C/C++ Causal Cycles Confound Compositionality

ABSTRACT

The rise of multicore processors has made concurrency ubiquitous. In response, mainstream languages have begun to offer primitives for concurrent programming. To avoid the cost of inter-core synchronisation, the new C/C++ standard, C11 [2, 3], offers weakly consistent *relaxed* operations, alongside traditional reads, writes and mutexes. When using relaxed operations, different threads may see different, apparently contradictory orders of events.

C11 permits a particularly suprising kind of relaxed behaviour: *cycles in causality*. Two conditional branches on different threads may be satisfied by writes down the opposite thread's branch. That is, neither branch could be taken if the other didn't occur – an apparent paradox.

```
          x = 0; y = 0;
if (x == 42)  ||  if (y == 23)
  y = 23;     ||    x = 42;
```

Such cycles could arise from hardware speculation or compiler optimisations; however, it is unclear whether they occur on current implementations. They are known to be problematic: the C11 standard heavily deprecates them, but to allow certain important optimisations, it falls short of banning them entirely.

A property is *compositional* if each program sub-components can be analysed separately, while assuming its surrounding context is well-behaved. By allowing programs to be decomposed, compositionality aids documentation, testing and verification. In most languages, safety properties (e.g., absence of memory faults) are compositional, because a given fault must originate in the sub-component or its context, but not both. Causal cycles in C11 allow two faults to cause each other, which violates this assumption and breaks compositionality [1]. This is probably undesirable.

BODY

C/C++ permit seemingly-impossible cycles in causality. This breaks compositionality: two apparently safe programs may fault when composed.

REFERENCES

[1] M. Batty, M. Dodds, and A. Gotsman. Library abstraction for C/C++ concurrency. In *Principles of Programming Languages*, 2013.

[2] M. Batty, S. Owens, S. Sarkar, P. Sewell, and T. Weber. Mathematizing C++ concurrency. In *Principles of Programming Languages*, 2011.

[3] ISO/IEC. *Programming Languages – C, 9899:2011.*

Volume 1 of Tiny Transactions on Computer Science