Speeding Up Maximal Causality Reduction with Static Analysis

Shiyou Huang  Jeff Huang
huangsy@tamu.edu
Parasol Lab, Texas A&M University
Maximal Causality Reduction (MCR)

Concurrent Program Verification is Hard

Huge Interleaving Space

Stateless Model Checker

MCR

(Huang, PLDI’15)

Under the given input

+ No redundancy
+ Sound and Complete
+ More efficient than DPOR$^1$ and ICB$^2$

- Purely Dynamic, #constraints cubic in trace size
- Without considering input non-determinism

1. DPOR: Flanagan and Godefroid, PLDI’05
2. ICB: Musuvathi, OSDI’08
Maximal Causality Reduction (MCR)

- **Trace**: A sequence of events executed by the program
- **Constraints**: An order variable \( O \) for each event in the trace
  - E.g., if \( e_1 \) happens before \( e_2 \), \( O_{e_1} < O_{e_2} \)
- **Interleaving**: A sequence of thread schedule

(Huang, PLDI’15)
Constraints Model -- $\Omega(t)$

$$\Omega(t) = \phi_{mhb} \land \phi_{lock} \land \phi_{validity} \land \phi_{state}$$

- **must-happen-before**($\phi_{mhb}$)
  E.g., $O_1 < O_2$ if $e_1$ and $e_2$ are by the same thread, and $e_1$ occurs before $e_2$

- **lock-mutual-exclusion**($\phi_{lock}$)
  E.g., for a lock pair, $(l_1, u_1)$ and $(l_2, u_2)$, $O_{u_1} < O_{l_2} \lor O_{u_2} < O_{l_1}$

- **validity**($\phi_{validity}$)
  an event is feasible if every read that must-happen-before it returns the same value

- **new state**($\phi_{state}$)
  At least one read in $t$ returns a different value
An Example

Init: $x=y=0$

### Possible schedules:
1. 1-2-3-4-5
2. 1-2-4-3-5
3. 1-4-5-2
4. ...

### Constraints:
- **HB:** $e_1 < e_2 < e_3$, $e_4 < e_5$
- **State:** $e_3 < e_4$
- **Validity:** $e_1 < e_5$, $e_2 < e_4$

### S0: 1-2-3-4-5, $r_1 = r_2 = 0$, $True \equiv x == 0$
- return the same value as that in S0 to enforce $True \equiv x == 0$

### 4-1-2-3-5
- $x$
- $False \equiv x == 0$

### 1-2-4-3-5
- $✓$

**Diagram:**
- **T1**
  1: $r_1 = y$
  2: if $(x==0)$
  3: $r_2 = x$
- **T2**
  4: $x=1$
  5: $y=1$
Validity Constraints

\(<_e \) : set of events that happen before \( e \)

\( W_v^x \) : set of writes that write value \( v \) to a variable, \( x \)

\( W^x \) : set of writes that write other values to \( x \)

\[ \Phi_{validity} = \bigwedge_{r \in <_e} \Phi_{value}(r, v), \]

\( \Phi_{value}(r, v) \) enforces \( r \) returns the value \( v \)

\[ \Phi_{value}(r, v) \equiv \bigvee_{w \in W_v^x} (\Phi_{validity}(w) \land O_w < O_r) \]

\[ \land_{w \neq w' \in W^x} (O_{w'} < O_w \lor O_r < O_{w'}) \]

- every read \( r \) before \( e \), return the same value \( v \)
- match \( r \) to a write that writes the value \( v \) to the same location
Limitations

Most events are reads and writes in a trace

- Complicated constraints, \textbf{cubic} in the size of the trace

Just a few reads influence the reachability of a later event

- Construct unnecessary constraints

\[\begin{array}{c|cc|cc}
T1 & r1=y & x & T2 & \text{hb} \\
1: & r1=y & x & 4: & x=1 \\
2: & \text{if (x==0)} & 5: & y=1 \\
3: & \text{r2=x} \\
\end{array}\]
Our Approach

Trace \rightarrow \text{Ordering Constraints} \rightarrow \text{More Schedules}

- Events happen before r5: r1, r2, r3, r4
- r5 depends on: r1, r2, r3, r4

\[ \phi_{validity}(r5) = \phi_{value}(r1, v) \land \phi_{value}(r4, v') \land \phi_{value}(r2, v') \land \phi_{value}(v3, v') \]

Reduced
Our Approach

Trace \rightarrow \text{Ordering Constraints} \rightarrow \text{More Schedules}

\begin{itemize}
  \item events happen before r5: r1, r2, r3, r4
  \item r5 depends on: r1, r2, r3, r4
\end{itemize}

MCR + Static Dependency Analysis

\[ \psi_{value} \left( \Phi^i \right) \]
System Dependency Graph (SDG)

Procedure **main()**
- `sum = 0;`
- `i = 1;`
- **while** `i<11:`
  - `sum = add(sum, i);`
  - `i = i+1;`

Procedure **add(x, y)**
- `x = x+y;`
- **return** `x;`
Case a: an event is directly depends on a read operation evaluated by an if predicate
\[ x == 1 \implies r = y \]

Case b: the dependency may be transmitted via a data dependency
\[ a = x \]
\[ \implies r = y \]
Control Dependency

Case c: the evaluation may depend on the return value of another procedure

\[ \text{return } x \rightarrow r = y \]

Case d: the read may depend on a if predicate in a different procedure

\[ x == 1 \rightarrow r = y \]
Control Dependency

Definition: given two nodes n1 and n2 in an SDG, we use $n1 \delta^c n2$ to denote that n2 is control dependent on n1

$$n1 \delta^c n2 \Leftrightarrow n1 \rightarrow e_{CD} n2,$$

\[ e := null \]

| CD | DD | PI | PO | CL |

CD: control dependency
DD: data dependency
PI/O: parameter in/out
CL: call
Constraints Reduction

Main Idea:
Only enforce reads that are control-dependency related to return the same value

\[
\begin{align*}
\prec_{\tau}(e) &\leftarrow \text{Happens-before}(\tau, e) \\
\prec_{\tau}^D(e) &\leftarrow \text{DependencyComputation}(\prec_{\tau}(e), e) \\
\text{foreach} & \text{ read } r \in \prec_{\tau}^D(e) \text{ with value } v \text{ do} \\
& \quad \text{// } \Phi_{\text{value}}(r, v) \text{ recursively call } \text{DataValidityConstraints() } \\
& \quad \Phi_{\text{validity}} \land = \Phi_{\text{value}}(r, v) \\
\text{end}
\end{align*}
\]
Redundancy Problem

$S_0: 1-2-3, \ r_1 = r_2 = 1$

Init $x=0$

- T1
  1: $x=1$
- T2
  2: $r_1=x$
  3: $r_2=x$

Since $e_2 < e_3$, $e_2$ is enforced to return value 1

$r_2 = 0$
Redundancy Problem

S0: 1-2-3, \( r_1 = r_2 = 1 \)

Any order

T1: \( x = 1 \)

T2: \( r_1 = x \)

3: \( r_2 = x \)

Init \( x = 0 \)

Since \( e_3 \) is not control dependent on \( e_2 \), \( e_2 \) can read from any writes.

Our approach

\( r_2 = 0 \)
Solution to Redundancy Problem

We treat the events into two categories:
1. target read: a read considered to see a different value
2. other events

\[ \prec_{\tau} (e) \leftarrow \text{Happens-before}(\tau, e) \]

// target read: read considered to return new values

\[
\text{if } e \text{ is not a TARGET READ then} \\
\text{    } \prec_{\tau} (e) \leftarrow \text{DependencyComputation}(\prec_{\tau} (e), e) \\
\text{end}
\]

\[
\text{foreach read } r \in \prec_{\tau} (e) \text{ with value } v \text{ do} \\
\text{    } \Phi_{\text{value}}(r, v) \text{ recursively call DataValidityConstraints()} \\
\text{    } \Phi_{\text{validity}} \land = \Phi_{\text{value}}(r, v) \\
\text{end}
\]
Evaluation

- Dependency analysis using JOANA\textsuperscript{1} [Graf] and WALA\textsuperscript{2}
- Comparisons with MCR
  - \#reads/constraints reduced
  - solving time reduced
- Benchmarks [Huang, PLDI’15]

Benchmarks and SDG

<table>
<thead>
<tr>
<th>Program</th>
<th>time(s)</th>
<th>memory(M)</th>
<th>#nodes</th>
<th>#edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Counter</td>
<td>2.00</td>
<td>69</td>
<td>289</td>
<td>1,440</td>
</tr>
<tr>
<td>Airline</td>
<td>2.10</td>
<td>79</td>
<td>809</td>
<td>4,902</td>
</tr>
<tr>
<td>Pingpong</td>
<td>2.52</td>
<td>83</td>
<td>914</td>
<td>5,244</td>
</tr>
<tr>
<td>BubbleSort</td>
<td>2.14</td>
<td>81</td>
<td>911</td>
<td>5,710</td>
</tr>
<tr>
<td>Pool</td>
<td>3.67</td>
<td>75</td>
<td>2,848</td>
<td>17,586</td>
</tr>
<tr>
<td>StringBuf</td>
<td>2.96</td>
<td>111</td>
<td>2,129</td>
<td>12,310</td>
</tr>
<tr>
<td>Weblech</td>
<td>8.01</td>
<td>219</td>
<td>22,094</td>
<td>167,492</td>
</tr>
<tr>
<td>Derby</td>
<td>69.67</td>
<td>1,385</td>
<td>115,658</td>
<td>2,409,784</td>
</tr>
</tbody>
</table>

Table 1

To show the effectiveness improved by our hybrid analysis, we run our approach on the same benchmark set used by prior work \[16\] so that we can make a direct comparison. Table 1 summarizes the benchmarks evaluated in this work. Counter is the example introduced in Section 1, and we take \( \text{Max} = 5 \) during the evaluation.

Pingpong is a program that can arouse an NPE error on the shared variable player. BubbleSort is a small but read-write intense program with more than 10 million interleavings. Pool contains a concurrency bug in Apache Commons Pool causing more instances than allowed in the pool.

StringBuf contains an atomicity violation. Weblech and Derby are two large real-world programs with long trace and complicated constraints.

Table 2 reports the results by MCR, MCR-S and MCR-S+ on the benchmarks. Column \#reads lists the number of the reads the three approaches considered totally when constructing constraints to explore new interleavings. Column \#constraints gives the total number of data-validity (validity) constraints that map a read to a certain write. The number is the sum of the constraints generated by each exploration in the whole state-space search. As the other constraints remain the same for MCR and the new approaches, we just discuss the read-write constraints in the evaluation. Column time shows the time used by the solver to solve the constraints.

Figure 7 presents the reduction results by MCR-S and MCR-S+ compared to MCR on the number of the reads and constraints as well as the solving time. The figure is best viewed in color. The blue bar represents the results by MCR, green for MCR-S and yellow for MCR-S+, respectively. For comparison, we normalize MCR's results to 1 as the baseline and length of the green and yellow bars represents the ratio of the results of MCR-S and MCR-S+ to that of MCR.

Number of reads reduced.

Figure 7(a) summarizes the comparison on the number of the reads reduced by MCR and our approaches. Averagely, MCR-S reduces the number of the reads by \( 27.1\% \) and MCR-S+ by \( 12.1\% \) compared to MCR. And the reduction percentage by MCR-S ranges from \( 14.2\% \) to \( 51.3\% \), and MCR-S makes the greatest reduction on the Derby benchmark. Comparing time memory

<table>
<thead>
<tr>
<th></th>
<th>time</th>
<th>memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg.</td>
<td>11.6s</td>
<td>263M</td>
</tr>
</tbody>
</table>
Comparison with MCR

- **MCR-S**: Optimization with redundant executions
- **MCR-S+**: No redundancy, but less reads reduced

<table>
<thead>
<tr>
<th>Approach</th>
<th>MCR-S</th>
<th>MCR-S+</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reads</td>
<td>27.1% ↓</td>
<td>12.1% ↓</td>
</tr>
<tr>
<td>Constraints</td>
<td>31.6% ↓</td>
<td>15.7% ↓</td>
</tr>
<tr>
<td>Solving time</td>
<td>27.8% ↓</td>
<td>26.2% ↓</td>
</tr>
</tbody>
</table>
Conclusion & Future Work

- Improvement over MCR
  - #reads/constraints: 12.1% - 27.1%, 15.7% - 31.6
  - solving time: ~27%

- Future work
  - take input non-determinism into consideration
  - release the tool
Thank You