Problem of Programming Language Concurrency Semantics

Mark Batty, Kayvan Memarian, Kyndylan Nienhuis, Jean Pichon-Pharabod, Peter Sewell

Presentation by
Akshay Gopalakrishnan

McGill University

July 4, 2021

Introduction/Goals

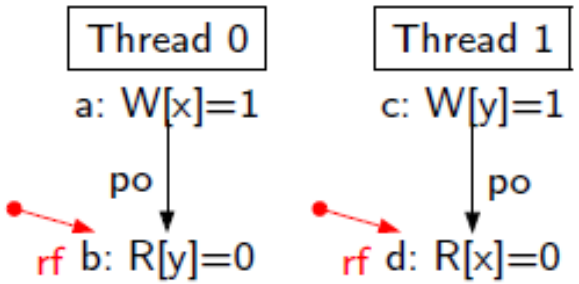
- An overview of the current problems in semantically defining relaxed memory models.
- Give mechanized proof for DRF-SC property of C11 (HOL4).
- Thin air executions in C11 and its lack of understanding.
- Proof that per-candidate execution model of C11 cannot disallow thin air executions.
- Operational model disallowing thin air behaviors but restricting compiler optimizations.
- Difficulty in defining the C11 notion of undefined behavior.

Sequential Consistency - Examples

Simplest model of reasoning. Interleaving semantics. Informally,

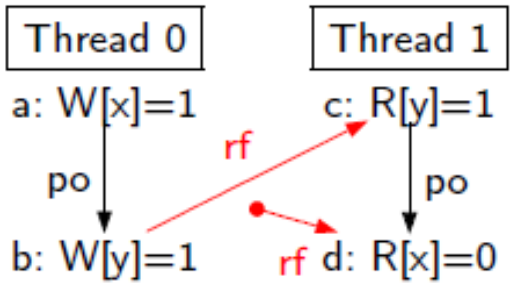
A concurrent program's outcome is equivalent to the same program being run in a uni-processor environment.
Note: Hardware exhibits Non-SC behaviors that give us better performance.

Store Buffering



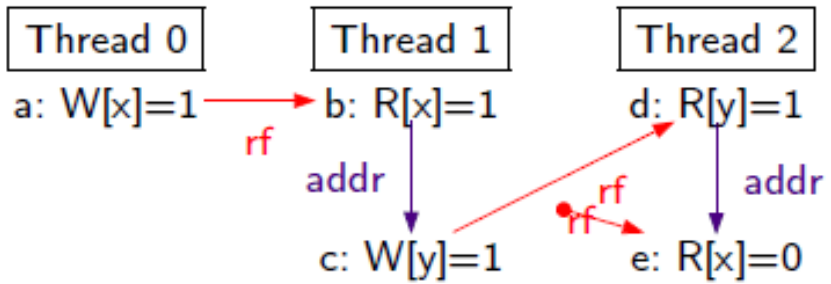
Test SB: Allowed on x86, Power, and ARM

Message Passing



Test MP: Allowed on Power and ARM

Non-Multi-Copy Atomicity



Test WRC+addrs: Allowed on Power and ARM

Data Races

Such relaxed behaviors can give rise to what is known as Data Race.

Consider timelines of two events operating on the same memory.

——————————-A

———-B

Behaviors beyond A-then-B or B-then-A will be present in Non-SC executions. Now difficult to reason with programs.

DRF-SC model

Statement:

If a program exhibits **no** data race in its SC executions, then it is guaranteed to have **only** SC behaviors.
Useful property for models of High Level Languages. Is central to the C11 memory model.

High Level Language models - C11

- Axiomatic model.
- Per-execution semantics.
- Description of disallowed behaviors using partial orders between program events in an execution.
- Has multiple degrees of atomicity - $RLX \geq REL \geq ACQ \geq CON \geq SC$.
- Has various partial order definitions between program events - rf, sb, hb, mo.
- Exhibits DRF-SC property for programs not using low-level atomics (only SC).

DRF-SC proof Theorem for C11 memory model

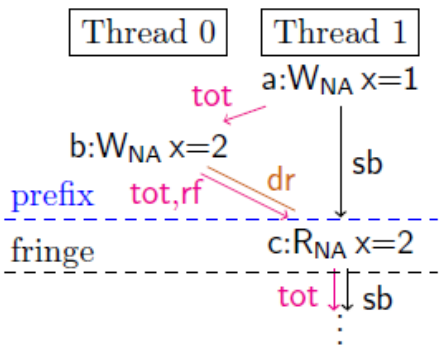
Theorem 1. *For programs whose pre-executions (i) use only mutex, non-atomic and SC-atomic accesses, (ii) have atomic initialisations ordered by sequenced-before and parent-to-child thread synchronisation before all atomic accesses at the same location, and (iii) are bounded in size by some N, either both the C/C++11 model and the total model give undefined behaviour, or the sets of consistent executions in each, projected down to the pre-execution and the reads-from relation, are equal.*

The total model captures only the SC semantics of the C11 memory model. "Total" here being it is defined using only one total order between all events.

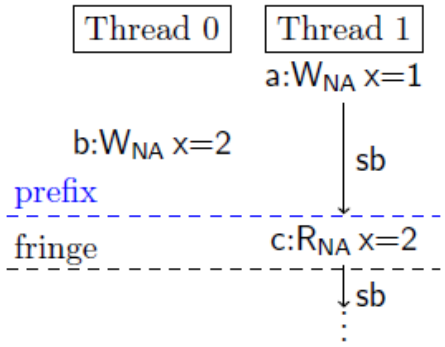
For every consistent execution in the C11 model, there exists a consistent execution in the Total model with the same reads-from relation.

For every consistent execution in the Total model, there exists a consistent execution in the C11 model with the same reads-from relation.

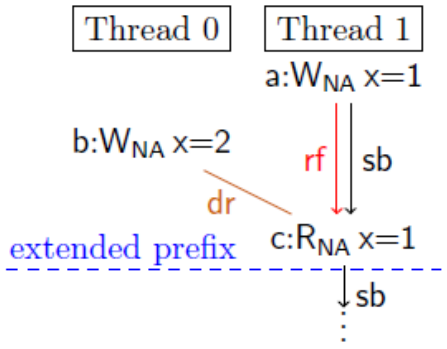
For every racy execution in the C11 model, there exists a racy execution in the total model.
The read values may not be the same, as the below example shows.



A racy execution in the total model



The bare pre-execution



Extended racy execution in the C/C++11 model

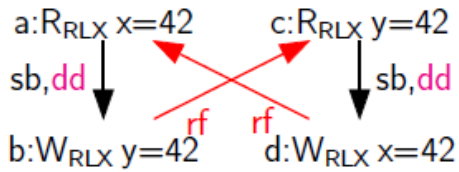
Note: It is okay to have another racy execution with different read values as program exhibits undefined behavior anyways.

Thin Air problem (also known as Out-of-thin-air(OOTA))

Even though we restrict data-races in a program, semantically it is difficult to avoid certain outcomes that must be forbidden. For instance, the following outcome should be disallowed:

Example LB+datas (language can and should forbid)

```
r1=load_rlx(x) //reads 42
store_rlx(y,r1)
-----------------------
r2=load_rlx(y) //reads 42
store_rlx(x,r2)
```

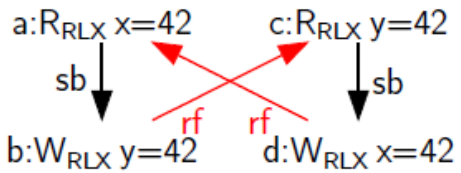


OOTA cnt'd

But disallowing the previous execution, will also disallow the following load buffering (load-store reordering) case:

Example LB (language must allow)

```
r1=load_rlx(x) //reads 42
store_rlx(y,42)
-----------------------
r2=load_rlx(y) //reads 42
store_rlx(x,42)
```

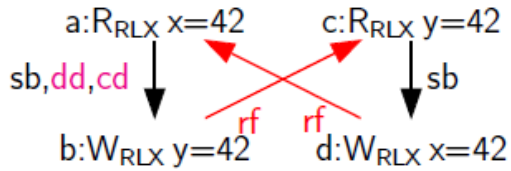


OOTA cnt'd

Some variants, after transformations, the outcomes should be allowed.

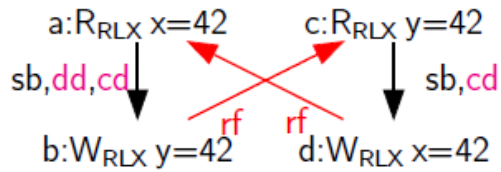
Example LB+ctrldata+po (language must allow)

```
r1=load_rlx(x) //reads 42
if (r1 == 42)
  store_rlx(y,r1)
-----------------------
r2=load_rlx(y) //reads 42
store_rlx(x,42)
```



Example LB+ctrldata+ctrl-double (language must allow)

```
r1=load_rlx(x) //reads 42
if (r1 == 42)
  store_rlx(y,r1)
-----------------------
r2=load_rlx(y) //reads 42
if (r2 == 42)
  store_rlx(x,42)
else
  store_rlx(x,42)
```



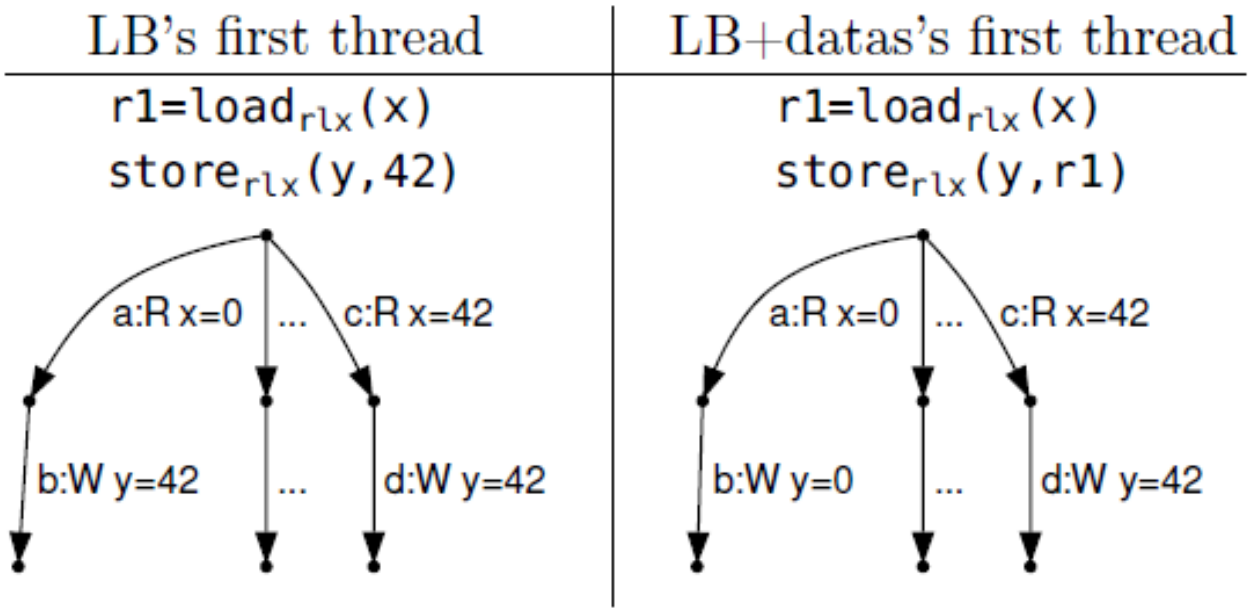
Difficulty in semantically defining OOTA using per-candidate execution model

The above variants of programs have the exact candidate execution pattern in C11 style semantics. Hence, the possibility of semantically disallowing OOTA using this style is not possible.

Theorem 2. *No adaptation of the C/C++11 per-candidate-execution definition that uses the same notion of candidate execution can give the desired behaviour for both of these examples.*

Alterative solution - Operational model

A labelled transition system for each thread, depicting thread-local semantics. Has all possible reachable states.



Operational model - cnt'd

States are LTS in-order with some edges *ticked*. A set of edges can be "ticked" iff:

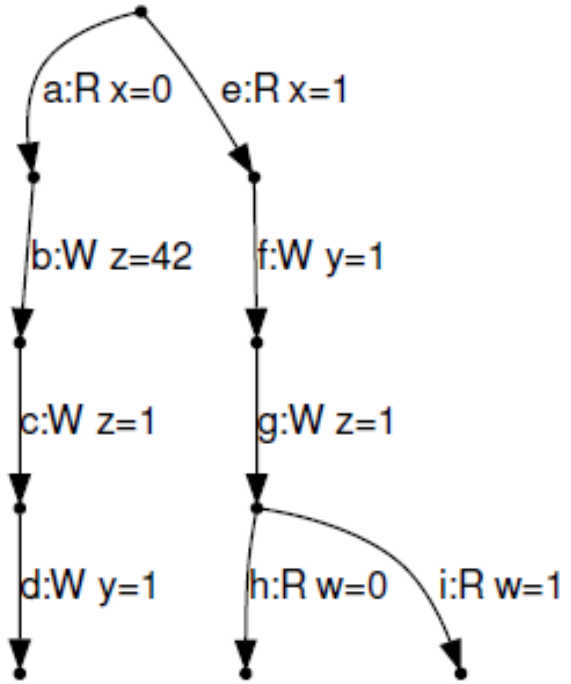
- Set of edges is non-empty.
- The edges themselves are not ticked.
- Edges have the same memory action label.
- Each non-discarded path is either having its event in the set being ticked, or the path discarded due to tick.
- No edge is blocked (due to control dependency for example).

The above semantics would cover all our previous examples of LB and its variants.

Problem of Compiler Optimizations

Irrelevant read elimination:

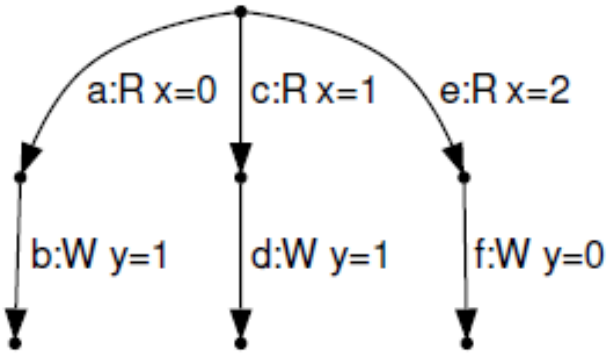
```
r1=load_rlx(x)
if (r1 == 1) {
  store_rlx(y,1)
  store_rlx(z,1)
  r2=load_rlx(w)
} else {
  store_rlx(z,42)
  store_rlx(z,1)
  store_rlx(y,1)
}
```

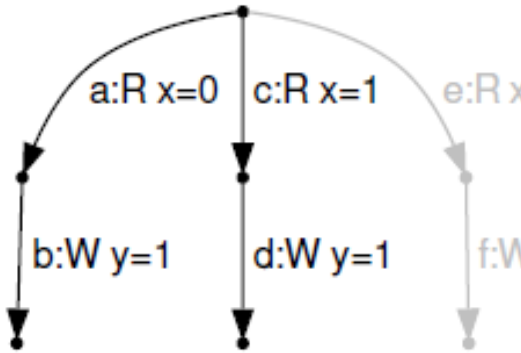


Problem of Compiler optimizations - cnt'd

Inter-thread optimizations

```
r1=load_rlx(x)
if (r1 == 2)
  store_rlx(y,0)
else
  store_rlx(y,1)
```





"Undefined" in C/C11

Undefined literally gives no semantics for the programs in C11.
Examples of programs that would exhibit undefined behavior are:

- Out of bounds array accesses.
- Divide by zero.
- Programs with data races.

Example asserting Divide-by-zero

```
for(int i=0; i<5; i++) {
  puts("foo\n");
  ret += i + 1/x;
}
```

Loop invariant code motion is possible, due to x being zero. Thus, the compiler is free to move $1/x$ outside the loop itself, as a form of optimization.

Example asserting OOBA in a concurrent program

$r1 = load_{rlx}(x)$ $r3 = a[r1]$ $store_{rlx}(y,2)$	$r2 = load_{rlx}(y)$ $store_{rlx}(x,r2)$

SC semantics would not give OOBA. // But C11 semantics pertaining ARM/POWER H/W would ensure OOBA (through means of Load Buffering), thus undefined behavior.

Conclusion

1. Hardware exhibits relaxed memory behaviors for better performance.
2. C11 semantics inevitably allows thin-air executions.
3. No per-candidate C11 execution style can capture thin-air execution semantics.
4. An operational model to remedy it inevitably asks for taking into account of all possible compiler optimizations.
5. Lack of semantics for C11 Undefined behavior makes it difficult to reason about programs and impact on it due to compiler optimizations.

Thank you

Questions?