# Address-Aware Query Caching for Symbolic Execution


David Trabish

Shachar Itzhaky

Noam Rinetzky

Tel-Aviv University, Israel

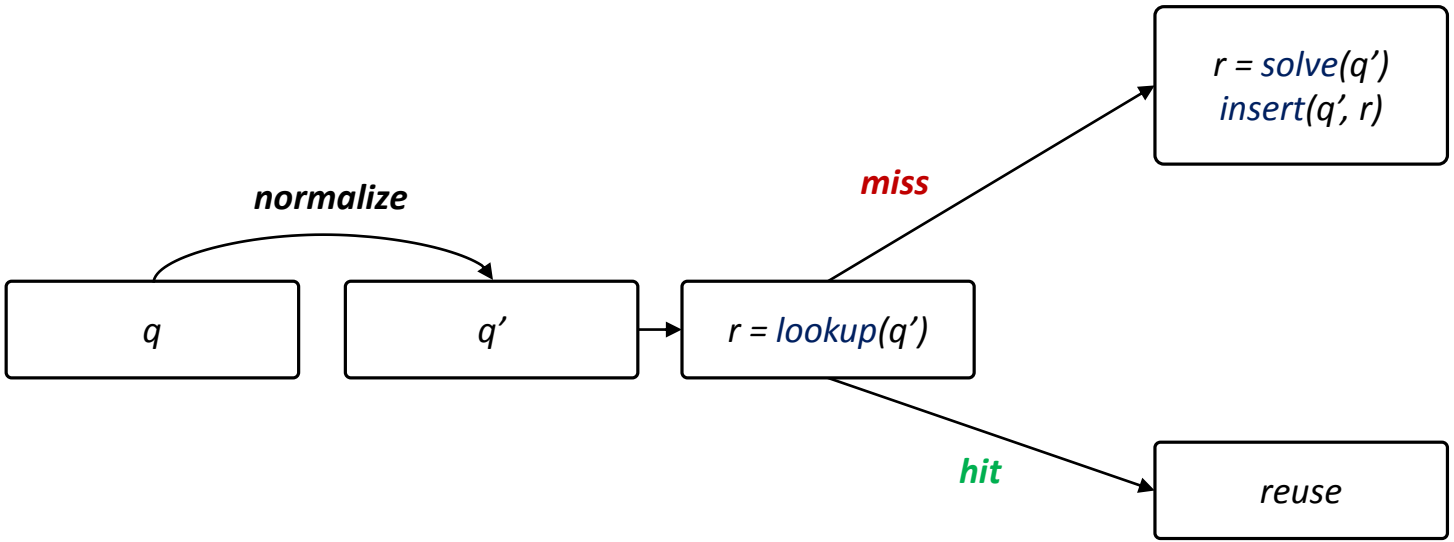
Technion, Israel



**KLEE Workshop 2022**
**(ICST 2021)**

# Query Caching

- Constraint solving is a **main bottleneck**
- A common mitigation – **caching queries!**

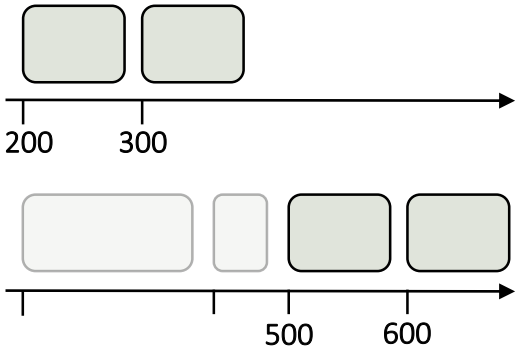


# Query Caching

- Incomplete solution
- There are queries which are:
  - Equi-satisfiable
  - But can’t be reduced to the **same normal form**

# Address-Dependent Queries

address spaces




$$q_1 \stackrel{\text{def}}{=} i < 2 \wedge j < 2 \wedge 200 \leq select(a_1[0 \to 200, 1 \to 300], i) + j \leq 202 \wedge select(a_2, select(a_1[0 \to 200, 1 \to 300], i) + j - 200) = 7$$

$$q_2 \stackrel{\text{def}}{=} i < 2 \wedge j < 2 \wedge 500 \leq select(a_1[0 \to 500, 1 \to 600], i) + j \leq 502 \wedge select(a_2, select(a_1[0 \to 500, 1 \to 600], i) + j - 500) = 7$$

- Equi-satisfiable
- No common normal form

⇨

```
// symbolic: z, i < 2, j < 2
if (z > 1) allocate();
array = calloc(2, sizeof(char *));
for (int i = 0; i < 2; i++)
  array[i] = calloc(2, 1);
array[0][1] = 7;
if (array[i][j] == 7) ...
```

# Address-Aware Query Caching

Goal

- Efficiently apply query caching for address-dependent queries

How?

- Modified expression encoding
- Matching algorithm

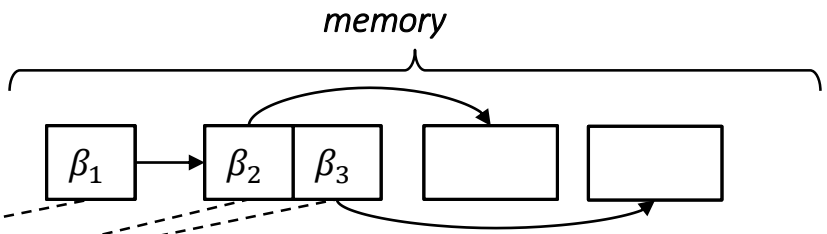
# Expression Encoding

Use the relocatable addressing model (ISSTA'20)

- Base addresses are **symbolic values**
- Maintain **address constraints** to avoid **overlapping**

# Expression Encoding

```
// symbolic: z, i < 2, j < 2
if (z > 1) allocate();
array = calloc(2, sizeof(char *));
for (int i = 0; i < 2; i++)
  array[i] = calloc(2, 1);
array[0][1] = 7;
if (array[i][j] == 7) ...
```




*address constraints*

$\beta_1 = 100, \beta_2 = 200, \beta_3 = 300$

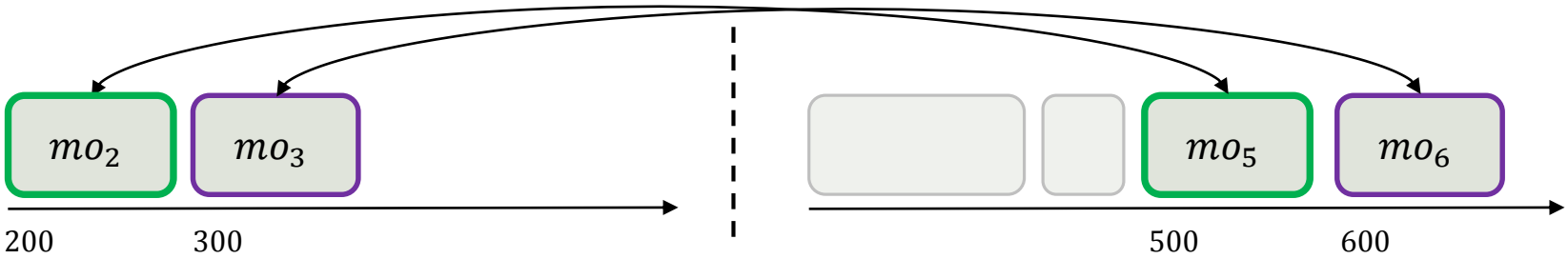
$$q_1 \stackrel{\text{def}}{=} i < 2 \wedge j < 2 \wedge$$
$$\beta_2 \leq select(a_1[0 \rightarrow \beta_2, 1 \rightarrow \beta_3], i) + j \leq \beta_2 + 2 \wedge$$
$$select(a_2, select(a_1[0 \rightarrow \beta_2, 1 \rightarrow \beta_3], i) + j - \beta_2) = 7$$

# Matching Algorithm

$$q_1 \stackrel{\text{def}}{=} i < 2 \wedge j < 2 \wedge \beta_2 \leq select(a_1[0 \to \beta_2, 1 \to \beta_3], i) + j \leq \beta_2 + 2 \wedge select(a_2, select(a_1[0 \to \beta_2, 1 \to \beta_3], i) + j - \beta_2) = 7$$

$$q_2 \stackrel{\text{def}}{=} i < 2 \wedge j < 2 \wedge \beta_5 \leq select(a_1[0 \to \beta_5, 1 \to \beta_6], i) + j \leq \beta_5 + 2 \wedge select(a_2, select(a_1[0 \to \beta_5, 1 \to \beta_6], i) + j - \beta_5) = 7$$

$\beta_2 \leftrightarrow \beta_5$
$\beta_3 \leftrightarrow \beta_6$




$|mo_2| = |mo_5|$
$|mo_3| = |mo_6|$

# Evaluation

- Implemented on top of KLEE
  - *https://github.com/davidtr1037/klee-aaqc*
- Evaluated on several benchmarks:
  - m4, make, sqlite, apr, libxml2, expat, bash, json-c, coreutils,libosip,libyaml
- Performance improvement:
  - More query cache hits
  - Faster analysis time
  - Lower memory usage