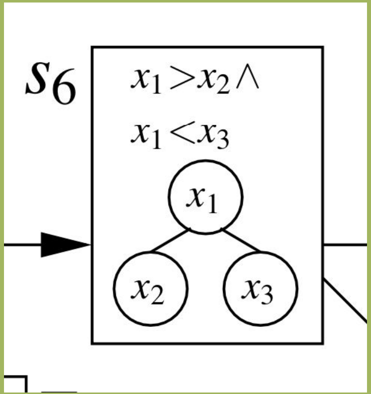
HEAP ISOMORPHISM

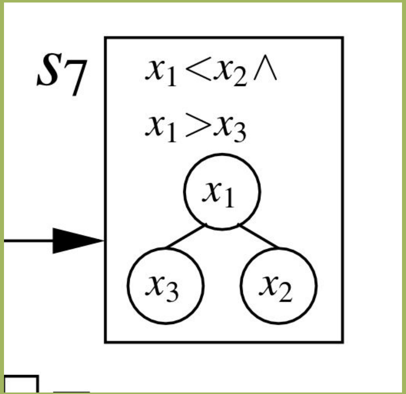
Two heaps $\langle O1, E1 \rangle$ and $\langle O2, E2 \rangle$ are isomorphic iff there are bijections $\rho : O1 \to O2$ and $\tau : V \to V$ such that:

$$
\begin{aligned}
E2 = &\{\langle \rho(o), f, \rho(o') \rangle \mid \langle o, f, o' \rangle \in E1, o' \in O1\} \\
&\cup \{\langle \rho(o), f, null \rangle \mid \langle o, f, null \rangle \in E1\} \\
&\cup \{\langle \rho(o), f, \tau(o') \rangle \mid \langle o, f, o' \rangle \in E1, o' \in U\}.
\end{aligned}
$$

HEAP ISOMORPHISM

“Two isomorphic heaps have the same fields for all objects and equal (up to renaming) symbolic expressions for all primitive fields.”





HEAP ISOMORPHISM

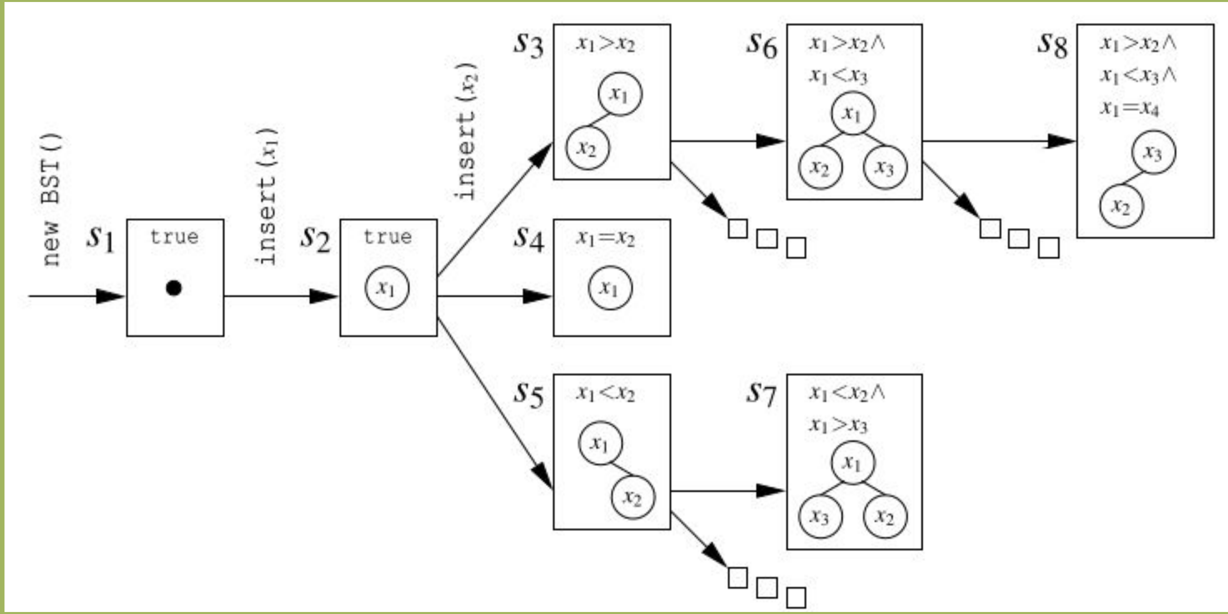
- Rooted heaps: fields reachable from an object
- Can be efficiently checked
- States are linearized
- Depth-first traversal

STATE SUBSUMPTION

- When prune the exploration of a branch?
- Instantiate symbolic heaps

State $\{C_1, H_1\}$ subsumes $\{C_2, H_2\}$ iff for every concrete heap H'_2 there exists a concrete heap H^*_1 so that H^*_1 and H^*_2 are isomorphic

STATE SUBSUMPTION



SYMBOLIC EXECUTION

- $\sigma_m(\langle C, H\rangle)$: set of states that the symbolic execution, σ, of the method m produces starting from the state ⟨C, H⟩
- Both branches of conditional statements explored
- Path condition

SYMBOLIC EXECUTION

- $\sigma_m(\langle C, H \rangle)$ potentially infinite
- Code re-executed from beginning
- No intermediate states
- Standard symbolic execution optimizations

STATE EXPLORATION

- State space: *All states reachable by executing all possible method sequences*
- Sequences have to start with constructor methods

→ State space is infinite

STATE EXPLORATION

- Symstra uses breadth-first-search
- Input: Set of methods and bound on length of sequences
- Maintains set of explored states and queue of unexplored

BREADTH-FIRST-SEARCH

- Take unprocessed state, execute every MUT on it
- Check if new states are subsumed by others
- If yes: Prune exploration of that path
- If not, add state to queue

CONCRETE TESTS GENERATION

- Made during states exploration
- Constraint and shortest method sequence
- POOC constraint solver (added as comments)
- Sequences are JUnit test classes

02

Evaluation

TESTED CLASSES

class	methods under test	some private methods	#ncnb lines	# branches
IntStack	push,pop	–	30	9
UBStack	push,pop	–	59	13
BinSearchTree	insert,remove	removeNode	91	34
BinomialHeap	insert,extractMin delete	findMin,merge unionNodes,decrease	309	70
LinkedList	add,remove,removeLast	addBefore	253	12
TreeMap	put,remove	fixAfterIns fixAfterDel,delEntry	370	170
HeapArray	insert,extractMax	heapifyUp,heapifyDown	71	29

COMPARED VALUES

		Symstra				Rostra			
class	N	time	states	tests	%cov	time	states	tests	%cov
UBStack	5	0.95	22	43(5)	92.3	4.98	656	1950(6)	92.3
	6	4.38	30	67(6)	100.0	31.83	3235	13734(7)	100.0
	7	7.20	41	91(6)	100.0	*269.68	*10735	*54176(7)	*100.0
	8	10.64	55	124(6)	100.0	-	-	-	-

- Rostra: System with concrete states
- Asterisk: Time limit of 3 minutes was reached
- Blank lines: Rostra exceeded memory limit

FINDINGS

BinomialHeap	5	1.39	6	40(13)	84.3	4.97	380	1320(12)	84.3
	6	2.55	7	66(13)	84.3	50.92	3036	12168(12)	84.3
	7	3.80	8	86(15)	90.0	-	-	-	-
	8	8.85	9	157(16)	91.4	-	-	-	-

- Symstra generates sequences much faster
- Achieves Branch coverage in less time

FINDINGS

HeapArray	5	1.36	14	36(9)	75.9	3.75	664	1296(10)	75.9
	6	2.59	20	65(11)	89.7	-	-	-	-
	7	4.78	35	109(13)	100.0	-	-	-	-
	8	11.20	54	220(13)	100.0	-	-	-	-

- Symstra needs less memory
- Rostra often exceeds memory with higher N

COMPLETE RESULTS

class	N	Symstra time	Symstra states	Symstra tests	Symstra %cov	Rostra time	Rostra states	Rostra tests	Rostra %cov
UBStack	5	0.95	22	43(5)	92.3	4.98	656	1950(6)	92.3
	6	4.38	30	67(6)	100.0	31.83	3235	13734(7)	100.0
	7	7.20	41	91(6)	100.0	*269.68	*10735	*54176(7)	*100.0
	8	10.64	55	124(6)	100.0	-	-	-	-
IntStack	5	0.23	12	18(3)	55.6	12.76	4836	5766(4)	55.6
	6	0.42	16	24(4)	66.7	-	-	-	-
	7	0.50	20	32(5)	88.9	*689.02	*30080	*52480(5)	*66.7
	8	0.62	24	40(6)	100.0	-	-	-	-
BinSearchTree	5	7.06	65	350(15)	97.1	4.80	188	1460(16)	97.1
	6	28.53	197	1274(16)	100.0	23.05	731	7188(17)	100.0
	7	136.82	626	4706(16)	100.0	-	-	-	-
	8	*317.76	*1458	*8696(16)	*100.0	-	-	-	-
BinomialHeap	5	1.39	6	40(13)	84.3	4.97	380	1320(12)	84.3
	6	2.55	7	66(13)	84.3	50.92	3036	12168(12)	84.3
	7	3.80	8	86(15)	90.0	-	-	-	-
	8	8.85	9	157(16)	91.4	-	-	-	-
LinkedList	5	0.56	6	25(5)	100.0	32.61	3906	8591(6)	100.0
	6	0.66	7	33(5)	100.0	*412.00	*9331	*20215(6)	*100.0
	7	0.78	8	42(5)	100.0	-	-	-	-
	8	0.95	9	52(5)	100.0	-	-	-	-
TreeMap	5	3.20	16	114(29)	76.5	3.52	72	560(31)	76.5
	6	7.78	28	260(35)	82.9	12.42	185	2076(37)	82.9
	7	19.45	59	572(37)	84.1	41.89	537	6580(39)	84.1
	8	63.21	111	1486(37)	84.1	-	-	-	-
HeapArray	5	1.36	14	36(9)	75.9	3.75	664	1296(10)	75.9
	6	2.59	20	65(11)	89.7	-	-	-	-
	7	4.78	35	109(13)	100.0	-	-	-	-
	8	11.20	54	220(13)	100.0	-	-	-	-

03

Related Work

DISCUSSION

Specifications

Are post-conditions or invariants violated?

Performance

Union states with disjunction in constraints

Limitations

Array indexes as variables, non-primitive arguments

04

Conclusion

CONCRETE TESTS GENERATION

- Symstra uses symbolic execution to generate method sequences for high branch coverage
- State subsumption based pruning speeds up exploration
- Faster and memory efficient test generation

THANK YOU!