Tools for exploring and understanding memory models

John Wickerson, Imperial College

Joint work with
Jade Alglave (UCL), Mark Batty (Cambridge), Alastair Donaldson (Imperial), Ganesh Gopalakrishnan (Utah), Luc Maranget (INRIA), Daniel Poetzl (Oxford) and Tyler Sorensen (Utah)
Outline

• Memory models
• Lem
• Herd
Memory models

• Memory model:
  • which values might be read from memory
  • assembly code / higher-level language
  • operational / axiomatic
Axiomatic memory model

• Defines predicates over program executions

• Contract between programmer and language

The value of an atomic object \( M \), as determined by evaluation \( B \), shall be the value stored by some operation in the visible sequence of \( M \) with respect to \( B \).

C standard §5.1.2.4:22

The execution of a program contains a data race if it contains two conflicting actions in different threads, at least one of which is not atomic, and neither happens before the other. Any such data race results in undefined behavior.

C standard §5.1.2.4:25
A visible side effect \( A \) on an object \( M \) with respect to a value computation \( B \) of \( M \) satisfies the conditions:

- \( A \) happens before \( B \), and
- there is no other side effect \( X \) to \( M \) such that \( A \) happens before \( X \) and \( X \) happens before \( B \).

**NOTE 8** If there is ambiguity about which side effect to a non-atomic object is visible, then there is a data race and the behavior is undefined.

**NOTE 9** This states that operations on ordinary variables are not visibly reordered. This is not actually detectable without data races, but it is necessary to ensure that data races, as defined here, and with suitable restrictions on the use of atomics, correspond to data races in a simple interleaved (sequentially consistent) execution.

**NOTE 10** This effectively disallows compiler reordering of atomic operations to a single object, even if both operations are “relaxed” loads. By doing so, we effectively make the “cache coherence” guarantee provided by most hardware available to C atomic operations.

**NOTE 11** The visible sequence depends on the “happens before” relation, which in turn depends on the values observed by loads of atomics, which we are restricting here. The intended reading is that there must exist an association of atomic loads with modifications they observe that, together with suitably chosen modification orders and the “happens before” relation derived as described above, satisfy the resulting constraints as imposed here.

**NOTE 12** It can be shown that programs that correctly use simple mutexes and \texttt{memory\_order\_seq\_cst} operations to prevent all data races, and use no other synchronization operations, behave as though the operations executed by their constituent threads were simply interleaved, with each value computation of an object being the last value stored in that interleaving. This is normally referred to as “sequential consistency”. However, this applies only to data-race-free programs, and data-race-free programs cannot observe most program transformations that do not change single-threaded program semantics. In fact, most single-threaded program transformations continue to be allowed, since any program that behaves differently as a result must contain undefined behavior.

**let** visible\_side\_effect\_set \texttt{actions} \texttt{hb} =

\{ (A,B) | forall ((A,B) \texttt{IN} \texttt{hb}) | is\_write A && is\_read B && (loc\_of A = loc\_of B) && not (exists (X \texttt{IN} \texttt{actions}). not (X \texttt{IN} \{A;B\}) && is\_write X && (loc\_of X = loc\_of B) && (A,X) \texttt{IN} \texttt{hb} && (X,B) \texttt{IN} \texttt{hb}) \}
A visible side effect $A$ on an object $M$ with respect to a value computation $B$ of $M$ satisfies the conditions:

- $A$ happens before $B$, and
- there is no other side effect $X$ to $M$ such that $A$ happens before $X$ and $X$ happens before $B$.
Herd

\[(a,b) \in R \quad (b,c) \in S\]
\[\therefore (a,c) \in R \quad ; \quad S\]

\[a \in s \quad \Rightarrow (a,a) \in [s]\]

R & S \quad R \mid S \quad R ; S \quad [s]

\sim R \quad R^{-1} \quad R^* \quad R+ \quad R?
let hbl = hb & loc
let vse = [W];hbl;[R] & ~(hbl;[W];hbl)
let hbl = hb & loc
let vse = [W];hbl;[R] & ~(hbl;[W];hbl)
The POWER of Herd
Lem vs. Herd

• Lem:
  • Expressive language $\rightarrow$ match prose

• Herd:
  • Concise $\rightarrow$ optimise model, compare models
  • Efficient simulation
Our work

• C/C++ memory model in Herd, prove equivalent to existing Lem version

• OpenCL memory model in Lem and Herd

• NVIDIA's PTX memory model
Tools for exploring and understanding memory models

John Wickerson, Imperial College

Joint work with
Jade Alglave (UCL), Mark Batty (Cambridge), Alastair Donaldson (Imperial), Ganesh Gopalakrishnan (Utah), Luc Maranget (INRIA), Daniel Poetzl (Oxford) and Tyler Sorensen (Utah)