C/C++ Concurrency: Formalization and Model Finding

Mark Batty  Jasmin Blanchette  Scott Owens
Susmit Sarkar  Peter Sewell  Tjark Weber

7th International Workshop on Reachability Problems (RP’2013)
Informal Presentation
Concurrency in C/C++ (Before 2011)

- Pthreads
- Hardware model

😊😊😊
C11 / C++11

New versions of the ISO standards for C and C++ were ratified in 2011.

These standards define a memory model for C/C++ that allows programmers to write portable, yet highly efficient concurrent code.

Support for this model has recently become available in popular compilers (GCC 4.4, Intel C++ 13.0, MSVC 11.0, Clang 3.1).
Memory Models

A memory model describes the interaction of threads through shared data.
Sequential Consistency

“The result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program.”
Simple concurrency:

- Sequential consistency for data-race free code (→ locks).
- Data races cause undefined behavior.

Expert concurrency:

- Atomic memory locations
Data Races

- Two (or more) threads concurrently\(^1\) access the same memory location.
- At least one of the threads writes.

Example (Dekker’s algorithm)

```plaintext
int x(0); int y(0);
x = 1;       || y = 1;
r1 = y;     || r2 = x;
```

\(^1\)I.e., not ordered by happens-before.
Data Races

- Two (or more) threads concurrently\(^1\) access the same memory location.
- At least one of the threads writes.

Example (Dekker’s algorithm)

\[
\text{int } x(0); \quad \text{int } y(0);
\]
\[
x = 1; \quad \| \quad y = 1;
\]
\[
r1 = y; \quad \| \quad r2 = x;
\]

\(^1\text{i.e., not ordered by happens-before.}\)
Data Races

- Two (or more) threads concurrently\(^1\) access the same memory location.
- At least one of the threads writes.

Example (Dekker’s algorithm)

```plaintext
int x(0); int y(0);

x = 1;  // I.e., not ordered by happens-before.
y = 1;

r1 = y;  // I.e., not ordered by happens-before.
r2 = x;
```

\(^1\)I.e., not ordered by happens-before.
Data Races

- Two (or more) threads concurrently\(^1\) access the same memory location.
- At least one of the threads writes.

\(^1\)I.e., not ordered by happens-before.
std :: atomic<T>

Operations:
- x.load(memory_order)
- x.store(T, memory_order)

Concurrent accesses on atomic locations do not race.¹

The memory_order argument specifies ordering constraints between atomic and non-atomic memory accesses in different threads.

¹Except during initialization.
std :: memory_order

- strict
  - seq_cst
  - release / acquire
  - release / consume

- relaxed
  - no synchronization

- total order (SC for DRF code)

- message passing
Program executions consist of memory actions. The program source determines several relations over these actions.

Example

```c
int x = 0;
int y = (x == x);
```
A candidate execution is specified by three relations:

- \( \textbf{sc} \) is a total order over all seq_cst actions.
- \( \textbf{reads-from} \ (\textbf{rf}) \) relates write actions to read actions at the same location that read the written value.
- For each atomic location, the \( \textbf{modification order} \ (\textbf{mo}) \) is a total order over all writes at this location.

From these, various other relations (e.g., happens-before) are derived.
The memory model imposes constraints on these relations.
Program Semantics

Consider all consistent candidate executions.

If at least one of them has a data race, the program has undefined behavior.

Otherwise, its semantics is the set of consistent candidate executions.

\(^2\)There are actually several kinds.
Exploring Program Behavior

source code

static semantics

consistent executions
We have generated an executable (OCaml) version of the consistent predicate from the formal memory model.

CPPMEM exhaustively enumerates all candidate executions (for a given program) and tests for consistency.
Nitpick is a generic model finder for higher-order logic.

Nitpick translates the formal memory model—together with the constraints imposed by a given program—into first-order relational logic (→ Kodkod) and then into SAT.
Example: Write-to-Read Causality

```cpp
atomic_int x = 0, y = 0;

{{{
    x.store(1, rlx);
    x.load(rlx);    // = 1
    y.store(1, rlx);
    y.load(rlx);    // = 1
    x.load(rlx);    // = 0
}}}
```
## Write-to-Read Causality: Performance

<table>
<thead>
<tr>
<th>Locations ($n$)</th>
<th>Actions ($3n + 1$)</th>
<th>States ($2^{3(3n+1)^2}$)</th>
<th>CPPMEM</th>
<th>NITPICK</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>relaxed</td>
<td>SC</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>$2^{147}$</td>
<td>0.0 s</td>
<td>0.5 s</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>$2^{300}$</td>
<td>0.0 s</td>
<td>90.5 s</td>
</tr>
<tr>
<td>4</td>
<td>13</td>
<td>$2^{507}$</td>
<td>0.1 s</td>
<td>$&gt;10^4$ s</td>
</tr>
<tr>
<td>5</td>
<td>16</td>
<td>$2^{768}$</td>
<td>0.2 s</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>19</td>
<td>$2^{1083}$</td>
<td>0.7 s</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>22</td>
<td>$2^{1452}$</td>
<td>2.5 s</td>
<td></td>
</tr>
</tbody>
</table>
Since 2011, C and C++ have a memory model.

We have

- a formal (machine-readable, executable) version of this memory model and
- automatic tools to explore the behavior of small programs.
Future Challenges

Use the model!

- Compiler correctness
- Program transformations
- Static analysis
- Dynamic analysis

- Program logics
- Formal verification
- Equivalent models
- ...