Symbolic Complexity Analysis using Context-preserving Histories

Kasper Luckow, Rody Kersten, Corina Pasareanu

Motivating Example

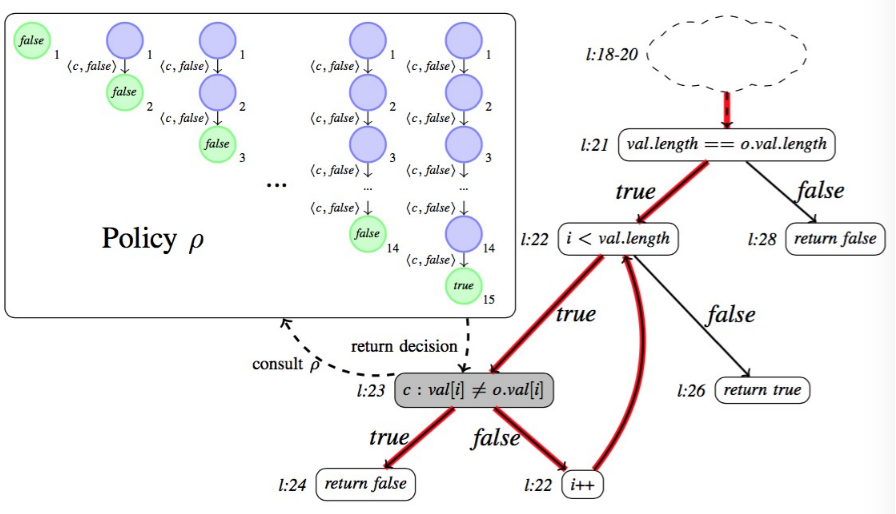
```
class Entry {
  String key; Action val; Entry next;
  public Entry(String key, Action val, Entry next) {
    this.key = key; this.val = val; this.next = next;
  }
}
Entry findEntry(String o, int n) {
  for(Entry e = table[n]; e != null; e = e.next) {
    if(e.key.equals(o)) {
      return e;
    }
  }
  return null;
}
class String {
  char[] val;
  // ...
  public boolean equals(Object oObj) {
    // ...
    String o = (String) oObj;
    if(val.length == o.val.length) {
      for(int i = 0; i < val.length; i++) {
        if(val[i] != o.val[i])
          return false;
      }
      return true;
    }
    return false;
  }
}
```

Motivating Example

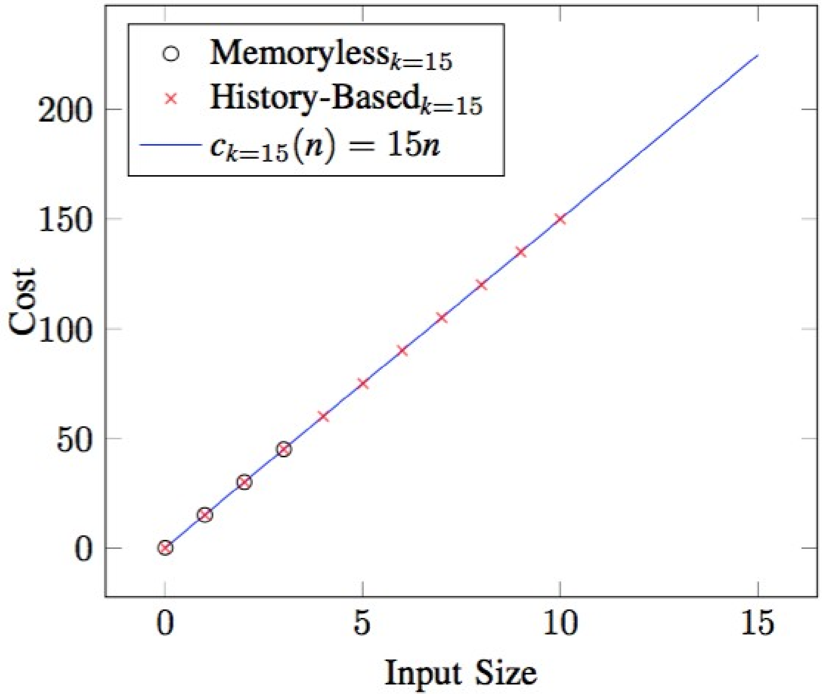
- Worst Case
  - strings: length k; Input size: length n;
  - n*k
- We aim to extract a policy that dictates what branch to take during symbolic execution at any input size

Motivating Example

- We take into account the history of decisions taken along the worst-case path



Motivating Example



Guided Policy

- Policy p: a policy p is a function mapping a CFG branch c to a choice b {{},{true},{false},{true/false}}
- p is deterministic if it contains no {} or {true/false}

Complexity analysis

Procedure 1 Worst-case complexity analysis.

Input: Program $P(n)$, input size bound N, policy gen. input sizes L, H s.t. $L \leq H < N$, cost model $\mathcal{C}$, policy score κ

{`Phase (i)`}

$\rho_{\cup}$ initialized to $\perp$ for all CFG conditions c
for $j \leftarrow L$ **to** H **do**
    $W_e \leftarrow \mathit{exhaustiveWCA}(P(j), \mathcal{C})$
    $\rho_{best} \leftarrow \mathit{null}$
    for all $\langle \phi, \mathcal{C}(\phi), PC \rangle \in W_e$ **do**
        $\rho \leftarrow \mathit{computePolicy}(\phi)$
        if ρ_{best} is *null* **then**
            $\rho_{best} \leftarrow \rho$
        else if $\kappa(\rho) > \kappa(\rho_{best})$ **then**
            $\rho_{best} \leftarrow \rho$
    $\rho_{\cup} \leftarrow \rho_{\cup} \bigcup \rho_{best}$

{`Phase (ii)`}

$D \leftarrow \varnothing$
for $i \leftarrow 1$ **to** N **do**
    $W_g \leftarrow \mathit{guidedSymExe}(P(i), \rho_{\cup}, \mathcal{C})$
    $cost_i \leftarrow \mathcal{C}(\phi)$ s.t. $\langle \phi, \mathcal{C}(\phi), PC \rangle \in W_g$
    $D \leftarrow D \cup \langle i, cost_i \rangle$
$\langle f, r^2 \rangle \leftarrow \mathit{regressionAnalysis}(D)$
Output $\langle f, r^2 \rangle$, input constraints and solutions.
return

Complexity Analysis

Procedure 2 computePolicy

Input: Worst-case path ϕ
Output: Policy ρ

ρ initialized to $\perp$ for all CFG conditions c
for all $\pi_k = \langle c, b, \alpha \rangle$ where $k = 1, ..., length(\phi)$ **do**
    $\rho(c) \leftarrow \rho(c) \cup b$
return ρ

Procedure 3 guidedSymExe

Input: Program P, policy ρ, cost model $\mathcal{C}$
Output: $W = \{\langle \phi, \mathcal{C}(\phi) \rangle_1, ...\}$

Run symbolic execution on P and record worst-case paths in set W
for all $\pi = \langle c, b, \alpha \rangle$ about to be explored **do**
    $choice \leftarrow \rho(c)$
    if $choice \neq \perp$ and $choice \neq \top$ **then**
        Explore $b = choice$ for c in π
    else
        Explore both $b = true$ and $b = false$ for c in π
return W

Procedure 4 κ

Input: Policy ρ
Output: Policy rank

$rank \leftarrow 0$
for all c in $P(n)$ **do**
    $res \leftarrow \rho(c)$
    if $res \neq \perp$ and $res \neq \top$ **then**
        $rank \leftarrow rank + 1$
return $rank$

History-based policy

- Policy updating

For (1), we update line 3 in Procedure 2 as follows:

$$\rho(c, \downarrow(\mathcal{H}_h(\pi_k))) \leftarrow \rho(c, \downarrow(\mathcal{H}_h(\pi_k))) \cup b$$

- Policy guided search

For (2), we update line 3 in Procedure 3 as follows

$$choice \leftarrow \rho(c, \downarrow(\mathcal{H}_h(\pi)))$$

Theoretical guarantee

Theorem 1 *Let $\rho_{\cup}^{L..H} = \bigcup_{n=L..H} \rho_n$ denote the unification of the policies obtained from the analysis at input sizes L..H, for same history size h. Then there exists a sufficiently large M such that the policy $\rho_{\cup}^{L..M}$ accurately predicts the worst-case path for any input size that is greater or equal to L.*

Proof: First observe monotonicity of policy generation. We define $\rho_1 \subseteq \rho_2$ as $\forall_{i \geq L}\{\Phi_{i,\rho_1} \subseteq \Phi_{i,\rho_2}\}$, where $\Phi_{i,\rho}$ is the set of paths explored with policy ρ at input size i. Unification of policies leads to increased coverage of program behaviors: if $\rho_{\cup}^{L..n+1} = \rho_{\cup}^{L..n} \bigcup \rho_{n+1}$ then $\rho_{\cup}^{L..n+1} \supseteq \rho_{\cup}^{L..n}$, since ρ_{n+1} can only add more behaviours that are allowed. Since the number of choices for the policy is finite and the history size is fixed, there is a finite number of possible policies. Hence, there exists an M for which $\rho_{\cup}^{L..M}$ is 'largest' according to $\subseteq$ and thus includes the worst-case path for any input size that greater or equal to L. ∎

Evaluation

Benchm.	Set-up			Input Size (N)												Complexity	r^2
	$L{=}H$	h		**1**	**2**	**3**	**4**	**5**	**10**	**15**	**20**	**30**	**100**	**250**	**1000**		
Blogger URI Verifier	**Exh.**		**Paths**	55	2213	114533	-	-	-	-	-	-	-	-	-		
			Time	0:02	0:25	26:46	-	-	-	-	-	-	-	-	-		
	1	$m.l.$	**Paths**	8	57	155	351	743	-	-	-	-	-	-	-	$\mathcal{O}(n^2)$	0.99986
			Time	0:00	0:02	0:31	4:45	45:09	-	-	-	-	-	-	-		
TextCrunchr ZIP Decompressor	**Exh.**		**Paths**	3	4	5	6	7	12	17	22	32	102	252	1002		
			Time	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:01	0:06	1:27		
	1	$m.l.$	**Paths**	1	1	1	1	1	1	1	1	1	1	1	1	$\mathcal{O}(n)$	1.0000
			Time	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:01	0:06	1:28		
Find Entry ($k = 15$)	**Exh.**		**Paths**	16	376	11656	-	-	-	-	-	-	-	-	-		
			Time	0:01	0:07	4:59	-	-	-	-	-	-	-	-	-		
	2	$m.l.$	**Paths**	16	376	11656	-	-	-	-	-	-	-	-	-	(too few predictors)	
			Time	0:01	0:07	4:09	-	-	-	-	-	-	-	-	-		
	2	14	**Paths**	1	1	1	1	1	1	1	1	1	1	1	-	$\mathcal{O}(n)$	1.0000
			Time	0:01	0:01	0:02	0:02	0:02	0:04	0:08	0:12	0:24	6:00	2:00:32	-		
TextCrunchr NGram Score (trigrams)	**Exh.**		**Paths**	4	13	40	121	364	88573	-	-	-	-	-	-		
			Time	0:00	0:00	0:00	0:01	0:02	3:15	-	-	-	-	-	-		
	2	$m.l.$	**Paths**	4	13	40	121	364	88573	-	-	-	-	-	-	$\mathcal{O}(n)$	1.0000
			Time	0:00	0:00	0:00	0:01	0:02	5:10	-	-	-	-	-	-		
	2	2	**Paths**	1	1	1	1	1	1	1	1	1	1	1	1	$\mathcal{O}(n)$	1.0000
			Time	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:02	0:15	7:07		

Password Checker ($k = 32$)	Exh.		**Paths**	33	1057	33825	-	-	-	-	-	-	-	-	-		
			Time	0:01	0:04	2:00	-	-	-	-	-	-	-	-	-		
	2	*m.l.*	**Paths**	33	1057	33825	-	-	-	-	-	-	-	-	-	(too few predictors)	
			Time	0:00	0:03	2:04	-	-	-	-	-	-	-	-	-		
	2	31	**Paths**	1	1	1	1	1	1	1	1	1	1	-	-	$\mathcal{O}(n)$	1.0000
			Time	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:04	-	-		
LawDB Database B-Tree	Exh.		**Paths**	3	13	75	541	4683	-	-	-	-	-	-	-		
			Time	0:01	0:01	0:01	0:01	0:07	-	-	-	-	-	-	-		
	2	*m.l.*	**Paths**	2	3	4	3	3	4	5	5	6	8	583	-	$\mathcal{O}(\log n)$	0.99755
			Time	0:01	0:01	0:01	0:01	0:01	0:01	0:01	0:01	0:01	0:29	06:23	-		
Sorted Linked-List insert	Exh.		**Paths**	1	2	6	24	120	-	-	-	-	-	-	-		
			Time	0:00	0:00	0:00	0:01	0:01	-	-	-	-	-	-	-		
	3†	*m.l.*	**Paths**	1	1	1	1	1	1	1	1	1	1	-	-	$\mathcal{O}(n)$	1.0000
			Time	0:00	0:01	0:01	0:01	0:01	0:01	0:02	0:04	0:10	58:51	-	-		
Heap insert (JDK 1.5)	Exh.		**Paths**	1	2	4	12	36	20736	-	-	-	-	-	-		
			Time	0:00	0:00	0:00	0:00	0:01	0:46	-	-	-	-	-	-		
	2	*m.l.*	**Paths**	1	1	1	1	1	1	1	1	1	1	1	1	$\mathcal{O}(\log n)$	0.99699
			Time	0:00	0:01	0:01	0:01	0:01	0:01	0:01	0:01	0:01	0:02	0:09	0:53		
Red-Black Tree search	Exh.		**Paths**	3	10	42	216	1320	-	-	-	-	-	-	-		
			Time	0:00	0:00	0:00	0:01	0:03	-	-	-	-	-	-	-		
	8	*m.l.*	**Paths**	1	1	1	1	1	1	1	1	1	1	1	-	$\mathcal{O}(\log n)$	0.99837
			Time	0:01	0:01	0:01	0:01	0:01	0:01	0:01	0:01	0:02	0:25	7:09	-		
Quicksort (JDK 1.5)	Exh.		**Paths**	1	2	6	24	120	-	-	-	-	-	-	-		
			Time	0:00	0:00	0:00	0:00	0:01	-	-	-	-	-	-	-		
	8	*m.l.*	**Paths**	1	1	1	1	1	1	1	1	1	1	-	-	$\mathcal{O}(n^2)$	0.99997
			Time	0:00	0:01	0:01	0:01	0:01	0:01	0:02	0:06	0:09	37:42	-	-		
Binary Search Tree search	Exh.		**Paths**	1	3	13	75	541	-	-	-	-	-	-	-		
			Time	0:00	0:00	0:00	0:01	0:02	-	-	-	-	-	-	-		
	3	*m.l.*	**Paths**	1	1	1	1	1	1	1	1	1	-	-	-	$\mathcal{O}(n)$	1.0000
			Time	0:00	0:01	0:01	0:01	0:01	0:01	0:03	0:05	0:13	-	-	-		
Merge Sort (JDK 1.5)	Exh.		**Paths**	1	2	6	24	120	3628800	-	-	-	-	-	-		
			Time	0:00	0:00	0:00	0:00	0:01	2:06:22	-	-	-	-	-	-		
	7	*m.l.*	**Paths**	1	1	1	1	1	251	-	-	-	-	-	-	$\mathcal{O}(n \log n)$	0.99591
			Time	0:00	0:00	0:00	0:00	0:00	0:02	-	-	-	-	-	-		
	7	1	**Paths**	1	1	1	1	1	1	1	1	1	1	1	-	$\mathcal{O}(n \log n)$	0.99941
			Time	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:01	0:11	2:03	-		
	8	1	**Paths**	1	1	1	1	1	1	1	1	1	1	1	-	$\mathcal{O}(n \log n)$	0.99962
			Time	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:00	0:11	1:33	-		
Bellman-Ford‡	Exh.		**Paths**	1	2	63	-	-	-	-	-	-	-	-	-		
			Time	0:00	0:00	0:02	-	-	-	-	-	-	-	-	-		
	2	*m.l.*	**Paths**	1	1	1	1	1	1	-	-	-	-	-	-	$\mathcal{O}(n^3)$	1.0000
			Time	0:00	0:00	0:00	0:01	0:02	6:19	-	-	-	-	-	-		
Dijkstra's‡	Exh.		**Paths**	1	1	4	56	2592	-	-	-	-	-	-	-		
			Time	0:00	0:00	0:00	0:00	0:10	-	-	-	-	-	-	-		
	3	*m.l.*	**Paths**	1	1	1	1	1	1	1	1	1	-	-	-	$\mathcal{O}(n^2)$	1.0000
			Time	0:00	0:00	0:00	0:00	0:00	0:01	0:03	0:11	1:16	-	-	-		
Traveling Salesman‡	Exh.		**Paths**	1	1	3	297	-	-	-	-	-	-	-	-		
			Time	0:00	0:00	0:00	0:02	-	-	-	-	-	-	-	-		
	3	*m.l.*	**Paths**	1	1	1	1	1	-	-	-	-	-	-	-	$\mathcal{O}(n!)$	0.99935
			Time	0:01	0:01	0:01	0:02	0:04	-	-	-	-	-	-	-		
Insertion Sort	Exh.		**Paths**	1	2	6	24	120	3628800	-	-	-	-	-	-		
			Time	0:00	0:00	0:00	0:00	0:01	1:37:46	-	-	-	-	-	-		
	2	*m.l.*	**Paths**	1	1	1	1	1	1	1	1	1	1	1	1	$\mathcal{O}(n^2)$	1.0000
			Time	0:01	0:01	0:01	0:01	0:01	0:01	0:01	0:01	0:01	0:02	0:05	1:10		