Model Checking Race-Freeness
MCC’08

Frédéric Haziza <daz@it.uu.se>

Department of Computer Systems
Uppsala University

Friday, November 28th, 2008
Race Condition

Situation in which one process changes a variable which another process has previously read and the other process does not get notified of the change.

Classic

Write

Read

Write

i.e. getting the right answer relies on lucky timing.

Check-then-act

if (check){

act

}

Update
Related Work

- Feasible races vs. False positives
- Coverage
- Expressiveness
- Annotations

On-the-fly
- dynamic tools

Ahead-of-time
- static analysis
- compile-time heuristics

Post-mortem
Combination of static and dynamic techniques
Outline

1. Language
2. Model
3. Backward Reachability Analysis
4. Experiments and Future Work
Language

We analyze programs

- written in C
- using POSIX threads.

The language we allow

- abstracts away CPU operations (add, sub, ⋅ ⋅ ⋅ )
- narrows down to movers operations (load, store, ⋅ ⋅ ⋅ )
- thread synchronization and bookkeeping
Language

\[ X, Y, Z, \ldots \]

\begin{align*}
L_1 & \quad L_3 & \quad L_4 \\
L_2 & \quad L_4 & \\
\vdots & \quad \vdots & \\
\end{align*}

Thread 1 \quad \text{Thread 2} \quad \text{Thread 3} \quad \text{Thread 4}
Language

Control statements
- if
- if-then-else
- while
- for-loop

→ branch and label

Movers
- Read X
- Write X

Lock – mutex
- acquire
- release

Condition Variable
- wait
- signal
Language

X, Y, Z, \ldots, M_1, M_2, \ldots, CV_1, CV_2, \ldots

L_1
L_2
\ldots

Thread 1

L_3
L_4
\ldots

Thread 2

\ldots

Thread 3

\ldots

Thread 4
Language – examples

```c
int counter;
pthread_mutex_t L;
pthread_mutex_lock(L);
counter++;
pthread_mutex_unlock(L);
```

```c
shared counter, L;
acquire L;
read counter;
write counter;
release L;
```
Language – examples

```c
int buffer; pthread_mutex_t L;
pthread_cond_t cvEmpty, cvFull;

//Many Producers
pthread_mutex_lock(L);
while (true){ /*branch*/
    pthread_cond_wait(cvEmpty, L);
    buffer = data;
    pthread_cond_signal(cvFull);
}
pthread_mutex_unlock(L);

//Many Consumers
pthread_mutex_lock(L);
while (true){ /*branch*/
    pthread_cond_wait(cvFull, L);
    val = buffer;
    pthread_cond_signal(cvEmpty);
}
pthread_mutex_unlock(L);
```

```c
shared buffer, L, cvEmpty, cvFull;

//Many Producers
acquire L;
while (true){ /*branch*/
    wait cvEmpty, L;
    write buffer;
    signal cvFull;
}
release L;

//Many Consumers
acquire L;
while (true){ /*branch*/
    wait cvFull, L;
    read buffer;
    signal cvEmpty;
}
release L;
```
Model – Petri Nets

\[
\begin{array}{c}
\text{lock} & \quad & \text{init} \\
\text{Read}_{counter} & \quad & \text{Write}_{counter} \\
\text{end} & \quad & L
\end{array}
\]
Reading and Writing a shared variable

\[ \text{Read}_x \]

\[ \text{Write}_x \]
Acquiring and releasing a lock

Diagram:
- **Acquiring:**
  - From `in` to `lock`.
  - From `lock` to `out`.

- **Releasing:**
  - From `in` to `unlock`.
  - From `unlock` to `out`.

\[L\]
Waiting on a condition variable
Signaling a condition variable
Branching and Jumps
Creating a new thread
Data race in our model

A data race occurs when multiple threads access a shared variable and at least one has the intention of changing it, without any synchronization constraints (i.e. event ordering).
Interesting properties

A bad configuration:

\[ Write_v \]

Bad state:

\[ Write_v \]
Ordering
Verification Method: Reachability

Safety property

Backward Reachable?

Init:
Computing Pre
Computing Pre

\[ \text{in}_1 \quad \text{in}_2 \quad \text{in}_3 \]

\[ \text{out}_1 \quad \text{out}_2 \]
Monotonicity

\[ C_1 \preceq C_2 \preceq C_3 \preceq C_4 \]

\[ \exists C_4 \preceq C_3 \]

Language  Model  Backward Reachability  Conclusion
Question!

Is $\text{Pre}()$ an upward-closed set?

Is that enough to take to $\text{pre}$ of the generator?
Backward reachability analysis

\[ \text{Pre}^* \]
Termination and Correctness

- WQO
  - Termination
- Over-approximation
  - Monotonicity
  - Pre$^*$ is computable
  - Correctness

Termination and Correctness

Language Model Backward Reachability Conclusion
# Experiments

<table>
<thead>
<tr>
<th>Programs</th>
<th>#Conf.</th>
<th>#Subsum.</th>
<th>#Iter.</th>
<th>Safe?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Counter</td>
<td>10</td>
<td>3</td>
<td>4</td>
<td>-</td>
</tr>
<tr>
<td>CounterWithLock</td>
<td>14</td>
<td>7</td>
<td>6</td>
<td>✓</td>
</tr>
<tr>
<td>CheckThenAct</td>
<td>20</td>
<td>12</td>
<td>5</td>
<td>-</td>
</tr>
<tr>
<td>CheckThenAct-Lock</td>
<td>13</td>
<td>4</td>
<td>4</td>
<td>✓</td>
</tr>
<tr>
<td>Prods/Cons</td>
<td>310</td>
<td>645</td>
<td>19</td>
<td>✓</td>
</tr>
<tr>
<td>Prods/Cons 2</td>
<td>290</td>
<td>561</td>
<td>17</td>
<td>✓</td>
</tr>
<tr>
<td>Lock-ReadWriteOnly</td>
<td>25</td>
<td>13</td>
<td>8</td>
<td>✓</td>
</tr>
</tbody>
</table>
Previous Work

- Mutual exclusion protocols, bus protocols, cache coherency protocols, telecommunication protocols
- Parameterized Tree Systems ← FORTE'08, Tokyo, June 2008
- Shape Analysis ← CAV'08, Princeton, July 2008
Conclusions

- Race-Freeness: Safety Property
- Backward Reachability analysis
- Symbolic representation

Future Work

- More pthreads constructs and maybe other libraries
- Usefullness of the method: false positives and coverage
- Compare with other tools
- Better Error reporting when a race is found