Finding Races in the Heap

Vesal Vojdani

University of Tartu
Technische Universität München

Mäetaguse Theory Days ’09

Data Races

Definition (Race condition)

An unintended indeterminism due to the lack of proper ordering constraints in the program.

- Lead to subtle and dangerous bugs.
- Violate Murphy's Law $\Rightarrow$ hard to test!

Famous Examples

- Therac-25 radiation therapy machine
    - Killed 3 people!
    - Race condition only occurred if setup was changed quickly; testers were not as fast.
- North American Blackout of 2003
    - Two processes got write accesses to a shared resource and corrupted it.
    - Alarm subsystem looped indefinitely.
    - The race had a window of only milliseconds!

Outline of Talk

1. Coarse-grained locking (static locks)
    - The Lockset Algorithm
2. Fine-grained (per-element locking)
    - JAVA: Conditional Must-Not Aliasing
    - C: Existentially Typed Flow (requires annotations)
    - C: Interprocedural Must-Alias Analysis
3. Medium-grained (per-bucket locking)
    - JAVA: Disjoint Reachability Analysis
    - C: Region Analysis
4. Unified Approach^{TM} to Race Detection

The Lockset Analysis

- For each program point
  - Compute set of locks that must be held.
  - `lock`$(\mathtt{l})$ adds the lock that $\mathtt{l}$ must point to.
  - `unlock`$(\mathtt{l})$ removes locks that $\mathtt{l}$ may point to.
- For each expression e in the program
  - Check if e may point to a shared variable.
  - Write down the access with set of mutexes held.
- Shared var has no common lock $\Rightarrow$ race!

Simple Example (no race)

$$\begin{aligned} T_1: &\ \texttt{lock}(\&l_1); \\ & v = v + 1; \\ & \texttt{unlock}(\&l_1); \end{aligned}$$

$$\begin{aligned} T_2: &\ \texttt{lock}(\&l_1); \\ & v = v + 1; \\ & \texttt{unlock}(\&l_1); \end{aligned}$$

- List of accesses:
  $\langle v, \{l_1\}, \text{write}, \texttt{file.c} : 2\rangle$
  $\langle v, \{l_1\}, \text{write}, \texttt{file.c} : 5\rangle$
- $\mathtt{v}$ is protected by $\{l_1\}$.

Simple Example (race!)

$T_1:$ $\texttt{lock}(\&l_1);$

$v = v + 1;$

$\texttt{unlock}(\&l_1);$

$T_2:$ $\texttt{lock}(\&l_2);$

$v = v + 1;$

$\texttt{unlock}(\&l_2);$

- List of accesses:
  $\langle v, \{l_1\}, \text{write}, \texttt{file.c} : 2\rangle$
  $\langle v, \{l_2\}, \text{write}, \texttt{file.c} : 5\rangle$
- No common lock!

Complications

- Context-Sensitivity
- Path-Sensitivity
- Synchronization-Sensitivity

- Dynamic Memory Allocation

Dynamic Data, Static Locks

$\mathtt{p} = \mathtt{malloc}();$
$\mathtt{lock}(\&\mathtt{l});$
$\mathtt{p} \rightarrow \mathtt{d} = 5;$
$\mathtt{unlock}(\&\mathtt{l});$

- List of accesses: $\langle \text{alloc@1.d}, \{\mathtt{l}\}, \ldots \rangle$
- Blob all elements allocated at that point.
- Can be handled as before.

Dynamic Data, Dynamic Locks

```
p = q = malloc();
lock(&p→l);
q→d = 7;
unlock(&p→l);
```

- List of accesses: $\langle \mathrm{alloc@1.d}, \{\mathrm{alloc@1.l}\}, \ldots \rangle$
- Is it the same element in the blob?
- Must-equality analysis!

Finding Races in the Heap

- Keep a symbolic lockset

$$\{p.l\}$$

- Use address must-equalities to match symbolic locksets with accesses.

$$\models p = q$$

- Use may points-to analysis to associate inferred invariant with memory locations.

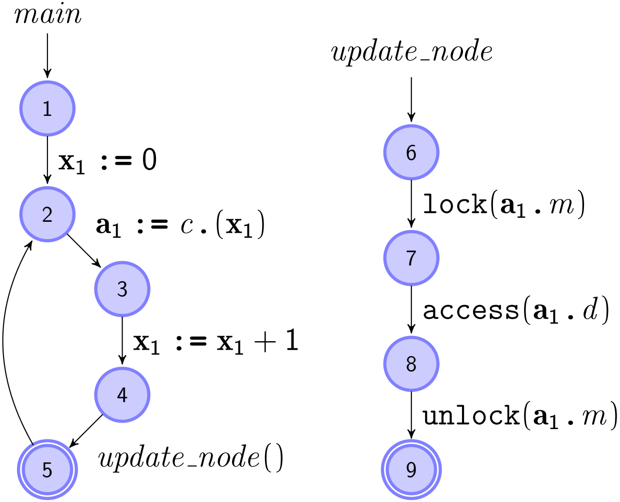
$$p \mapsto \{\mathrm{alloc@1}\}$$

Per-element locking

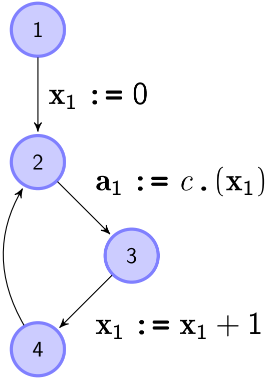
```
typedef struct {
  int datum;
  char filename[80];
  pthread_mutex_t mutex;
} node;

node cache[100];
```

Traverse cache



Valid Must-Equalities



1. No equalities hold!
2. No equalities hold!
3. $\mathbf{a}_1 \doteq c\,.\,(\mathbf{x}_1)$.
4. $\mathbf{a}_1 \doteq c\,.\,(-1 + \mathbf{x}_1)$.

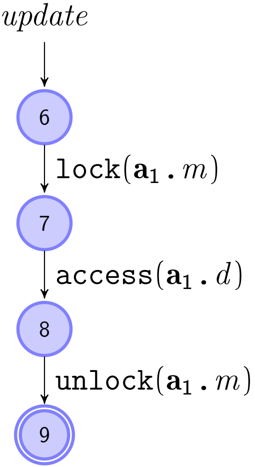
Abstract domain

$$\begin{tikzcd}
 & & \mathsf{true} & & \\
 & 8 - 4\mathbf{x}_1 \doteq 0 \arrow[ur, Rightarrow] & \cdots \arrow[u, Rightarrow] & \mathbf{a}_1 \doteq c\,\mathbf{.}\,(2) \arrow[ul, Rightarrow] & \\
\cdots \arrow[ur, Rightarrow] & & (4 - 2\mathbf{x}_1 \doteq 0) \wedge (\mathbf{a}_1 \doteq c\,\mathbf{.}\,(\mathbf{x}_1)) \arrow[ul, Rightarrow] \arrow[u, Rightarrow] \arrow[ur, Rightarrow] & & \cdots \arrow[ul, Rightarrow] \\
 & & \mathsf{false} \arrow[ull, Rightarrow] \arrow[u, Rightarrow] \arrow[urr, Rightarrow] & &
\end{tikzcd}$$

How do we compute $\mathrm{E}_1 \Rightarrow \mathrm{E}_2$?

Is this a complete lattice?

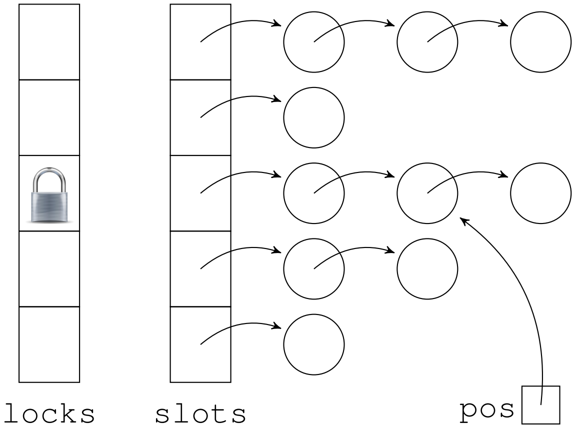
Back to Race detection



- We obtain the lockset: $\{c \,.\, (-1 + \mathbf{x}_1) \,.\, m,\ \mathbf{a}_1 \,.\, m\}$
- Lockset must equal held locks at the access!
- Correlate $\mathbf{a}_1 \,.\, d$ to $\mathbf{a}_1 \,.\, m$.
- Associate invariant with array c.

- Why integer equalities?

Synchronized Hashtable



What do we need?

- Infer the locked addresses
  $\texttt{locks}[i]$
- Information about pointers
  $\texttt{pos} \in \texttt{slot}[i]$
- Disjointness information
  $\texttt{slot}[i] \cap \texttt{slot}[j] = \emptyset$

Region Analysis

- Heap Abstraction:

$$\Pi \times \mathrm{R}$$

- What are the disjoint regions?
- What region does each pointer belong to?

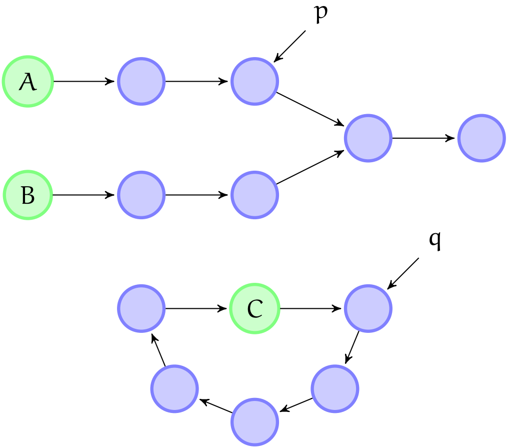
Region Analysis

- Heap Abstraction:

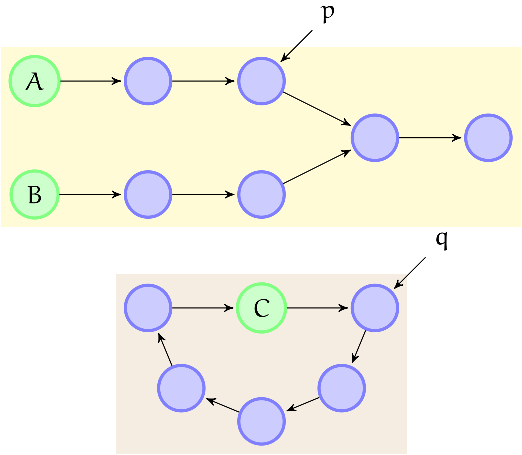
$$\Pi \times R$$

- lattice of partitions on A (owners)
- region mapping $R\colon V \to \mathcal{P}(A \cup \{\bullet\})$

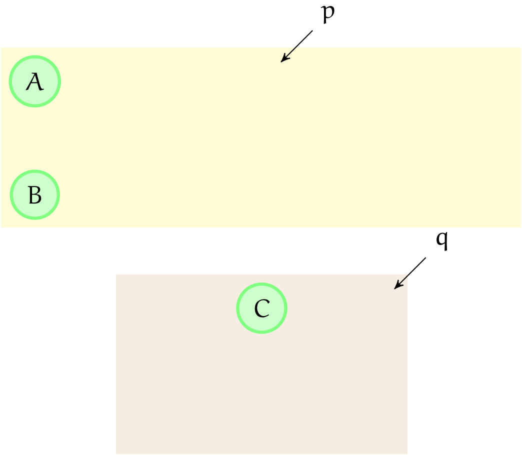
Abstraction



Abstraction



Abstraction



Abstraction

- Partitions: $\{A, B\}, \{C\}$
- Mapping:
  $p \mapsto \{A, B\}$
  $q \mapsto \{C\}$

Analysis

$$p = \texttt{malloc}();$$
$$q = \&A;$$
$$p \rightarrow n = q \rightarrow n;$$
$$q \rightarrow n = \&B;$$
$$\texttt{if}\ (\star)\ q = \&C;$$

- Partitions:
  $\{A\}, \{B\}, \{C\}$
- Mapping:
  $p \mapsto \emptyset$
  $q \mapsto \emptyset$

Analysis

$p = \texttt{malloc}();$

$q = \&A;$

$p \rightarrow n = q \rightarrow n;$

$q \rightarrow n = \&B;$

$\texttt{if}\ (\star)\ q = \&C;$

- Partitions:
  $\{A\}, \{B\}, \{C\}$
- Mapping:
  $p \mapsto \{\bullet\}$
  $q \mapsto \emptyset$

Analysis

$\mathtt{p} = \mathtt{malloc}();$

$\mathtt{q} = \&A;$

$\mathtt{p} \rightarrow \mathtt{n} = \mathtt{q} \rightarrow \mathtt{n};$

$\mathtt{q} \rightarrow \mathtt{n} = \&B;$

$\mathtt{if}\ (\star)\ \mathtt{q} = \&C;$

- Partitions:
  $\{A\}, \{B\}, \{C\}$
- Mapping:
  $\mathtt{p} \mapsto \{\bullet\}$
  $\mathtt{q} \mapsto \{A\}$

Analysis

$\mathtt{p} = \mathtt{malloc}();$

$\mathtt{q} = \&A;$

$\mathtt{p} \rightarrow \mathtt{n} = \mathtt{q} \rightarrow \mathtt{n};$

$\mathtt{q} \rightarrow \mathtt{n} = \&\mathrm{B};$

$\mathtt{if}\ (\star)\ \mathtt{q} = \&\mathrm{C};$

- Partitions:
  $\{A\}, \{B\}, \{C\}$
- Mapping:
  $\mathtt{p} \mapsto \{A\}$
  $\mathtt{q} \mapsto \{A\}$

Analysis

```
p = malloc();
q = &A;
p→n = q→n;
q→n = &B;
if (⋆) q = &C;
```

- Partitions:
  $\{A, B\}, \{C\}$
- Mapping:
  $p \mapsto \{A, B\}$
  $q \mapsto \{A, B\}$

Analysis

$\mathtt{p} = \mathtt{malloc}();$

$\mathtt{q} = \&A;$

$\mathtt{p} \rightarrow \mathtt{n} = \mathtt{q} \rightarrow \mathtt{n};$

$\mathtt{q} \rightarrow \mathtt{n} = \&B;$

$\mathtt{if}\ (\star)\ \mathtt{q} = \&C;$

- Partitions: $\{A, B\}, \{C\}$
- Mapping: $\mathtt{p} \mapsto \{A, B\}$ $\mathtt{q} \mapsto \{C\}$

Analysis

$p = \texttt{malloc}();$

$q = \&A;$

$p \rightarrow n = q \rightarrow n;$

$q \rightarrow n = \&B;$

$\texttt{if}\ (\star)\ q = \&C;$

- Partitions: $\{A, B\}, \{C\}$
- Mapping:
  $p \mapsto \{A, B\}$
  $q \mapsto \{A, B, C\}$

Dealing with the arrays

- Allow symbolic index expressions.
- If we need to join two regions with owners $A[e_1]$ and $A[e_2]$
  - If we $\models e_1 = e_2$, just keep one of them.
  - Otherwise, collapse the array.
- Partition lattice Π tracks collapsed arrays.
- If an array A has not collapsed in Π,
  the regions of $A[\mathrm{i}]$ and $A[\mathrm{j}]$ are disjoint ($\mathrm{i} \neq \mathrm{j}$).

Back to the example

- The symbolic lockset will be

  $$\{\texttt{locks.(}\texttt{i}\texttt{)}\}$$

- As we access `pos`, we have
  - Points-to analysis: `pos` $\mapsto$ {alloc@55}.
  - Region analysis: `pos` $\in$ `slots.(i)`.
- Associate the invariant with the array. Why?

Unified Approach to Race Detection

- Want to deal with dynamic regions.
    - List (instead of array) of regions.
    - Type-based regions for, e.g., per-device structures.
- Where to associate invariants?
    - Static names.
    - Allocation sites.
    - Types.
    - Regions.

Solution

- Under construction . . .
- All the ingredients are there.
- Implementation is making good progress.
- Where are the theorems?