

High-level languages, compilers, multiprocessors... an elusive mix?

Francesco Zappa Nardelli

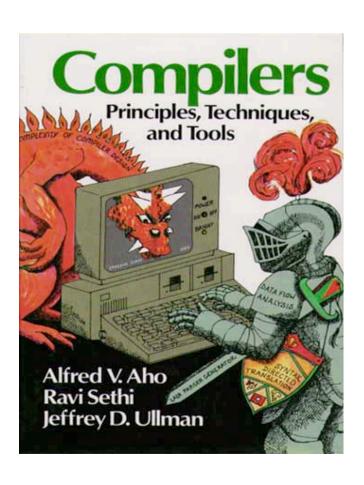
INRIA Paris

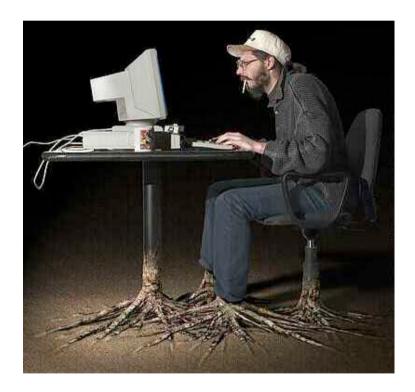
http://www.di.ens.fr/~zappa/projects/weakmemory

Based on work done by or with

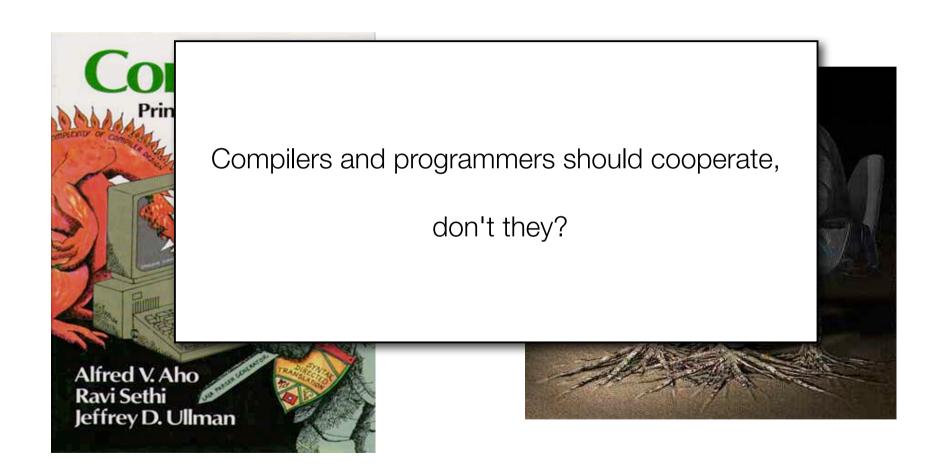
Peter Sewell, Jaroslav Ševčík, Susmit Sarkar, Tom Ridge, Scott Owens, Viktor Vafeiadis, Magnus O. Myreen, Kayvan Memarian, Luc Maranget, Derek Williams, Pankaj Pawan, Thomas Braibant, Mark Batty, Jade Alglave.

Compilers vs. programmers





Compilers vs. programmers



Constant propagation (an optimising compiler breaks your program)

A simple and innocent looking optimization:

int
$$x = 14$$
;
int $y = 7 - x / 2$;
int $y = 7 - 14 / 2$;

3

Constant propagation (an optimising compiler breaks your program)

A simple and innocent looking optimization:

int
$$x = 14$$
;
int $y = 7 - x / 2$;
int $y = 7 - 14 / 2$;

Consider the two threads below:

Intuitively, this program always prints 0

Constant propagation (an optimising compiler breaks your program)

A simple and innocent looking optimization:

int
$$x = 14$$
;
int $y = 7 - x / 2$;
int $y = 7 - 14 / 2$;

Consider the two threads below:

Sun HotSpot JVM or GCJ: always prints 1.

Background: lock and unlock

Suppose that two threads increment a shared memory location:

• If both threads read 0, (even in an ideal world) x == 1 is possible:

```
tmp1 = *x; tmp2 = *x; *x = tmp1 + 1; *x = tmp2 + 1
```

Background: lock and unlock

 Lock and unlock are primitives that prevent the two threads from interleaving their actions.

```
lock();
tmp1 = *x;
*x = tmp1 + 1;
unlock();
lock();
tmp2 = *x;
*x = tmp2 + 1;
unlock();
```

• In this case, the interleaving below is forbidden, and we are guaranteed that x == 2 at the end of the execution.

```
top the temp1 = *x; tmp2 = *x; *x = tmp1 + 1; *x = tmp2 +1
```

Lazy initialisation (an unoptimising compiler breaks your program)

Deferring an object's initialisation util first use: a big win if an object is never used (e.g. device drivers code). Compare:

The singleton pattern

Lazy initialisation is a pattern commonly used. In C++ you would write:

But this code is not thread safe! Why?

Making the singleton pattern thread safe

A simple thread safe version:

```
class Singleton {
public:
    static Singleton *instance (void) {
        Guard<Mutex> guard (lock_); // only one thread at a time
        if (instance_ == NULL)
            instance_ = new Singleton;
        return instance_;
    }
private:
    static Mutex lock_;
    static Singleton *instance_;
};
```

Every call to instance must acquire and release the lock: excessive overhead.

Obvious (broken) optimisation

```
class Singleton {
public:
    static Singleton *instance (void) {
    if (instance_ == NULL) {
        Guard<Mutex> guard (lock_); // lock only if unitialised
        instance_ = new Singleton; }
    return instance_;
}

private:
    static Mutex lock_;
    static Singleton *instance_;
};
```

Exercise: why is it broken?

Clever programmers use double-check locking

```
class Singleton {
public:
 static Singleton *instance (void) {
  // First check
  if (instance == NULL) {
    // Ensure serialization
    Guard<Mutex> guard (lock );
    // Double check
    if (instance == NULL)
       instance = new Singleton;
  return instance;
private: [..]
};
```

Idea: re-check that the Singleton has not been created after acquiring the lock.

12

lec2 - 1 February 2018

Double-check locking: clever but broken

The instruction

```
instance_ = new Singleton;
```

does three things:

- 1) allocate memory
- 2) construct the object
- 3) assign to instance_ the address of the memory

Not necessarily in this order! For example:

If this code is generated, the order is 1,3,2.

Broken...

Thread 1:

executes through Line 2 and is suspended; at this point, instance_ is non-NULL, but no singleton has been constructed.

Thread 2:

executes Line 1, sees instance_ as non-NULL, returns, and dereferences the pointer returned by Singleton (i.e., instance_).

Thread 2 attempts to reference an object that is not there yet!

The fundamental problem

Problem: You need a way to specify that step 3 come after steps 1 and 2.

There is no way to specify this in C++

Similar examples can be built for any programming language...

That pesky hardware (1)

Consider misaligned 4-byte accesses

(Disclaimer: compiler will normally ensure alignment)

Intel SDM x86 atomic accesses:

- n-bytes on an n-byte boundary (n = 1,2,4,16)
- P6 or later: ... or if unaligned but within a cache line

Question: what about multi-word high-level language values?

That pesky hardware (1)

Consider misaligned 4-byte accesses

int32_t a = 0

$$a = 0x44332211$$

if (a == 0x00002211)

print "error"

(Disclaime

Intel SDN

- n-byte
- P6 or I

This is called a *out-of-thin air read*:

the program reads a value

that the programmer never wrote.

Question: what about multi-word high-level language values

That pesky hardware (2)

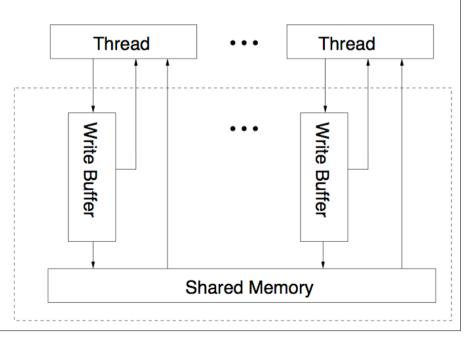
Hardware optimisations can be observed by concurrent code:

| Thread 0 | Thread 1 |
|----------|----------|
| x = 1 | y = 1 |
| print y | print x |

At the end of some executions:

0 0

is printed on the screen, both on x86 and Power/ARM).



That pesky hardware (2)

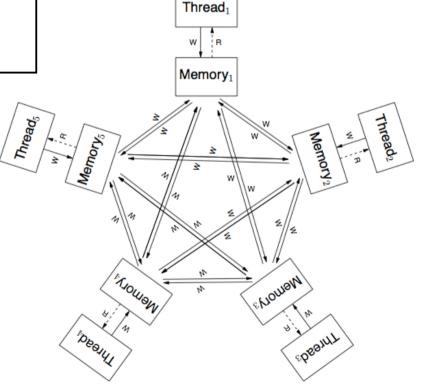
...and differ between architectures...

| Thread 0 | Thread 1 |
|----------|----------|
| x = 1 | print y |
| y = 1 | print x |

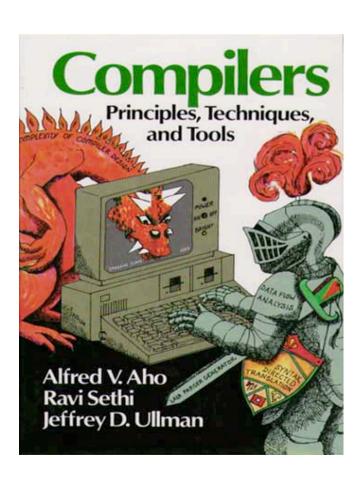
At the end of some executions:

1 0

is printed on the screen on Power/ARV but not on x86.



Compilers vs. programmers





Compilers vs. programmers

Tension:

- the programmer wants to understand the code he writes
- the compiler and the hardware want to optimise it.

Which are the valid optimisations that the compiler or the hardware can perform without breaking the expected semantics of a concurrent program?

Which is the semantics of a concurrent program?

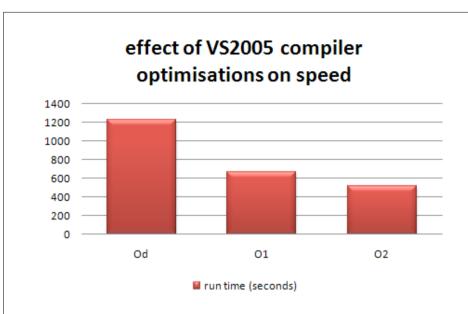
This lecture

Programming language models

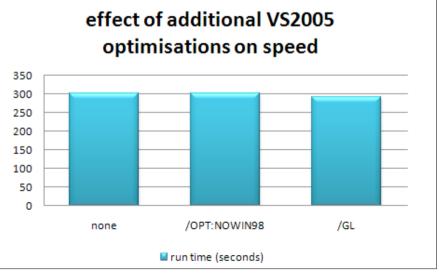
- 1) defining the semantics of a concurrent programming language
- 2) data-race freedom
- 3) soundness of compiler optimisations

Previous lecture: hardware models

- 1) why are industrial specs so often flawed? focus on x86, with a glimpse of Power/ARM
- 2) usable models: x86-TSO, PowerARM



A brief tour of compiler optimisations



World of optimisations

A typical compiler performs many optimisations.

gcc 4.4.1. with -02 option goes through 147 compilation passes.

computed using -fdump-tree-all and -fdump-rtl-all

Sun Hotspot Server JVM has 18 high-level passes with each pass composed of one or more smaller passes.

http://www.azulsystems.com/blog/cliff-click/2009-04-14-odds-ends

World of optimisations

A typical compiler performs many optimisations.

- Common subexpression elimination (copy propagation, partial redundancy elimination, value numbering)
- (conditional) constant propagation
- dead code elimination
- loop optimisations
 (loop invariant code motion, loop splitting/peeling, loop unrolling, etc.)
- vectorisation
- peephole optimisations
- tail duplication removal
- building graph representations/graph linearisation
- register allocation
- call inlining
- local memory to registers promotion
- spilling
- instruction scheduling

World of optimisations

However only some optimisations change shared-memory traces:

- Common subexpression elimination
 (copy propagation, partial redundancy elimination, value numbering)
- (conditional) constant propagation
- dead code elimination
- loop optimisations
 (loop invariant code motion), loop splitting/peeling, loop unrolling, etc.)
- vectorisation
- peephole optimisations
- tail duplication removal
- building graph representations/graph linearisation
- register allocation
- call inlining
- local memory to registers promotion
- spilling
- instruction scheduling

Compiler Writer





Compiler Writer



Sophisticated program analyses Fancy algorithms Source code or IR

Operations on AST



Compiler Writer



Sophisticated program analyses Fancy algorithms Source code or IR

Operations on AST



```
for (int i=0; i<2; i++) {
  z = i;
  x[i] += y+1;
}</pre>
```

Compiler Writer



Sophisticated program analyses Fancy algorithms Source code or IR

Operations on AST



```
tmp = y+1;
for (int i=0; i<2; i++) {
  z = i;
  x[i] += tmp;
}</pre>
```

Compiler Writer



Sophisticated program analyses Fancy algorithms Source code or IR

Operations on AST

Semanticist



Elimination of run-time events Reordering of run-time events Introduction of run-time events

Operations on sets of events

```
tmp = y+1;
for (int i=0; i<2; i++) {
  z = i;
  x[i] += tmp;
}</pre>
```

Compiler Writer



Sophisticated program analyses Fancy algorithms Source code or IR

Operations on AST

tmp = y+1; for (int i=0; i<2; i++) { z = i; x[i] += tmp; }</pre>

Semanticist



Elimination of run-time events Reordering of run-time events Introduction of run-time events

Operations on sets of events

```
Store z 0
Load y 42
Store x[0] 43
Store z 1
Load y 42
Store x[1] 43
```

Compiler Writer



Sophisticated program analyses Fancy algorithms Source code or IR

Operations on AST

```
tmp = y+1;
for (int i=0; i<2; i++) {
  z = i;
  x[i] += tmp;
}</pre>
```

Semanticist



Elimination of run-time events Reordering of run-time events Introduction of run-time events

Operations on sets of events

Store
$$x[1]$$
 43

Eliminations

This includes common subexpression elimination, dead read elimination, overwritten write elimination, redundant write elimination.

Irrelevant read elimination:

$$r=*x; C \rightarrow C$$

where r is not free in c.

Redundant read after read elimination:

$$r1=*x$$
; $r2=*x \rightarrow r1=*x$; $r2=r1$

Redundant read after write elimination:

$$*x=r1; r2=*x \rightarrow *x=r1; r2=r1$$

Reordering

Common subexpression elimination, some loop optimisations, code motion.

Normal memory access reordering:

```
r1=*x; r2=*y → r2=*y; r1=*x

*x=r1; *y=r2 → *y=r2; *x=r1

r1=*x; *y=r2 ⇌ *y=r2; r1=*x
```

Roach motel reordering:

```
memop; lock m → lock m; memop
unlock m; memop → memop; unlock m
where memop is *x=r1 or r1=*x
```

Memory access introduction

Can an optimisation introduce memory accesses?

Yes, but rarely:

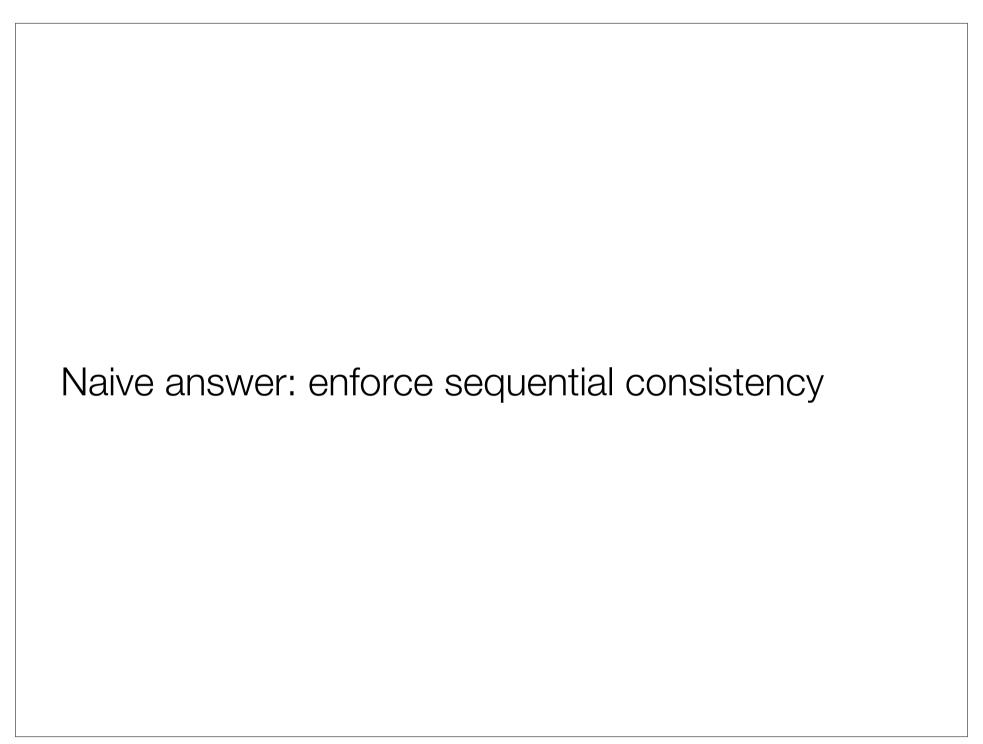
Note that the loop body is not executed.

Memory access introduction

Back to our question now:

Which is the semantics of a concurrent program?

Trole that the loop body is not executed.

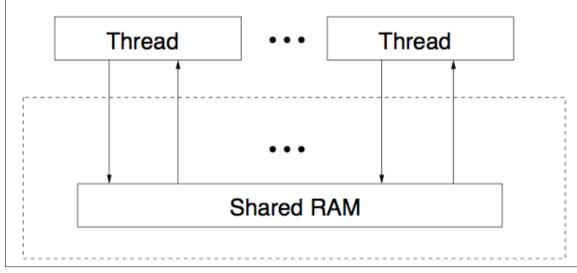


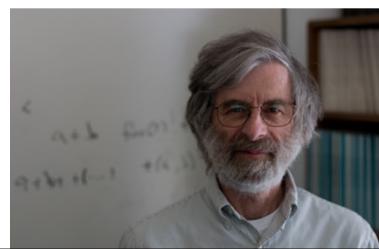
Sequential consistency

Multiprocessors have a *sequentially consistent* shared memory when:

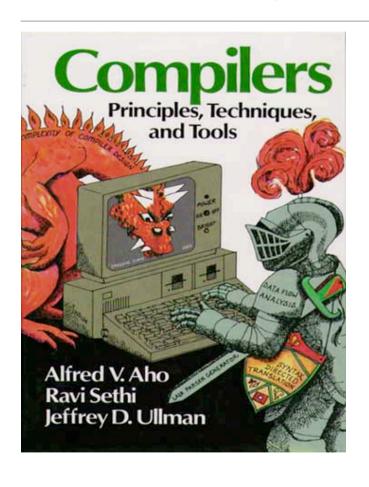
...the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program...

Lamport, 1979.



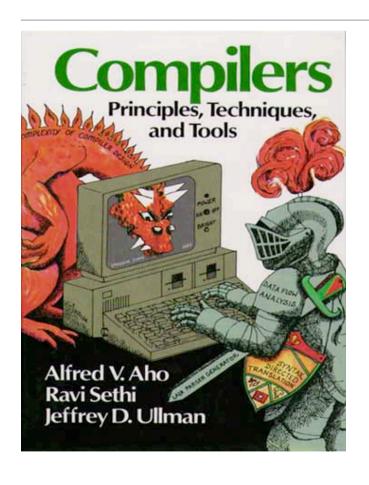


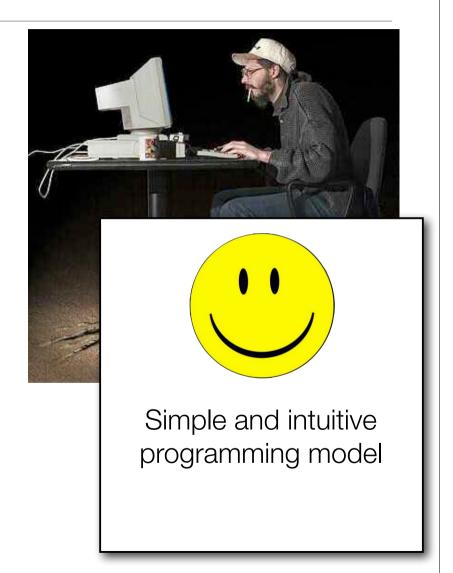
Compilers, programmers & sequential consistency



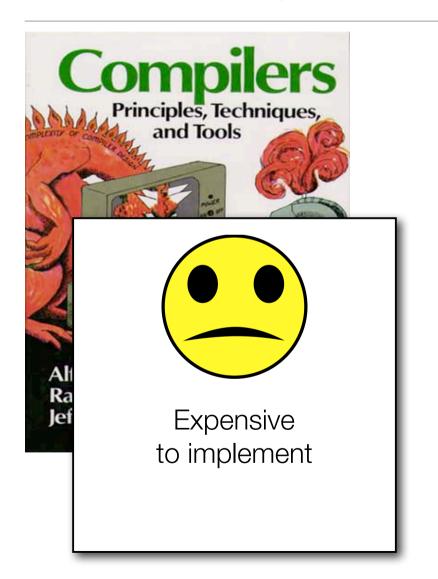


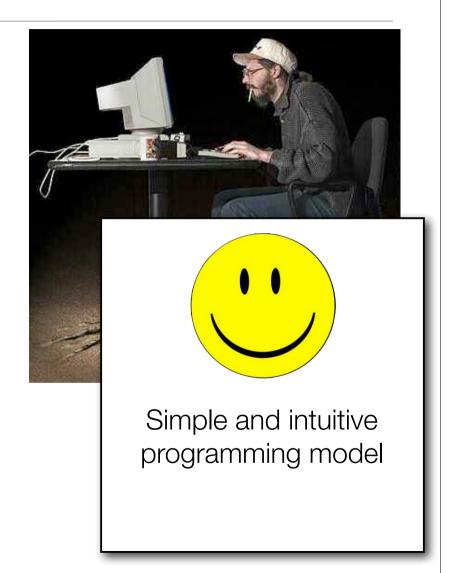
Compilers, programmers & sequential consistency





Compilers, programmers & sequential consistency





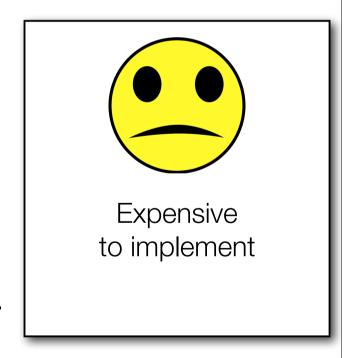
A Case for an SC-Preserving Compiler

Daniel Marino[†] Abhayendra Singh* Todd Millstein[†] Madanlal Musuvathi[‡] Satish Narayanasamy*

[†]University of California, Los Angeles *University of Michigan, Ann Arbor [‡]Microsoft Research, Redmond

An SC-preserving compiler, obtained by restricting the optimization phases in LLVM, a state-of-the-art C/C++ compiler, incurs an average slowdown of 3.8% and a maximum slowdown of 34% on a set of 30 programs from the SPLASH-2, PARSEC, and SPEC CINT2006 benchmark suites.

And this study supposes that the hardware is SC.



SC and hardware

The compiler must insert enough synchronising instructions to prevent hardware reorderings. On x86 we have:

- MFENCE flush the local write buffer
- LOCK prefix (e.g. CMPXCHG) flush the local write buffer globally lock the memory

| Initial: [x]=0 ∧ [y]=0 | |
|------------------------|-------------|
| proc 0 | proc 1 |
| MOV [x]←\$1 | MOV [y]←\$1 |
| MFENCE | MFENCE |
| MOV EAX←[y] | MOV EBX←[x] |
| Forbid: EAX=0 ∧ EBX=0 | |

Initally, [100] = 0At the end, [100] = 2

| proc:0 | proc:1 |
|-----------------|-----------------|
| LOCK; INC [100] | LOCK; INC [100] |

These consumes hundreds of cycles... ideally should be avoided.

Naively recovering SC on x86 incurs in a ~40% overhead.

A Case for an SC-Preserving Compiler

Daniel Marino[†] Todd Millstein[†] Abhayendra Singh* Madanlal Musuvathi[‡] Satish Narayanasamy* [‡]Microsoft Research, Redmond [†]University of California, Los Angeles *University of Michigan, Ann Arbor

An SC-preserving compiler, obtained by restricting the optimization phases in LLVM, a state-of-the-art C/C++ compiler, incurs an average slowdown of 3.8% and a maximu

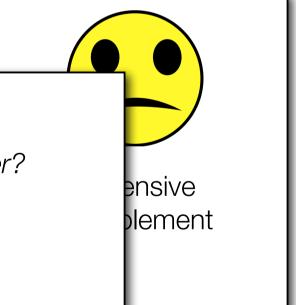
program

and SPF

What is an SC-preserving compiler?

When is a compiler correct?

And this st



When is a compiler correct?

A compiler is correct if any behaviour of the compiled program could be exhibited by the original program.

i.e. for any execution of the compiled program, there is an execution of the source program with the same observable behaviour.

Intuition: we represent programs as sets of memory action traces, where the trace is a sequence of memory actions of a single thread.

Intuition: the observable behaviour of an execution is the subtrace of external actions.

Is the transformation from P1 to P2 correct (in an SC semantics)?

Executions of P1:

$$\begin{aligned} & \mathsf{W}_{t_1} \, x{=}1, \mathsf{R}_{t_2} \, x{=}1, \mathsf{R}_{t_2} \, x{=}1, \mathsf{P}_{t_2} \, 1 \\ & \mathsf{R}_{t_2} \, x{=}0, \mathsf{W}_{t_1} \, x{=}1, \mathsf{R}_{t_2} \, x{=}1, \mathsf{P}_{t_2} \, 2 \\ & \mathsf{R}_{t_2} \, x{=}0, \mathsf{R}_{t_2} \, x{=}0, \mathsf{W}_{t_1} \, x{=}1, \mathsf{P}_{t_2} \, 1 \\ & \mathsf{R}_{t_2} \, x{=}0, \mathsf{R}_{t_2} \, x{=}0, \mathsf{P}_{t_2} \, 1, \mathsf{W}_{t_1} \, x{=}1 \end{aligned}$$

Executions of P1:

$$\begin{aligned} & \mathsf{W}_{t_1} \, x{=}1, \mathsf{R}_{t_2} \, x{=}1, \mathsf{R}_{t_2} \, x{=}1, \mathsf{P}_{t_2} \, 1 \\ & \mathsf{R}_{t_2} \, x{=}0, \mathsf{W}_{t_1} \, x{=}1, \mathsf{R}_{t_2} \, x{=}1, \mathsf{P}_{t_2} \, 2 \\ & \mathsf{R}_{t_2} \, x{=}0, \mathsf{R}_{t_2} \, x{=}0, \mathsf{W}_{t_1} \, x{=}1, \mathsf{P}_{t_2} \, 1 \\ & \mathsf{R}_{t_2} \, x{=}0, \mathsf{R}_{t_2} \, x{=}0, \mathsf{P}_{t_2} \, 1, \mathsf{W}_{t_1} \, x{=}1 \end{aligned}$$

Executions of P2:

$$\begin{aligned} & \mathsf{W}_{t_1} \, x{=}1, \mathsf{R}_{t_2} \, x{=}1, \mathsf{P}_{t_2} \, 1 \\ & \mathsf{R}_{t_2} \, x{=}0, \mathsf{W}_{t_1} \, x{=}1, \mathsf{P}_{t_2} \, 1 \\ & \mathsf{R}_{t_2} \, x{=}0, \mathsf{P}_{t_2} \, 1, \mathsf{W}_{t_1} \, x{=}1 \end{aligned}$$

Executions of P1:

$$\begin{aligned} & \mathsf{W}_{t_1} \, x{=}1, \mathsf{R}_{t_2} \, x{=}1, \mathsf{R}_{t_2} \, x{=}1, \mathsf{P}_{t_2} \, 1 \\ & \mathsf{R}_{t_2} \, x{=}0, \mathsf{W}_{t_1} \, x{=}1, \mathsf{R}_{t_2} \, x{=}1, \mathsf{P}_{t_2} \, 2 \\ & \mathsf{R}_{t_2} \, x{=}0, \mathsf{R}_{t_2} \, x{=}0, \mathsf{W}_{t_1} \, x{=}1, \mathsf{P}_{t_2} \, 1 \\ & \mathsf{R}_{t_2} \, x{=}0, \mathsf{R}_{t_2} \, x{=}0, \mathsf{P}_{t_2} \, 1, \mathsf{W}_{t_1} \, x{=}1 \end{aligned}$$

Executions of P2:

$$\begin{aligned} & \mathsf{W}_{t_1} \, x{=}1, \mathsf{R}_{t_2} \, x{=}1, \mathsf{P}_{t_2} \, 1 \\ & \mathsf{R}_{t_2} \, x{=}0, \mathsf{W}_{t_1} \, x{=}1, \mathsf{P}_{t_2} \, 1 \\ & \mathsf{R}_{t_2} \, x{=}0, \mathsf{P}_{t_2} \, 1, \mathsf{W}_{t_1} \, x{=}1 \end{aligned}$$

Behaviours of P1: $[P_{t_2} 1], [P_{t_2} 2]$

Behaviours of P2: $[P_{t_2} 1]$

$$P_1 = *x = 1$$
 | r1 = *x; r2 = *x;
if r1=r2 then print 1 else print 2
$$P_2 = *x = 1$$
 | r1 = *x; r2 = r1;
if r1=r2 then print 1 else print 2

Executions of P1: Executions of P2: W_{t_1} R_{t_2} It is correct to rewrite P1 into P2, but not the opposite! R_{t_2}

Behaviours of P1: $[P_{t_2} 1], [P_{t_2} 2]$ Behaviours of P2: $[P_{t_2} 1]$

There is only one execution with a printing behaviour:

$$\mathsf{W}_{t_1} \, x {=} 1, \mathsf{W}_{t_1} \, y {=} 1, \mathsf{R}_{t_2} \, x {=} 1, \mathsf{W}_{t_2} \, x {=} 2, \mathsf{W}_{t_2} \, y {=} 2, \mathsf{R}_{t_1} \, y {=} 2, \mathsf{R}_{t_1} \, x {=} 2, \mathsf{P}_{t_1} \, 2$$

But a compiler would optimise to:

The only execution with a printing behaviour in the optimised code is:

$$\mathsf{W}_{t_1} \, x \!\!=\!\! 1, \mathsf{W}_{t_1} \, y \!\!=\!\! 1, \mathsf{R}_{t_2} \, x \!\!=\!\! 1, \mathsf{W}_{t_2} \, x \!\!=\!\! 2, \mathsf{W}_{t_2} \, y \!\!=\!\! 2, \mathsf{R}_{t_1} \, y \!\!=\!\! 2, \mathsf{P}_{t_1} \, 1$$

So the optimisation is not correct.

Our first example highlighted that CSE is incorrect in SC.

Here is another example.

$$\begin{aligned} & [\mathsf{P}_{t_2} \, \mathsf{1}, \mathsf{P}_{t_2} \, \mathsf{0}, \mathsf{P}_{t_2} \, \mathsf{1}] \\ & [\mathsf{P}_{t_2} \, \mathsf{0}, \mathsf{P}_{t_2} \, \mathsf{1}, \mathsf{P}_{t_2} \, \mathsf{1}] \\ & [\mathsf{P}_{t_2} \, \mathsf{0}, \mathsf{P}_{t_2} \, \mathsf{0}, \mathsf{P}_{t_2} \, \mathsf{1}] \\ & [\mathsf{P}_{t_2} \, \mathsf{0}, \mathsf{P}_{t_2} \, \mathsf{0}, \mathsf{P}_{t_2} \, \mathsf{0}] \end{aligned}$$

The observable behaviours are (note that 0 - 1 - 0 is not observable):

$$\begin{aligned} & [\mathsf{P}_{t_2}\,1,\mathsf{P}_{t_2}\,1,\mathsf{P}_{t_2}\,1] \\ & [\mathsf{P}_{t_2}\,1,\mathsf{P}_{t_2}\,0,\mathsf{P}_{t_2}\,1] \\ & [\mathsf{P}_{t_2}\,0,\mathsf{P}_{t_2}\,1,\mathsf{P}_{t_2}\,1] \\ & [\mathsf{P}_{t_2}\,0,\mathsf{P}_{t_2}\,0,\mathsf{P}_{t_2}\,1] \\ & [\mathsf{P}_{t_2}\,0,\mathsf{P}_{t_2}\,0,\mathsf{P}_{t_2}\,0] \end{aligned}$$

But a compiler would optimise as:

Let's compare the behaviours of the two programs:

$$\begin{aligned} & [\mathsf{P}_{t_2}\,1,\mathsf{P}_{t_2}\,1,\mathsf{P}_{t_2}\,1] & [\mathsf{P}_{t_2}\,1,\mathsf{P}_{t_2}\,1,\mathsf{P}_{t_2}\,1] \\ & [\mathsf{P}_{t_2}\,1,\mathsf{P}_{t_2}\,0,\mathsf{P}_{t_2}\,1] & [\mathsf{P}_{t_2}\,1,\mathsf{P}_{t_2}\,0,\mathsf{P}_{t_2}\,1] \\ & [\mathsf{P}_{t_2}\,0,\mathsf{P}_{t_2}\,1,\mathsf{P}_{t_2}\,1] & [\mathsf{P}_{t_2}\,0,\mathsf{P}_{t_2}\,1,\mathsf{P}_{t_2}\,0] \\ & [\mathsf{P}_{t_2}\,0,\mathsf{P}_{t_2}\,0,\mathsf{P}_{t_2}\,1] & [\mathsf{P}_{t_2}\,0,\mathsf{P}_{t_2}\,0,\mathsf{P}_{t_2}\,0] \\ & [\mathsf{P}_{t_2}\,0,\mathsf{P}_{t_2}\,0,\mathsf{P}_{t_2}\,0] \end{aligned}$$

$$*x = 1; r = *x; *x = 1; r = *x;$$

The optimised program exhibits a new, unexpected, behaviour.

Let's compare the behaviours of the two programs.

$$[P_{t_2} 1, P_{t_2} 1, P_{t_2} 1] \\ [P_{t_2} 1, P_{t_2} 0, P_{t_2} 1] \\ [P_{t_2} 1, P_{t_2} 0, P_{t_2} 1] \\ [P_{t_2} 0, P_{t_2} 1, P_{t_2} 1] \\ [P_{t_2} 0, P_{t_2} 1, P_{t_2} 1] \\ [P_{t_2} 0, P_{t_2} 0, P_{t_2} 1] \\ [P_{t_2} 0, P_{t_2} 0, P_{t_2} 0] \\ [P_{t_2} 0, P_{t_2} 0, P_{t_2} 0]$$

Reordering incorrect

$$*x = 1;$$
 $*y = 1;$ $r1 = *y$ $*y = 1;$ $r1 = *y$ $r2 = *x;$ $r2 = *x;$ print r1 print r2 print r1 print r2

Again, the optimised program exhibits a new behaviour:

$$\begin{array}{c} [\mathsf{P}_{t_1} \, \mathsf{0}, \mathsf{P}_{t_2} \, \mathsf{1}] & \qquad \qquad [\mathsf{P}_{t_1} \, \mathsf{0}, \mathsf{P}_{t_2} \, \mathsf{1}] \\ [\mathsf{P}_{t_1} \, \mathsf{1}, \mathsf{P}_{t_2} \, \mathsf{0}] & \qquad \qquad [\mathsf{P}_{t_1} \, \mathsf{1}, \mathsf{P}_{t_2} \, \mathsf{0}] \\ [\mathsf{P}_{t_1} \, \mathsf{1}, \mathsf{P}_{t_2} \, \mathsf{1}] & \qquad \qquad [\mathsf{P}_{t_1} \, \mathsf{1}, \mathsf{P}_{t_2} \, \mathsf{1}] \\ [\mathsf{P}_{t_1} \, \mathsf{0}, \mathsf{P}_{t_2} \, \mathsf{0}] & \qquad \qquad [\mathsf{P}_{t_2} \, \mathsf{0}] \\ \end{array}$$

Elimination of adjacent accesses

There are some correct optimisations under SC. For example it is correct to rewrite:

$$r1 = *x; r2 = *x \rightarrow r1 = *x; r2 = r1$$

The basic idea: whenever we perform the read r1 = *x in the optimised program, we perfom both reads in the source program.

(More on this later)

Elimination of adjacent accesses

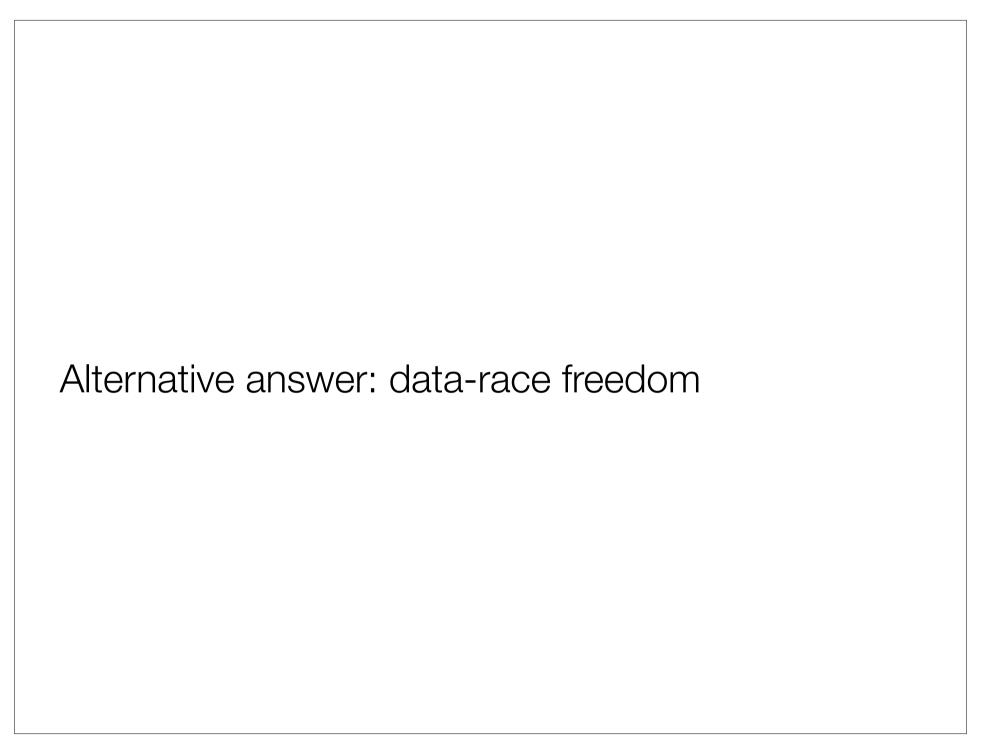
There are some correct optimisations under SC. For example it is correct to rewrite:

$$r1 = *x; r2 = *x \rightarrow r1 = *x; r2 = r1$$

Can we define a model that:

- 1) enables more optimisations than SC, and
- 2) retains the simplicity of SC?

(iviore on this later



Data-race freedom

Our examples again:

| Thread 0 | Thread 1 |
|----------|---------------|
| *y = 1 | if *x == 1 |
| *x = 1 | then print *y |

- the problematic transformations
 (e.g. swapping the two writes in thread 0) do not change the meaning of single-threaded programs;
- the problematic transformations are detectable only by code that allows two threads to access the same data simultaneously in conflicting ways (e.g. one thread writes the datas read by the other).

Data-race freedom

Thread 1 ...intuition... Our exam == 1 print *y Programming languages provide the prob viour: 0 synchronisation mechanisms (e.g. sw thread orograms; if these are used (and implemented) correctly, we might avoid the issues above... • the prol de that allows two threads to access the same data simultaneously in

conflicting ways (e.g. one thread writes the datas read by the other).

The basic solution

Prohibit data races

Observable behaviour: 0

Defined as follows:

- two memory operations **conflict** if they access the same memory location and at least one is a store operation;
- a SC execution (interleaving) contains a data race if two conflicting operations corresponding to different threads are adjacent (maybe executed concurrently).

Example: a data race in the example above:

$$W_{t_1} y=1, W_{t_1} x=1, R_{t_2} x=1, R_{t_2} y=1, P_{t_2} 1$$

The basic solution

Prohibit data races

Observable behaviour: 0

Defined as follows:

two men location

• a SC exe

operatio

The definition of data race quantifies only over the sequential consistent executions

mory

ıflicting It (maybe

executed concurrently).

Example: a data race in the example above:

$$W_{t_1} y=1, W_{t_1} x=1, R_{t_2} x=1, R_{t_2} y=1, P_{t_2} 1$$

How do we avoid data races? (focus on high-level languages)

Locks

No lock(I) can appear in the interleaving unless prior lock(I) and unlock(I) calls from other threads balance.

Atomic variables

Allow concurrent access "exempt" from data races. Called volatile in Java.

Example:

| Thread 0 | Thread 1 |
|--------------------|---------------|
| *y = 1 | lock(); |
| <pre>lock();</pre> | tmp = *x; |
| *x = 1 | unlock(); |
| unlock(); | if tmp = 1 |
| | then print *y |

How do we avoid data races? (focus on high-level languages)

| Thread 0 | Thread 1 |
|-----------|---------------|
| *y = 1 | lock(); |
| lock(); | tmp = *x; |
| *x = 1 | unlock(); |
| unlock(); | if tmp = 1 |
| | then print *y |

This program is data-race free:

```
*y = 1; lock(); *x = 1;unlock(); lock(); tmp = *x;unlock(); if tmp=1 then print *y

*y = 1; lock(); tmp = *x; unlock(); lock(); *x = 1; unlock(); if tmp=1

*y = 1; lock(); tmp = *x; unlock(); if tmp=1; lock(); *x = 1; unlock();

lock(); tmp = *x;unlock(); *y = 1; lock(); *x = 1; unlock(); if tmp=1

lock(); tmp = *x; unlock(); if tmp=1; *y = 1; lock(); *x = 1; unlock();

lock(); tmp = *x;unlock(); *y = 1; if tmp=1; lock(); *x = 1; unlock();
```

How do we avoid data races? (focus on high-level languages)

- lock(), unlock() are opaque for the compiler: viewed as potentially modifying any location, memory operations cannot be moved past them
- •lock(), unlock() contain "sufficient fences" to prevent hardware reordering across them and global orderering

```
*y = 1; lock();*x = 1;unlock(); lock();tmp = *x;unlock(); if tmp=1 then print *y

*y = 1; lock(); tmp = *x; unlock(); lock(); *x = 1; unlock(); if tmp=1

*y = 1; lock(); tmp = *x; unlock(); if tmp=1; lock(); *x = 1; unlock();

lock();tmp = *x;unlock(); *y = 1; lock(); *x = 1; unlock(); if tmp=1

lock(); tmp = *x; unlock(); if tmp=1; *y = 1; lock(); *x = 1; unlock();

lock();tmp = *x;unlock(); *y = 1; if tmp=1; lock(); *x = 1; unlock();
```

How do v

Compiler/hardware can continue to reorder accesses

Intuition:

compiler/hardware do not know about threads, but only racing threads can tell the difference!
 potentially m

moved past them

•lock(), unlock() contain "sufficient fences" to prevent hardware reordering across them and global orderering

```
*y = 1; lock();*x = 1;unlock(); lock();tmp = *x;unlock(); if tmp=1 then print *y

*y = 1; lock(); tmp = *x; unlock(); lock(); *x = 1; unlock(); if tmp=1

*y = 1; lock(); tmp = *x; unlock(); if tmp=1; lock(); *x = 1; unlock();

lock();tmp = *x;unlock(); *y = 1; lock(); *x = 1; unlock();

lock();tmp = *x; unlock(); if tmp=1; *y = 1; lock(); *x = 1; unlock();

lock();tmp = *x;unlock(); *y = 1; if tmp=1; lock(); *x = 1; unlock();
```

Another example of DRF program

Exercise: is this program DRF?

| Thread 0 | Thread 1 |
|-------------|-------------|
| if *x == 1 | if *y == 1 |
| then *y = 1 | then *x = 1 |

Another example of DRF program

Exercise: is this program DRF?

| Thread 0 | Thread 1 |
|-------------|-------------|
| if *x == 1 | if *y == 1 |
| then *y = 1 | then *x = 1 |

Answer: yes!

The writes cannot be executed in any SC execution, so they cannot participate in a data race.

Another example of DRF program

Exercise: is this program DRF?

| Thread 0 | Thread 1 |
|-------------|---------------|
| if *x == 1 | if *y == 1 |
| then *y = 1 | then $*x = 1$ |

Data-race freedom is not the ultimate panacea

The par

An

- the absence of data-races is hard to verify / test (undecidable)
- imagine debugging: my program ended with a wrong result, then either my program has a bug OR it has a data-race

Validity of compiler optimisations, summary

| Transformation | SC | DRF |
|--|---------------|--------------|
| Memory trace preserving transformations | ✓ | ✓ |
| Redundant read after read elimination | ✓ * | ✓ |
| Redundant read after write elimination | ✓ * | |
| Irrelevant read elimination | ✓ | |
| Redundant write before write elimination | ✓ * | ✓ |
| Redundant write after read elimination | ✓ * | |
| Irrelevant read introduction | ✓ | $ \times $ |
| Normal memory accesses reordering | × | |
| Roach-motel reordering | ×(√for locks) | |
| External action reordering | × | ✓ |

^{*} Optimisations legal only on adjacent statements.

Validity of compiler optimisations, summary

| Transformation | SC |
|---|----------|
| Memory trace preserving transformations | √ |



Jaroslav Sevcik

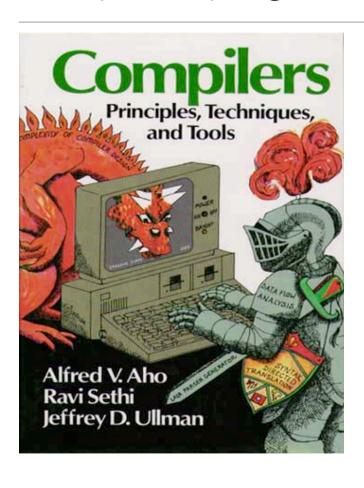
Safe Optimisations for Shared-Memory Concurrent Programs

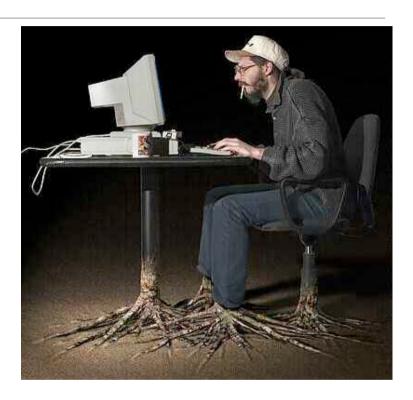
PLDI 2011

| Roach-motel reordering | ×(√for locks) | \checkmark |
|----------------------------|---------------|--------------|
| External action reordering | × | ✓ |

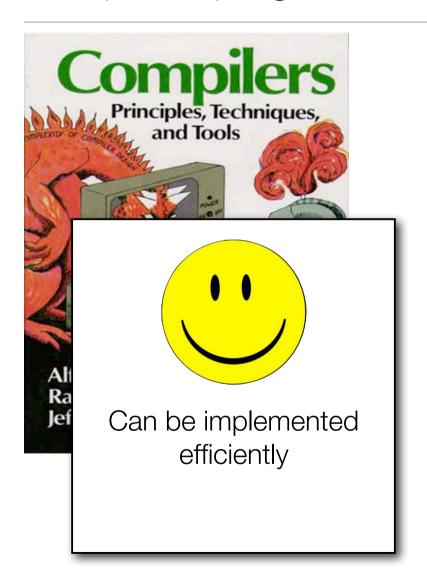
^{*} Optimisations legal only on adjacent statements.

Compilers, programmers & data-race freedom



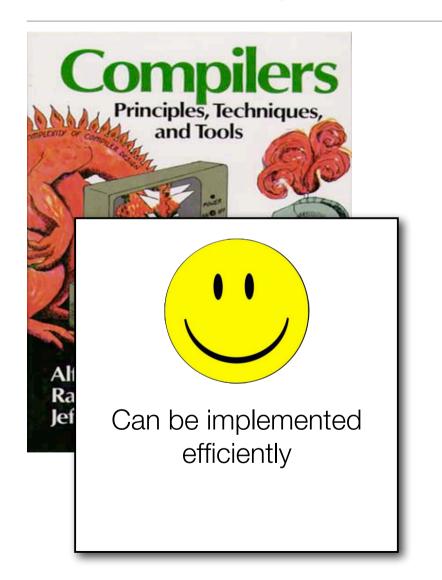


Compilers, programmers & data-race freedom

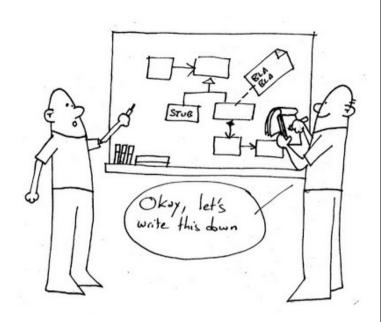




Compilers, programmers & data-race freedom







Data-race freedom, formalisation

A toy language: semantics

```
location, x shared memory location
register, r thread-local variable
integer, n integers
thread_id, t thread identifier
statement, s := statements
  r := x
x := r
                     read from memory
                     write to memory
  r := n
                     load constant into register
                     lock
  lock
                  unlock
  unlock
  print r
                    output
program, p ::= s;...;s a program is a sequence of statements
system ::= concurrent system
  | t_0:p_0 | ... | t_n:p_n parallel composition of n threads
```

A toy language: semantics

```
location, x
                    shared memory location
                    thread-local variable
register, r
in
th
   We work with a toy language, but this approach scales to the full
                 Java Memory Model or C11/C++11.
   lock
                         lock
                        unlock
   unlock
                        output
   print r
program, p ::= s;...;s a program is a sequence of statements
system ::= concurrent system
                       parallel composition of n threads
  | t_0: p_0 | \dots | t_n: p_n
```

Traces and tracesets

Definition [trace]: a sequence of memory operations (read, write, thread start, I/O, synchronisation). Thread start is always the first action of thread. All actions in a trace belong to the same thread.

Definition [traceset]: a traceset is a prefix-closed set of traces.

Sample traceset:

$$\{[S(0), R[x=v], W[y=v]] \mid v \in V\}$$

 $\cup \{[S(1), R[y=v], W[x=1], X(v)] \mid v \in V\}$

Remarks:

1. Reads can read arbitrary values from memory.

thre

sta

Del

2. Tracesets should not be confused with interleavings.

3. Tracesets do not enforce receptiveness or determinism:

$$\{[S(0)], [S(0), R[x=1]], [S(0), W[y=1]]\}$$

is also a valid traceset for the example below.

Sample traceset:

$$\{[S(0), R[x=v], W[y=v]] \mid v \in V\}$$

 $\cup \{[S(1), R[y=v], W[x=1], X(v)] \mid v \in V\}$

Associate tracesets to toy language programs

$$< S, r := x; s > \frac{R[x=v]}{\longrightarrow} < S[r=v], s >$$
 $< S, x := r; s > \frac{W[x=S(r)]}{\longrightarrow} < S, s >$
 $< S, r := n; s > \frac{T}{\longrightarrow} < S[r=n], s >$
 $< S, lock; s > \frac{L}{\longrightarrow} < S, s >$
 $< S, unlock; s > \frac{U}{\longrightarrow} < S, s >$
 $< S, print r; s > \frac{X(S(r))}{\longrightarrow} < S, s >$
 $< S, t_0:p_0 \mid ... \mid t_n:p_n > \frac{S(i)}{\longrightarrow} < S, p_i >$

Tracesets and interleavings

Definition [interleaving]: an interleaving is a sequence of thread-identifier-action pairs.

Example: y:=1; || r2:=v; print r2;

$$I' = [\langle 0, S(0) \rangle, \langle 1, S(1) \rangle, \langle 0, W[y=1] \rangle, \langle 1, R[v=0] \rangle, \langle 1, X(0) \rangle]$$

Given an interleaving *I*, the trace of *tid* in *I* is the sequence of actions of thread *tid* in *I*, e.g.:

trace
$$1 l' = [S(1), R[v=0], X(0)].$$

Conversely, given a traceset, we can compute all the well-formed interleavings (called *executions*)... (next slide)

Tracesets and interleavings

An interleaving *I* is an *execution* of a traceset *T* if:

- for all tid, trace $tid I \in T$ (traces belong to the traceset)
- tids correspond to entry points S(tid)
- lock / unlock alternates correctly
- each read sees the most recent write to the same location (read/from).

(The last property enforce the sequentially consistent semantics for memory accesses).

Tracesets and interleavings

An interleaving *I* is an *execution* of a traceset *T* if:

- for
- tid
- 100
- 1. Interleavings order totally the actions, and do not keep track of which actions happen in parallel.

(The

2. It is however possible to put more structure on interleavings, and recover informations about concurrency.

Remarks:

sses).

Happens-before

Definition [program order]: program order, $<_{po}$, is a total order over the actions of the same thread in an interleaving.

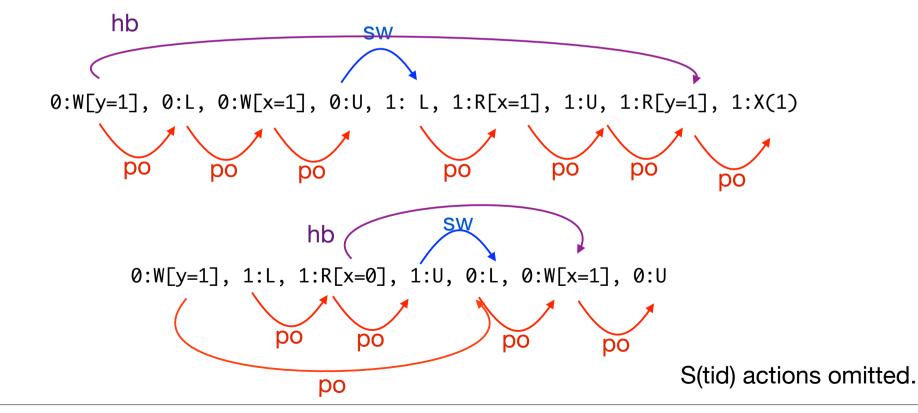
Definition [synchronises with]: in an interleaving I, index i synchroniseswith index j, $i <_{sw} j$, if i < j and $A(l_i) = U$ (unlock), $A(l_j) = L$ (lock).

Definition [happens-before]: Happens-before is the transitive closure of program order and synchronises with.

61

Examples of happens before

| Thread 0 | Thread 1 |
|-----------|---------------|
| *y = 1 | lock(); |
| lock(); | tmp = *x; |
| *x = 1 | unlock(); |
| unlock(); | if tmp = 1 |
| | then print *y |



Data-race freedom

Definition [data-race-freedom]: A traceset is data-race free if none of its executions has two adjacent conflicting actions from different threads.

Equivalently, a traceset is data-race free if in all its executions all pairs of conflicting actions are ordered by happens-before.

A racy program

| Thread 0 | Thread 1 |
|----------|---------------|
| *y = 1 | if *x == 1 |
| *x = 1 | then print *y |

Two conflicting accesses not related by happens before.

Data-race freedom: equivalence of definitions

Given an execution

$$\alpha ++ [a] ++ \beta ++ [b]$$

of a traceset T where [a] and [b] are the first conflicting actions not related by happen-before, we build the interleaving

$$\alpha ++ \beta' ++ [a] ++ [b]$$

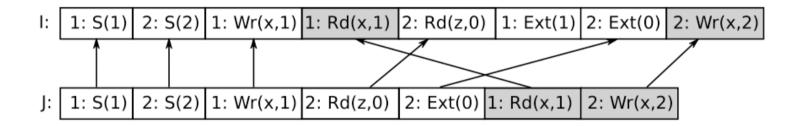
where β ' are all the actions from β that strictly happen-before [b].

It remains to show that $\alpha ++ \beta' ++ [a] ++ [b]$ is an execution of T.

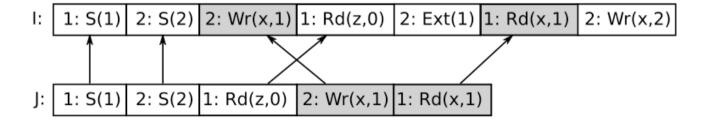
The formal proof is tedious and not easy (see Boyland 2008, Bohem & Adve 2008, Sevcik), here will give the intuitions of the construction on an example.

Data-race freedom: equivalence of definitions

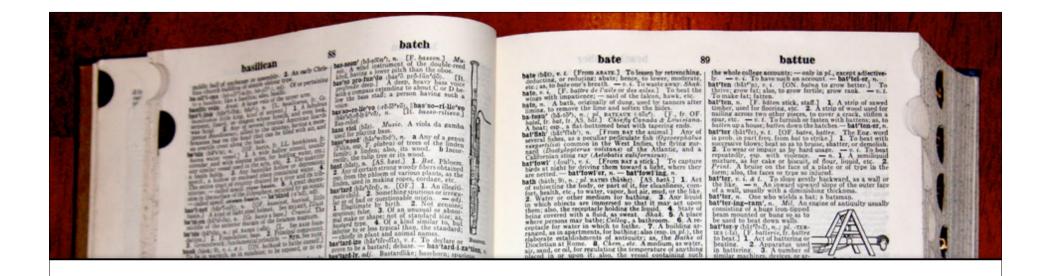
```
Thread 1: x := 1; r1 := x; print r1;
Thread 2: r2 := z; print r2; x := 2;
```



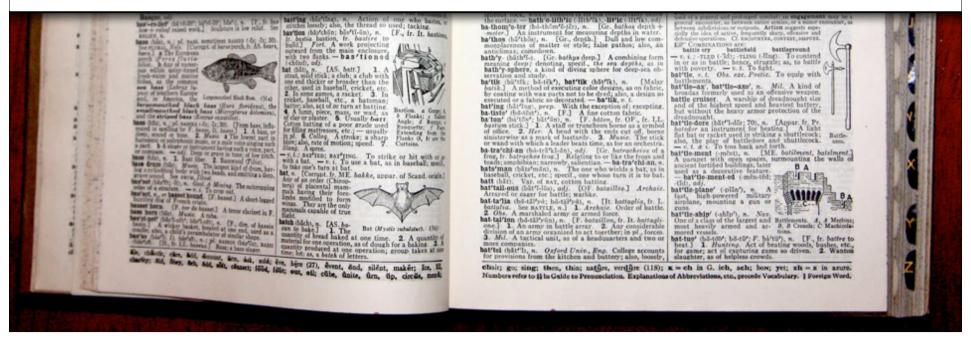
read first



write first



Defining programming language memory models



Don't.

No concurrency.

Implemented by highly-successful programming languages (**OCaml**)

Poor match for current trends

Don't. No shared memory

A good match for some problems (see Erlang, MPI, ...)

Don't.

But language ensures data-race freedom

Possible:

- syntactically ensuring data accesses protected by associated locks
- fancy effect type systems

Not suitable for general purpose programming.

Don't.

Leave it (sort of) up to the hardware

Example:

MLton, a high performance ML-to-x86 compiler with concurrency extensions

Accesses to ML refs exhibit the underlying x86-TSO behaviour (atomicity is guaranteed though)

Do.

Use data race freedom as a definition

- 1. Programs that race-free have only sequentially consistent behaviours
- 2. Programs that have a race in some execution can behave in any way

Sarita Adve & Mark Hill, 1990

Do.

Use data race freedom as a definition

Pro:

- simple
- strong guarantees for most code
- allows lots of freedom for compiler and hardware optimisations

Cons:

- undecidable premise
- can't write racy programs (escape mechanisms?)

Ada 83

[ANSI-STD-1815A-1983, 9.11] For the actions performed by a program that uses shared variables, the following assumptions can always be made:

- If between two synchronization points in a task, this task reads a shared variable whose type is a scalar or access type, then the variable is not updated by any other task at any time between these two points.
- If between two synchronization points in a task, this task updates a shared variable whose task type is a scalar or access type, then the variable is neither read nor updated by any other task at any time between these two points.

The execution of the program is erroneous if any of these assumptions is violated.

Data-races are errors

Posix Threads Specification

[IEEE 1003.1-2008, Base Definitions 4.11] Applications shall ensure that access to any memory location by more than one thread of control (threads or processes) is restricted such that no thread of control can read or modify a memory location while another thread of control may be modifying it.

Data-races are errors

C++ 2011 / C1x

[C++ 2011 FDIS (WG21/N3290) 1.10p21] 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.

Data-races are errors

Data race freedom as a definition

• Core of the C11/C++11 standard.

Hans Boehm & Sarita Adve, PLDI 2008.



• Part of the JSR-133 standard.

Jeremy Manson & Bill Pugh & Sarita Adve, PLDI 2008.



Data race freedom as a definition

Core of the C11/C++11 standard.

Hans Boehm & Sarita Adve, PLDI 2008.

with some escape mechanism called "low level atomics".

Mark Batty & al., POPL 2011.

Part of the JSR-133 standard.

Jeremy Manson & Bill Pugh & Sarita Adve, PLDI 2008.

DRF gives no guarantees for untrusted code: a disaster for Java, which relies on unforgeable pointers for its security guarantees.

JSR-133 is DRF + some out-of-thin-air guarantees for all code.

A word on JSR-133

Goal 1: data-race free programs are sequentially consistent;

Goal 2: all programs satisfy some memory safety requirements;

Goal 3: common compiler optimisations are sound.

Goal 2: all programs satisfy some memory safety requirements.

Programs should never read values that cannot be written by the program:

the only possible result should be printing two zeros because no other value appears in or can be created by the program.

Goal 2: all programs satisfy some memory safety requirements.

Programs should never read values that cannot be written by the program:

the only possible result should be printing two zeros because no other value appears in or can be created by the program.

Under DRF, it is correct to speculate on values of writes:

The transformed program can now print 42. This will be theoretically possible in C++11, but not in Java.

The program above looks benign, why does Java care so much about out-of-thin-air?

Out-of-thin-air is not so benign for references. Compare:

initially
$$x = y = 0$$

 $r1 := x$ $r2 := y$
 $y := r1$ $x := r2$ and $y := r1$ $x := r2$
print r1 print r2 initially $x = y = null$
 $r1 := x$ $r2 := y$
 $y := r1$ $x := r2$

What should r2.run() call?

If we allow out-of-thin-air, then it could do anything!



A word on JSR-133

Goal 1: data-race free programs are sequentially consistent;

Goal 2: all programs satisfy some memory safety requirements;

Goal 3: common compiler optimisations are sound.

The model is intricate, and fails to meet goal 3.

An example: should the source program print 1? can the optimised program print 1?

$$x = y = 0$$
HotSpot Optimization
$$x = y = 0$$

$$r1 = x$$

$$y = r1$$

$$r2 = y$$

$$x = (r2 = 1)?y : 1$$

$$print r2$$

$$x = y = 0$$

$$r1 = x$$

$$y = r1$$

$$r2 = y$$

$$print r2$$

Jaroslav Ševčík, David Aspinall, ECOOP 2008

A word on C11/C++11 low-level atomics

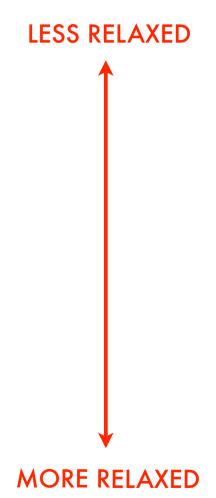
```
std::atomic<int> flag0(0),flag1(0),turn(0);
void lock(unsigned index) {
   if (0 == index) {
                                                     Atomic variable declaration
       flag0.store(1, std::memory_order_relaxed);
       turn.exchange(1, std::memory_order_aca_rel);
       while (flag1.load(std::memory_order_acquire)
           && 1 == turn.load(std::memory_order_relaxed))
           std::this_thread::yield();
   } else {
       flag1.store(1, std::memory_order_relaxed);
                                                                 New syntax for
       turn.exchange(0, std::memory_order_acq_rel);
                                                                 memory accesses
       while (flag0.load(std::memory_order_acquire)
           && 0 == turn.load(std::memory_order_relaxed))
           std::this_thread::yield();
void unlock(unsigned index) {
                                                                Qualifier
   if (0 == index) {
       flag0.store(0, std::memory_order_release);
   } else {
       flag1.store(0, std::memory_order_release);
```

MO_SEQ_CST

MO_RELEASE / MO_ACQUIRE

MO_RELEASE / MO_CONSUME

MO RELAXED



LESS RELAXED

Sequential consistent accesses

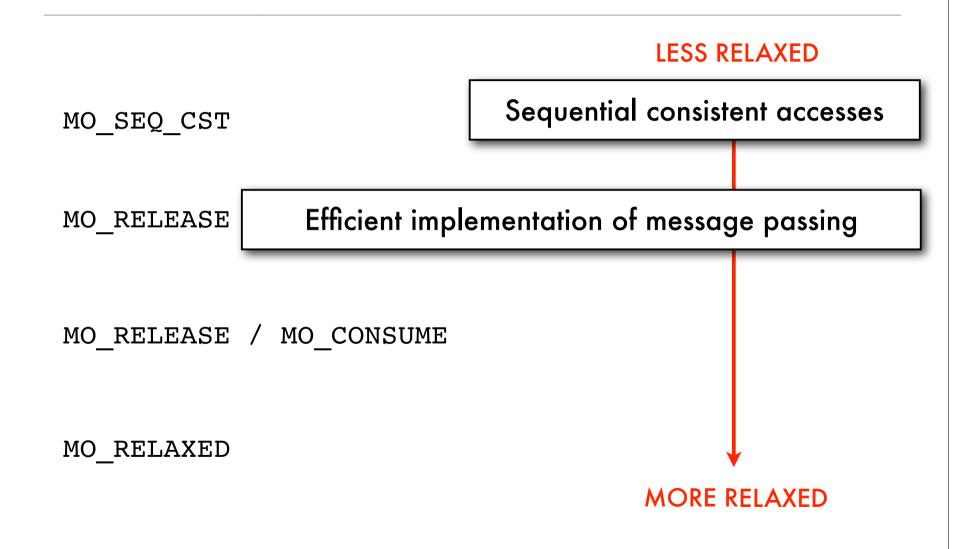
MO_SEQ_CST

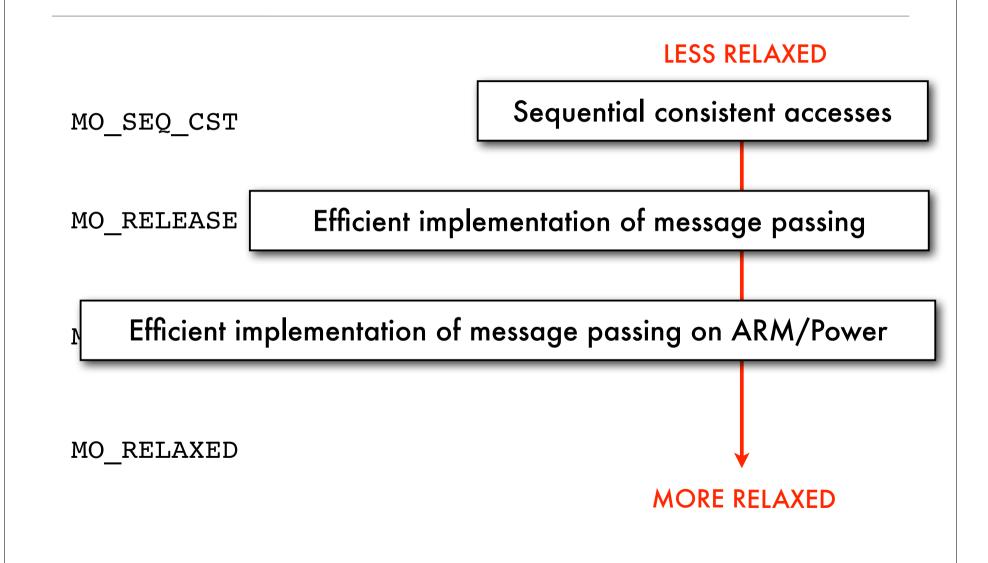
MO_RELEASE / MO_ACQUIRE

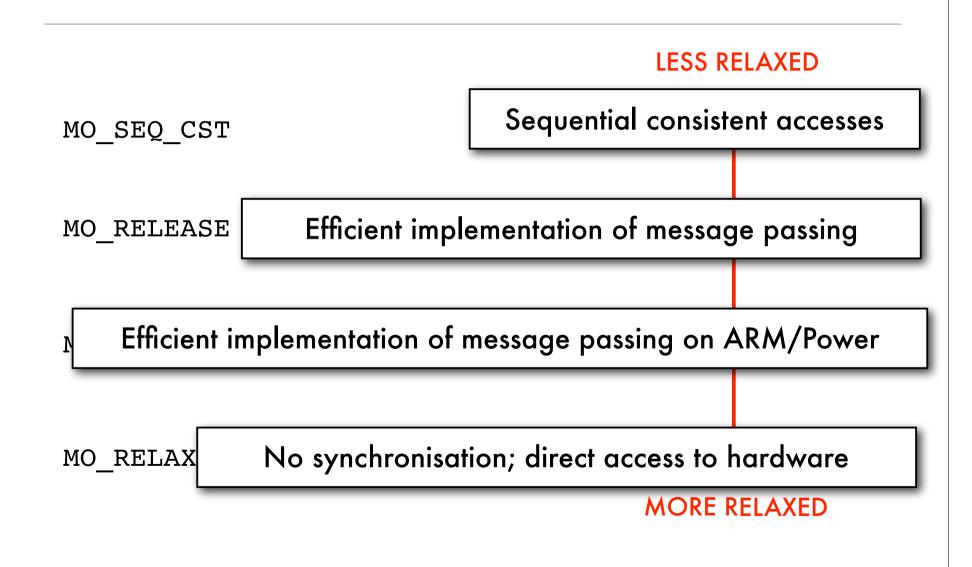
MO_RELEASE / MO_CONSUME

MO RELAXED

MORE RELAXED







Memory access synchronisation

$$x = y = 0$$

Thread 1 Thread 2

Memory access synchronisation

$$x = y = 0$$

Thread 1 Thread 2

$$\stackrel{happens-before}{\longrightarrow} = \\ \left(\begin{array}{c} \underline{\text{sequenced-before}} \\ \end{array} \right)^+$$

Non-atomic loads must return the *most recent write* in the happens-before order (unique in a DRF program)

Understanding MO_RELAXED

$$x = y = 0$$

Thread 1 Thread 2

Understanding MO_RELAXED

$$x = y = 0$$

Thread 1 Thread 2

DATA RACE

Two conflicting accesses not related by happens-before

Understanding MO_RELAXED

$$x = y = 0$$

Thread 1 Thread 2

WELL DEFINED

but r2 = 0 is possible

Intuition

the compiler (or hardware) can reorder independent accesses

$$x = y = 0$$

Thread 1 Thread 2

WELL DEFINED

but r2 = 0 is possible

Intuition

the compiler (or hardware) can reorder independent accesses

$$x = y = 0$$

Thread 1 Thread 2

Allow a RELAXED load to see any store that:

- does not happen-after it
- is not hidden by an intervening store hb-ordered between them

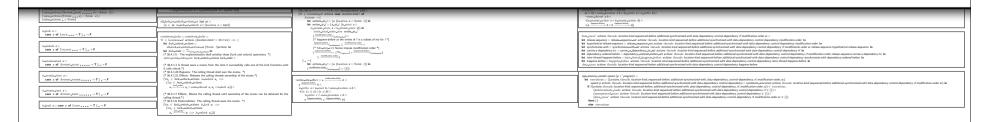
The full model

| $a \xrightarrow{r} b = (a, b) \in r$ | is store a = case a of Store \rightarrow T \rightarrow F | | |
|--|---|--|--|
| | is_fence a = case a of Freez→T →F | to_diment_m_based > :: comm_thend > m_based \lambda in_chand \lambda in_ch | sinhy, disk_effect_are actions through location bird segmented below additional questionated with data dependency control dependency targets below in (1, 4) = 2 in 1. (in & approximation set (1, 4) = 2 in 1. (in this papers before set (1, 4) = 2 in 1. (in this papers before a trining braidward set questionated with data-dependency targets before a (3). |
| $a \ r \ b = (a,b) \in r$ | ic_lock_or_unlock_a = ic_lock_a \times ic_unlock_a | | |
| $a \not\stackrel{c}{\to} b = (a,b) \not\in \ r$ | icatomicaction a :: | The state of the | $\begin{array}{ll} \operatorname{trible}_{\mathrm{constant}}(x,y) & \operatorname{constant}(x) & \operatorname{constant}(x) & \operatorname{constant}(y) & \operatorname{constant}(y) \\ & \operatorname{constant}(x) & \operatorname{constant}(y) &$ |
| | licatomicaction a ≡ licatomicacid a V licatomicatore a V licatomicacore a | | $\langle 0 \rangle = -c C \rangle$ $\langle 1 $ |
| $a \stackrel{\checkmark}{\rightarrow} b \stackrel{\Rightarrow}{\rightarrow} c = a \stackrel{\checkmark}{\rightarrow} b \wedge b \stackrel{\Rightarrow}{\rightarrow} c$ | Iscland_on_store a = iscland a V is_store a | (in-dimension $A_{ad} \otimes A_{ad} = b \otimes A_{ad}$ | |
| relation, over x $nt = domain \ nt \subseteq x \wedge range \ nt \subseteq x$ | la_read a = la_read a v is_atomic_rear a v is_lead a | release_sequence_set actions through, location-bind sequenced-before additional-synchronized-with data-dependency control-dependency modification-order :: | suppose $f x = [r, 2a \in a, (p = f x)]$ |
| | | release_sequence actions threads location-kind sequenced-before additional-synchronized-with data-dependency control-dependency modification-order a b) | Table proposes of all princes and table proposes of all princes and table and table as a second and table and table are a second as a |
| $\frac{nd}{-r} _{k}=nd \ \cap \ (\epsilon \ \times \ \epsilon)$ | k_write a = k_atomic_store a∨k_atomic_row a∨k_store a | Sportfolia - whom - regions = y and individual measures = 2 : Section (Control 2011 to 10 (Control 2011 to | (QR Lost interest Section 2 them (section 2) |
| $nd_a = nd \cap (a \times a)$ | le acceler a - | $(b=a)$ \vee | (1) |
| $\xrightarrow{nl} _{a} = nal \cap (a \times a)$ | (case memory_order a of Some mem_ord → | (to_distance $\Delta b \land \lambda = b \land b \land b$) (Ye. $\lambda = \frac{\text{millivation order}}{1 + (b \land b)} \land b \Rightarrow b$ | |
| $ref _{a}=ref\ \cap\ (a\ \times\ a)$ | (Mo_scorne, Mo_sco_ne, Mo_neq_cer) ∧ (s_read a v in_fence a)) ∨ | | while, sequence of side offices, art scient through location-list apparent before additional-spectroscope with data-dependency control-dependency multifaction-order suppose before while-side-side-side-side-side-side-side-sid |
| strict_preceder and = invelocive and \(\) trass and | Summarization of | hypothetical inflams sequences are actions thereads location-kind sequenced-before additional synchronized-with data-dependency control-dependency modification-order :: hnoathetical inflams sequence actions through location-kind sequenced-before additional-innothronized-with data-dependency control-dependency modification-order a bit | condition areado from susping = consistent areado from susping = |
| | | and confined to | continuous participating a material participating a material participating and the continuous part |
| total_over s and = relation_over s and A | le_romenne a = le_road a > (memory_order a = Some Mo_consense) | eypathenalom_with = 2 Authoritometh; b = ("-address) from these derivative etc ") | eten ~(la. x ≤ 2))) ∧ |
| $(\forall x \in x. \forall y \in x. x \xrightarrow{ad} y \lor y \xrightarrow{ad} x \lor (x = y))$ | ligations a :: | | (in Fourier 3 is two distinct destinates $\beta = 0$) (if [10], only $\alpha = 0$) the fourier $\beta = 0$ is the fourier $\beta = 0$ in the fourier $\beta $ |
| strict_total_order_over s and :: strict_preacher and \(\chi_total_over s and\) | Some menged → menged ∈ {Mo_numers, Mo_sco_num, Mo_sun_cum} ∧ | (mm.location a $bh.b. \le states h.b. \le states h.b. \le states h.f. (" - mate quertenation -") (Guarille a h.locate bh.b. = -b) \lor$ | $(\delta = \delta) \wedge (\delta = \epsilon \text{ was } \epsilon \stackrel{r}{\sim} \delta)$ $\operatorname{star} = (\Delta s \stackrel{r}{\sim} \delta - \delta) \wedge \delta$ |
| and | Buydene 2: See inservoyander 2 of See inservoyander 2 of See inservoyander 2 of see god of (Mourenaux) Mourenaux, Mourenaux) ((u_vrite 2) U_dinne 2) [Notes + Localance 2) | (* - release) izquin quedresitation - *) (ita-linear = 0.4 inpequine 0.4 - resumplimed x 0.4 (it. x = 2 intermediate (x = 0)) (*) | $(r(x,y) \in \mathcal{L}_y)$ |
| $\begin{array}{l} x \stackrel{\rm red}{\longrightarrow} _{pool} y = \\ pood \ x \wedge x \stackrel{\rm red}{\longrightarrow} y \wedge \neg (\exists x. \ pood \ x \wedge x \stackrel{\rm red}{\longrightarrow} x \stackrel{\rm red}{\longrightarrow} y) \end{array}$ | $ u_a e u_{a} _{C}$ at $z = (memory_a erder a = Some Mo_a e u_{a} _{C}$ corr | | $\forall (y,b) \in \mathbb{R}$, a summarise $b \land b$ |
| $x \xrightarrow{aab} y =$ $x \xrightarrow{aab} y \land \neg(\exists x. x \xrightarrow{aab} x \xrightarrow{aab} y)$ | location_kind = | (**) - The control of | (Scape 4 (1) (1) (1) (1) (1) (1) (1) (1) (1) (1) |
| | Scotion Aind = Mutter Nos_wrome Nos_wrome Arome | Restriction of the American Am | $ \begin{array}{c} (r(x,y) \in \frac{1}{N} \text{distribution}, \\ r(x,y) \in \frac{1}{N} \text{distribution}, \end{array} $ |
| well, decorded $r=ud$ r | | (is_fance a \(\lambda \) i_release a \(\lambda \) | To de his completion de la logica de la completion de la logica de la completion de la comp |
| type_abbov action_id : string | actions_respect_location_kinds = actions_respect_location_kinds = \forall x. | (in James at Angelonia A.) (in semication in | $e^{i\phi} = i\phi = i\phi = i\phi$ $e^{i\phi} = i\phi$ $e^{i\phi} = i\phi$ $e^{i\phi} = i\phi$ |
| typo_abloov thread_id : string | Va. case location a of Stone I (case location and II (case location local III (case location local II (case locat | a general dates $X \cap X $ | 2 days |
| type_abbov location: string | Non_xrouse → is_load_or_store a Arouse → is_load_or_store a ∨ is_atomic_action a) | (%_atomic_action a \(\lambda\) includes a \(\lambda\) | The state of the s |
| | | Conjuminary of the Association | $(r(x,b) \in \stackrel{d}{\longrightarrow} x \text{-basissisce} rans b$ $\Rightarrow x \text{-millionisce} b) \wedge$ |
| type_abboov val : string | in_at_blocation_kind = in_at_blocation_kind = case location a of Source -b_blocation-kind = 80 Nounce -b | $\begin{array}{cccccccccccccccccccccccccccccccccccc$ | |
| memory_order_enum :: M0.300_00T | Some $I \rightarrow (location-kind I = Rd)$ $ \text{None:} \rightarrow P$ | synchronizes_with_set actions through location-kind sequenced before additional-synchronized-with data-dependency correct dependency of modification-order ac release-sequence hypothetical-release-sequence :: | $(0, k) \in \mathbb{Z}$, homosot k $\cong ((a_1 a_2 a_3 a_4 a_4 a_5) (k \times E_{2a_1} a_{2a_2} a_{2a_3} $ |
| тенного дейна деят :: 304,0,492,077 306,0,471,0,3320 306,0,471,0,3320 306,0,471,0,471,0,471 306,0,471,0,471,0,471 306,0,471,0,471,0,471 306,0,471,0,471,0,471 306,0,471,0,471,0,471 306,0,471 306,0, | le_at_marter_location a :: le_at_location_lind a Merrox | synchronisms, with actions should location-kind sequences before additional-synchronised with data-dependency control-dependency of modification-order or release-sequence hypothetical-release-sequence a b) | (* -Fence restrictions *) |
| Mo_econsus Mo_econsus Mo_econsus | iout_location_kind a Merrex | Gardine ϕ_a dependency $\phi_a = a$, written dependency $\phi_b = a(d^2 - a)$ and the adjustment $\phi_b = a(d^2 - a)$ and the adjustment $\phi_b = a$ | (2.3.5) $\frac{1}{2}$ (0.3.5) $$ |
| | | $3\left(\left(\frac{d}{d}-\right)-\frac{againstable(e)}{2}\right)\cup\frac{deta-dependency}{2}\right)^{0}b$ | (beginner at Newgogat at A Newfatting action B A) (beginner at Newgogat at B A) |
| action = Lock of action id thread id location Unabout of action id thread id location | is at atomic focation a — | carries_a_dependency_to_set actions through location-kind sequenced before additional-synchronized-with data-dependency control-dependency rf = | $a \stackrel{\leftarrow}{=} x \wedge y \stackrel{\leftarrow}{=} 0$ $a \stackrel{\leftarrow}{=} y \circ y \circ a \stackrel{\leftarrow}{=} a \circ y \circ y \circ a \circ y \circ y \circ y \circ y \circ y \circ y \circ$ |
| Aronne: a care of action id thread id memory order enum location val Aronne: errone of action id thread id memory order enum location val | ls_at_atomic_location_a = b_at_location_bind_a Arrownc | cartine a dependency to actions through location-hind sequenced before additional-synchronized with data-dependency control-dependency of a b) | (* 23.44*) (* 10.4.10*) (* 10.4.10*) |
| Lices de sectory testing l'activation. Account activat de séctory testing l'activation. Account activat de séctory threadil memory-artier anun location val. Account activat de séctory threadil memory-artier anun location val. Account activat de séctory threadil memory-artier anun location val sel sectory activation de la conference de sectory activation de la conference de | same thread $a b = (thread id of a = thread id of b)$ | Superior system of their $x \in \mathcal{A}$ and $x \in \mathcal{A}$ | [7:23.4*] (Co.1a):::::::::::::::::::::::::::::::::::: |
| | threadwise_molation_precr s ref :: | (ib. $ e_{constraint} \ge h \land (constraint) \ge h \land (ib. 2 = abox sequence h \land (ib. 2 = abox sequence + a = abox) \land (ib. 3 = abox sequence + abox) \land (ib. 4 = abox sequence + ab$ | $x \xrightarrow{\gamma} h \land h_{\gamma} \text{ sometime of the } h$ $= (y = y) \land y \xrightarrow{\gamma} \text{ sometime } y \land h$ |
| (action_id_ef (Lorer aid =) = aid) ∧ (action_id_ef (Untorer aid =) = aid) ∧ | threadwise_melation_over e and $=$ relation_over e and $(\gamma(a,b) \in rel.$ some_thread a $b)$ | $(b \xrightarrow{a-rine a Augustano ya b} d \lor (b = d)))$ | (* 23.5.*) |
| (action-short (LOCK at $a = 1 = adt$) \wedge (action-short (Morrice at $a = 1 = adt$) $= adt$) \sim (action-short (Morrice at $a = 1 = adt$) \sim (action-short (Morrice at $a = adt$) \sim | same_location $ab = (\text{location } a = \text{location } b)$ | dependency and red before set actions through location-kind sequenced before additional-synchronized-with data-dependency control-dependency of modification-order release-sequence carrier-a-dependency-to :: | Buckette Anderson An Anderson An American An American An American Anderson |
| (action_id_of (Loans aid) = aid) ∧ (action_id_of (Storm aid) = aid) ∧ | locations of actions = $\{l.$ lin. (location $s = Scene. I)\}$ | dependency undered Justices actions themsels location-kind sequenced-before additional-synchronized with data-dependency control-dependency of modification-order release-sequence carriers-dependency-to a b) | (*23.55) (*(*).4 |
| | well_formed_action a = | single_lappeas_before = distributionships; = (minoralization_lill_particles_minoralization_lill_ | |
| (thread, id, of (Lover, iid $_{A}$) = iid) $_{A}$ (thread, id, of (thread, iid $_{A}$) = iid) $_{A}$ (thread, id, of (thread, iid) $_{A}$ = iid) $_{A}$ (thread, id, of (Arman, iid) $_{A}$ | will, journel, joetten z = extrame_factor_memor_jet, — mem_jet { | | $S^{1}_{i}(A)S_{i}(A)gand(a)y := \int_{A}^{A}dx_{i}Agand(a)y := \int_{A}^{A}dx_{i}Agand(a)y$ |
| (thread_id_of (Aromec_eroes _ sid) = sid) ∧ (thread_id_of (Aromec_enew _ sid) = sid) ∧ | Atomic store - memord - → memord ∈ {Morelaked, Morelake, Morelacet} | consistent_uiraqui_uhappenu_uhabove akb :: irreflective (^{cab}) | condense control dependancy = consistent custon Lependancy = tradition () and depending conference (see) |
| (thread.id.ef (Lone. nid -) = nid) ∧ (thread.id.ef (Strone. nid -) = nid) ∧ (thread.id.ef (Strone. nid) = nid) | ATOMIC, LINE MODIFIE AND | liter_alcos(_looppous_alcos = inivitual_loopmo.lutes = | |
| (comments (from our end) | | | considered personal rections in the seal for the seal of the seal |
| Sour memory) (Arounce memory) = (memory order (Arounce memory memory) = | is an action of actions) A | $ \frac{\text{optimized and only}}{\left\{ \begin{array}{c} \text{optimized and only} \\ \text{optimized and only} \end{array} \right\} \ln \\ \binom{n}{n} \cup \left\{ \begin{array}{c} \text{optimized and only} \\ \text{optimized and only} \end{array} \right\} - \frac{1}{n} \end{aligned} $ | |
| Sour men_ord) A (memory-order (Aroune_nowmen_ord) = | | (→∪ (<u>superiora anne</u> o →))* | constant lacks actions thanks Excellent-land restrictions for the constant lacks actions thanks Excellent-land restrictions and the constant lacks actions thanks Excellent-land restrictions and the constant lands in the |
| (memory_order (Arome_mww _ memord) = Score man ord > | (in. recommendation and a first threadering relation anguenced before A threadering relation, were actions anguenced before A threadering relation, were actions control dependency A threadering relation, were actions control dependency A. | | constructed size action (Linuxia). Incolors that diagnosed where Additional experimentary and and produces year of all produces year. If all replaces year of a produce year of |
| (memory order (Aronne, more, memord) = SOSIE memory order (FENCE: _memord) = SOSIE memory order (FENCE: _memord) = | (i.e. with an anti-million and a filter of the control of the cont | [inco(minimum_s, in]); makening inter_bloom_begon_before = makenin_bloom_before = makenin_bloom_before = makenin_bloom_before = makenin_bloom_before = makenin_bloom_before = makenin_bloom_bloom_before = makenin_bloom_bloo | construction between the data person of which appeared where Allithous operations are that the appeared person person of the person person of the person person of the person person of the person per |
| manner come dell') ((manner) come (de vivour come come del) = (manner) come (de vivour come come del) = (manner) come (Presen, manner) = Sozia: manner) ((manner) come) (manner) come Sozia: Sozia: | distribution platfall, acres attition sequenced defent in distribution, acres attition distribution of themselves, includins, acres attition distribution, acres attition acres of dependency in distribution, acres attition control dependency in acres agreement, acres dependency in acres agreement acres del agreement produced with in (Ca. tambel-date) acres (Ca. tambel-date). | constitute jibera jibera jibera ze constitute jibera jibera jibera jibera ze | construction active. The control of agreement with an adjustment of the control of agreement of the control of agreement of the control of agreement |
| $\begin{cases} \text{descent}_{i} & \text{clin}(\mathcal{L}(x)) = \{\text{clin}(x), \text{clin}(x), \text{clin}(x), \text{clin}(x)\} \\ \text{clin}(x) & \text{clin}(x), \text{clin}(x)$ | with Emmediation or will amount faith to liquid antiquality (immedia) in siliquid antiquality (immedia) in faith to the control of the control of the control of faith to the control of the control of the control of faith to the control of the control of the control of control of the control of the control of the control of control of the control of the control of the control of control of the control of the control of the control of (in the control of the control of the control of (in the control of the control of the control of decided production (in the control of the control of decided production (in the control of dec | | constructed by a street Union. Notice to an dependent with a street of process of proces |
| $\begin{aligned} & \max_{x \in \mathcal{X}} \sup_{x \in \mathcal{X}} x \\ & (\operatorname{none}_{\mathcal{X}} \sup_{x \in \mathcal{X}} \sup_{x \in \mathcal{X}} x) = (\operatorname{none}_{\mathcal{X}} \sup_{x \in \mathcal{X}} x) \\ & (\operatorname{none}_{\mathcal{X}} \sup_{x \in \mathcal{X}} x) = (\operatorname{none}_{\mathcal{X}} \sup_{x \in \mathcal{X}} x) \\ & (\operatorname{none}_{\mathcal{X}} \sup_{x \in \mathcal{X}} x) = (\operatorname{none}_{\mathcal{X}} x) \\ & (\operatorname{none}_{\mathcal{X}} (\operatorname{None}_{\mathcal{X}} x)) = (\operatorname{none}_{\mathcal{X}} x) \\ & (\operatorname{none}_{\mathcal{X}} (\operatorname{None}_{\mathcal{X}} x)) = (\operatorname{none}_{\mathcal{X}} x) \\ & (\operatorname{none}_{\mathcal{X}} (\operatorname{None}_{\mathcal{X}} x)) = (\operatorname{none}_{\mathcal{X}} x) \\ & (\operatorname{none}_{\mathcal{X}} (\operatorname{None}_{\mathcal{X}} x)) = (\operatorname{none}_{\mathcal{X}} x) \\ & (\operatorname{none}_{\mathcal{X}} (\operatorname{None}_{\mathcal{X}} x)) = (\operatorname{none}_{\mathcal{X}} x) \\ & (\operatorname{none}_{\mathcal{X}} (\operatorname{None}_{\mathcal{X}} x)) = (\operatorname{none}_{\mathcal{X}} x) \\ & (\operatorname{none}_{\mathcal{X}} (\operatorname{None}_{\mathcal{X}} x)) = (\operatorname{none}_{\mathcal{X}} x) \\ & (\operatorname{none}_{\mathcal{X}} (\operatorname{None}_{\mathcal{X}} x)) = (\operatorname{none}_{\mathcal{X}} x) \\ & (\operatorname{none}_{\mathcal{X}} (\operatorname{None}_{\mathcal{X}} x)) = (\operatorname{none}_{\mathcal{X}} x) \\ & (\operatorname{none}_{\mathcal{X}} (\operatorname{none}_{\mathcal{X}} x)) = (\operatorname{none}_{\mathcal{X}} x) \\ & (\operatorname{none}_{\mathcal{X}} (\operatorname{none}_{\mathcal{X}} x)) = (\operatorname{none}_{\mathcal{X}} x) \\ & (\operatorname{none}_{\mathcal{X}} (\operatorname{none}_{\mathcal{X}} x)) = (\operatorname{none}_{\mathcal{X}} (\operatorname{none}_{\mathcal{X}} x)) \\ & (\operatorname{none}_{\mathcal{X}} (\operatorname{none}_{\mathcal{X}} x)) = (\operatorname{none}_{\mathcal{X}} (\operatorname{none}_{\mathcal{X}} x)) \\ & (\operatorname{none}_{\mathcal{X}} (\operatorname{none}_{\mathcal{X}} x)) = (\operatorname{none}_{\mathcal{X}} (\operatorname{none}_{\mathcal{X}} x)) \\ & (\operatorname{none}_{\mathcal{X}} (\operatorname{none}_{\mathcal{X}} x)) = (\operatorname{none}_{\mathcal{X}} (\operatorname{none}_{\mathcal{X}} x)) \\ & (\operatorname{none}_{\mathcal{X}} (\operatorname{none}_{\mathcal{X}} x)) = (\operatorname{none}_{\mathcal{X}} (\operatorname{none}_{\mathcal{X}} x)) \\ & (\operatorname{none}_{\mathcal{X}} (\operatorname{none}_{\mathcal{X}} x)) = (\operatorname{none}_{\mathcal{X}} (\operatorname{none}_{\mathcal{X}} x)) $ | well formed muck from manning - well formed made from manning - | entaining along James Ja | An interest tracks for the first track for the first tracks for the first tracks and the second seco |
| | well formed muck from manning - well formed made from manning - | entations_ates_planes_lapines_lapines_content_ates_planes_lapi | consistent annual_interaccepting, actions through locations intelligences define additional spectroscope with data dependency control dependency of a modification order happens before additional spectroscope with data dependency of an modification order happens before which eight effect visible supercess of side effects) |
| $\begin{aligned} & \text{down sum and } Q_1 \\ & \text{down sum and } Q_2 \\ & \text{down sum and } Q_3 \\ & down sum and $ | well formed muck from manning - well formed made from manning - | rendering sizes from deproposal polare = constant from deproposal polare = medicine (= medicine | construction for the Conference of the Conferenc |
| Boardin (Lett., s_1^2) = Steam β). (boardin (Sharata, γ_1) = Steam β). (boardin (Arroman, s_1) = Steam β). (boardin (Arroman, s_2) = Steam β). (boardin (Arroman, s_3) = Steam β). (boardin (Steam, s_3) = Steam β). | well formed muck from manning - well formed made from manning - | The state of the s | restrict, explaint, planting a state of the control to the control of agreement before a state of agreement of the control of agreement of agreement of the control of agreement of the control of agreement of the control of agreement of agreement of agreement of agreement of agreeme |
| Boardin (Lett., s_1^2) = Steam β). (boardin (Sharata, γ_1) = Steam β). (boardin (Arroman, s_1) = Steam β). (boardin (Arroman, s_2) = Steam β). (boardin (Arroman, s_3) = Steam β). (boardin (Steam, s_3) = Steam β). | with front a problem a proper $g = mL_1$ around g -mod g -m | entations_ates_planes_lapines_lapines_content_ates_planes_lapi | restrict, and planting a state of the state |
| $\begin{aligned} & \operatorname{posite}_{A}\left(\operatorname{Res}_{A}, \phi\right) \leq \operatorname{Som} \beta \right), \\ & \operatorname{posite}_{A}\left(\operatorname{Neure}, \phi\right) \geq \operatorname{Som} \beta \right), \\ & \operatorname{locate}_{A}\left(\operatorname{Neure}, \phi\right) \geq \operatorname{Som} \beta \right), \\ & \operatorname{locate}_{A}\left(\operatorname{Atom}_{A}, \phi\right) = \sum_{i} \operatorname{Som} \beta \right), \\ & \operatorname{locate}_{A}\left(\operatorname{Atom}_{A}, \phi\right) = \sum_{i} \operatorname{Som} \beta \right), \\ & \operatorname{locate}_{A}\left(\operatorname{Neure}, \phi\right) \geq \operatorname{Nous} \beta \right), \\ & \operatorname{locate}_{A}\left(\operatorname{Neure}, \phi\right) \geq \operatorname{Nous} \gamma \right), \\ & \operatorname{locate}_{A}\left(\operatorname{Neure}, \phi\right) \geq \operatorname{Nous}_{A}\left(\operatorname{Neure}, \phi\right), \\ & \operatorname{locate}_{A}\left(\operatorname{Neure}, \phi\right) \geq \operatorname{Neure}_{A}\left(\operatorname{Neure}, \phi\right), \\ & \operatorname{locate}_{A}\left(\operatorname{Neure}, \phi\right) \geq \operatorname{locate}_{A}\left(\operatorname{Neure}, \phi\right), \\ & \operatorname{locate}_{A}\left(\operatorname{Neure}, \phi\right) \geq \operatorname{locatee}_{A}\left(\operatorname{Neure}, \phi\right), \\ & \operatorname{locatee}_{A}\left(\operatorname{Neure}, \phi\right), \\ & \operatorname{locatee}_{A}\left(\operatorname{Neure}, \phi\right) \geq \operatorname{locatee}_{A}\left(\operatorname{Neure}, \phi\right), \\ & \operatorname{locatee}_{A}\left(\operatorname{Neure}, \phi\right), \\ & \operatorname{locatee}_{A}\left(Neur$ | with front a problem a proper $g = mL_1$ around g -mod g -m | The state of the s | continue parally lamparings, active transit bottom for depended and additional questionated with data depending commit dependently of a multifaction under temporal additional questionated with data depending commit dependency of a multifaction under temporal additional questionated and data dependency of a multifaction under temporal additional questionated in the continue of the |
| $\begin{aligned} & \operatorname{posite}_{A}\left(\operatorname{Res}_{A}, \phi\right) \leq \operatorname{Som} \beta \right), \\ & \operatorname{posite}_{A}\left(\operatorname{Neure}, \phi\right) \geq \operatorname{Som} \beta \right), \\ & \operatorname{locate}_{A}\left(\operatorname{Neure}, \phi\right) \geq \operatorname{Som} \beta \right), \\ & \operatorname{locate}_{A}\left(\operatorname{Atom}_{A}, \phi\right) = \sum_{i} \operatorname{Som} \beta \right), \\ & \operatorname{locate}_{A}\left(\operatorname{Atom}_{A}, \phi\right) = \sum_{i} \operatorname{Som} \beta \right), \\ & \operatorname{locate}_{A}\left(\operatorname{Neure}, \phi\right) \geq \operatorname{Nous} \beta \right), \\ & \operatorname{locate}_{A}\left(\operatorname{Neure}, \phi\right) \geq \operatorname{Nous} \gamma \right), \\ & \operatorname{locate}_{A}\left(\operatorname{Neure}, \phi\right) \geq \operatorname{Nous}_{A}\left(\operatorname{Neure}, \phi\right), \\ & \operatorname{locate}_{A}\left(\operatorname{Neure}, \phi\right) \geq \operatorname{Neure}_{A}\left(\operatorname{Neure}, \phi\right), \\ & \operatorname{locate}_{A}\left(\operatorname{Neure}, \phi\right) \geq \operatorname{locate}_{A}\left(\operatorname{Neure}, \phi\right), \\ & \operatorname{locate}_{A}\left(\operatorname{Neure}, \phi\right) \geq \operatorname{locatee}_{A}\left(\operatorname{Neure}, \phi\right), \\ & \operatorname{locatee}_{A}\left(\operatorname{Neure}, \phi\right), \\ & \operatorname{locatee}_{A}\left(\operatorname{Neure}, \phi\right) \geq \operatorname{locatee}_{A}\left(\operatorname{Neure}, \phi\right), \\ & \operatorname{locatee}_{A}\left(\operatorname{Neure}, \phi\right), \\ & \operatorname{locatee}_{A}\left(Neur$ | The state of the s | Tendency Javes Agency Agency Agency Javes 1: monthing Javes Agency Javes 2: monthing Communications 2: monthing Communications 3: | continue parally lamparings, active transit bottom for depended and additional questionated with data depending commit dependently of a multifaction under temporal additional questionated with data depending commit dependency of a multifaction under temporal additional questionated and data dependency of a multifaction under temporal additional questionated in the continue of the |
| Boardin (Lett., s_1^2) = Steam β). (boardin (Sharata, γ_1) = Steam β). (boardin (Arroman, s_1) = Steam β). (boardin (Arroman, s_2) = Steam β). (boardin (Arroman, s_3) = Steam β). (boardin (Steam, s_3) = Steam β). | with front a problem a proper $g = mL_1$ around g -mod g -m | Tendency Javes Agency Agency Agency Javes 1: monthing Javes Agency Javes 2: monthing Communications 2: monthing Communications 3: | continue para humanique, activir de michi botto in el equencio del estido quebecció esti des dependen y comó dependen y en enflutare untri segores delle vide de del colò respecto del estido del colò respecto del colò respecto del estido del colò respecto del estido del colò respecto del estido del colò respecto del colò respe |
| Descript (Lows, a.g.) is from 0.5 | The state of the s | Tendency Javes Agency Agency Agency Javes 1: monthing Javes Agency Javes 2: monthing Communications 2: monthing Communications 3: | continue approachemic agreement and the continue of the agreement and the dependency of an authorities with the agreement and the dependency of an authorities with the agreement and the dependency of an authorities with the agreement and the dependency of the authorities and the agreement and the ag |
| $\begin{aligned} & \operatorname{Geometric}_{A}(\operatorname{Hom} x, -) & \operatorname{Sim} \theta_1 \wedge \\ & \operatorname{Geometric}_{A}(\operatorname{Hom} x, -) & \operatorname{Sim} \theta_1 \wedge \\ & \operatorname{Hom} \theta_1 \wedge \\ & \operatorname{Hom} \theta_1 \wedge \\ & \operatorname{Hom} \theta_2 \wedge \\ & \operatorname{Hom} \theta_1 \wedge \\ & \operatorname{Hom} \theta_2 \wedge \\ & \operatorname{Hom} \theta_1 \wedge \\ & \operatorname{Hom} \theta_1 \wedge \\ & \operatorname{Hom} \theta_2 \wedge \\ & \operatorname{Hom} \theta_1 \wedge \\ & \operatorname{Hom} \theta_2 \wedge \\ & \operatorname{Hom} \theta_1 \wedge \\ & \operatorname{Hom} \theta_2 \wedge \\ & \operatorname{Hom} \theta_1 \wedge \\ & \operatorname{Hom} \theta_2 \wedge \\ & \operatorname{Hom} \theta_2 \wedge \\ & \operatorname{Hom} \theta_1 \wedge \\ & \operatorname{Hom} \theta_2 \wedge \\ & Ho$ | The state of the s | Tendency Javes Agency Agency Agency Javes 1: monthing Javes Agency Javes 2: monthing Communications 2: monthing Communications 3: | continues and production and the continues of the continu |
| $\begin{aligned} & \text{Bosterial (Lower, q)} = \frac{1}{1000} \cdot \frac{1}{1000} \cdot \frac{1}{1000} \\ & \text{Bosterial (Lower, q)} = \frac{1}{10000} \cdot \frac{1}{1000} \cdot \frac{1}{1000} \\ & \text{Bosterial (Lower, q)} = \frac{1}{10000} \cdot \frac{1}{10000} \cdot \frac{1}{10000} \cdot \frac{1}{10000} \\ & \text{Bosterial (Lower, q)} = \frac{1}{10000} \cdot \frac{1}{10000} \cdot \frac{1}{10000} \cdot \frac{1}{10000} \cdot \frac{1}{100000} \\ & \text{Bosterial (Lower, q)} = \frac{1}{1000000} \cdot \frac{1}{10000000000000000000000000000000000$ | The state of the s | The state of the s | continues and production and the state of th |
| $\begin{aligned} & \text{Geometric (Lower, q)} = \frac{1}{1000} \sin \theta \cdot \hat{\theta} \\ & \text{Howeld (Name, q)} = \frac{1}{1000} \sin \theta \cdot \hat{\theta} \\ & \text{Howeld (Name, q)} = \frac{1}{1000} \sin \theta \cdot \hat{\theta} \\ & \text{Howeld (Name, q)} = \frac{1}{1000} \sin \theta \cdot \hat{\theta} \\ & \text{Howeld (Name, q)} = \frac{1}{1000} \sin \theta \cdot \hat{\theta} \\ & \text{Howeld (Name, q)} = \frac{1}{1000} \sin \theta \cdot \hat{\theta} \\ & \text{Howeld (Name, q)} = \frac{1}{1000} \sin \theta \cdot \hat{\theta} \\ & \text{Howeld (Name, q)} = \frac{1}{1000} \sin \theta \cdot \hat{\theta} \\ & \text{Howeld (Name, q)} = \frac{1}{1000} \sin \theta \cdot \hat{\theta} \\ & \text{Howeld (Name, q)} = \frac{1}{1000} \sin \theta \cdot \hat{\theta} \\ & \text{Howeld (Name, q)} = \frac{1}{1000} \sin \theta \cdot \hat{\theta} \\ & \text{Howeld (Name, q)} = \frac{1}{1000} \sin \theta \cdot \hat{\theta} \\ & \text{Howeld (Name, q)} = \frac{1}{1000} \sin \theta \cdot \hat{\theta} \\ & \text{Howeld (Name, q)} = \frac{1}{1000} \sin \theta \cdot \hat{\theta} \\ & \text{Howeld (Name, q)} = \frac{1}{1000} \sin \theta \cdot \hat{\theta} \\ & \text{Howeld (Name, q)} = \frac{1}{1000} \sin \theta \cdot \hat{\theta} \\ & \text{Howeld (Name, q)} = \frac{1}{1000} \sin \theta \cdot \hat{\theta} \\ & \text{Howeld (Name, q)} = \frac{1}{1000} \sin \theta \cdot \hat{\theta} \\ & \text{Howeld (Name, q)} = \frac{1}{1000} \sin \theta \cdot \hat{\theta} \\ & \text{Howeld (Name, q)} = \frac{1}{1000} \sin \theta \cdot \hat{\theta} \\ & \text{Howeld (Name, q)} = \frac{1}{1000} \sin \theta \cdot \hat{\theta} \\ & \text{Howeld (Name, q)} = \frac{1}{1000} \sin \theta \cdot \hat{\theta} \\ & \text{Howeld (Name, q)} = \frac{1}{1000} \sin \theta \cdot \hat{\theta} \\ & \text{Howeld (Name, q)} = \frac{1}{1000} \sin \theta \cdot \hat{\theta} \\ & \text{Howeld (Name, q)} = \frac{1}{1000} \sin \theta \cdot \hat{\theta} \\ & \text{Howeld (Name, q)} = \frac{1}{1000} \sin \theta \cdot \hat{\theta} \\ & \text{Howeld (Name, q)} = \frac{1}{1000} \sin \theta \cdot \hat{\theta} \\ & \text{Howeld (Name, q)} = \frac{1}{1000} \sin \theta \cdot \hat{\theta} \\ & \text{Howeld (Name, q)} = \frac{1}{1000} \sin \theta \cdot \hat{\theta} \\ & \text{Howeld (Name, q)} = \frac{1}{1000} \sin \theta \cdot \hat{\theta} \\ & \text{Howeld (Name, q)} = \frac{1}{1000} \sin \theta \cdot \hat{\theta} \\ & \text{Howeld (Name, q)} = \frac{1}{1000} \sin \theta \cdot \hat{\theta} \\ & \text{Howeld (Name, q)} = \frac{1}{1000} \sin \theta \cdot \hat{\theta} \\ & \text{Howeld (Name, q)} = \frac{1}{1000} \sin \theta \cdot \hat{\theta} \\ & \text{Howeld (Name, q)} = \frac{1}{1000} \sin \theta \cdot \hat{\theta} \\ & \text{Howeld (Name, q)} = \frac{1}{1000} \sin \theta \cdot \hat{\theta} \\ & \text{Howeld (Name, q)} = \frac{1}{1000} \sin \theta \cdot \hat{\theta} \\ & \text{Howeld (Name, q)} = \frac{1}{1000} \sin \theta \cdot \hat{\theta} \\ & \text{Howeld (Name, q)} = \frac{1}{1000} \sin \theta \cdot \hat{\theta} \\ & \text{Howeld (Name, q)} = \frac{1}{1000} \sin \theta \cdot \hat{\theta} \\ & \text{Howeld (Name, q)} = \frac{1}{1000} \sin \theta \cdot \hat{\theta} \\ & \text{Howeld (Name, q)} = \frac{1}{10000} \sin \theta \cdot \hat{\theta} \\ & \text{Howeld (Name, q)}$ | The second secon | The state of the s | continued process of the continued of th |
| Bosonia (Rose, q_1) is from A). Hence the Charles (Rose, q_2) is from B in the Charles (Rose, B) is from B in B . Hence B is from B in B | The state of the s | material planed | continued process and continued and continued process and continue |
| Denote (Lower, q_1) is from q_1 . From the following (Name, q_2) is from q_1 . In the following (Name, q_2) is from q_3 . In the following (Name, q_3) is from q_4 . In the following (Name, q_4) is from q_4 . In the following (Name, q_4) is from q_4 . In the following (Name, q_4) is from q_4 . In the following (Name, q_4) is from q_4 . In the following (Name, q_4) is from q_4 . In the following (Name, q_4) is from q_4 . In the following (Name, q_4) is from q_4 . In the following (Name, q_4) is from q_4 . In the following (Name, q_4) is from q_4 . In the following (Name, q_4) is from q_4 . In the following (Name, q_4) is from q_4 . In the following (Name, q_4) is from q_4 . In the following (Name, q_4) is from q_4 . In the following (Name, q_4) is from q_4 . In the following (Name, q_4) is from q_4 . In the following (Name, q_4) is from q_4 . In the following (Name, q_4) is from q_4 . In the following (Name, q_4) is q_4 . In the following (Name, q_4) is q_4 . In the following (Name, q_4) is q_4 . In the following (Name, q_4) is q_4 . In the following (Name, q_4) is q_4 . In the following (Name, q_4) is q_4 . In the following (Name, q_4) is q_4 . In the following (Name, q_4) is q_4 . In the following (Name, q_4) is q_4 . | The state of the s | material planed | continued processing a plant of the continued before a defined understand with dispension of the dependency of an artification with legacine before within the district and the suppress of dish official and the continued and the dependency of an artification with legacine before within the district and the suppress of district and the |
| Descript (Descript) is from h in the form of h | The second secon | Tendency John Cybron Cy | continued processing a plant of the continued before a defined understand with dispension of the dependency of an artification with legacine before within the district and the suppress of dish official and the continued and the dependency of an artification with legacine before within the district and the suppress of district and the |
| Description Contemporary Conte | The second secon | material planed | continued process of the continued of th |
| Describe (Bown, a_1) is then a_1 becomes (Nature, a_2) is then a_1 becomes (Nature, a_2) is then a_1 becomes (Nature, a_2) is then a_2 becomes a_2 becomes a_3 becomes a_4 beco | The second secon | Tendency John Cybron Cy | continue approachemic agreement and the continue of the agreement and the dependency of an authorities with the agreement and the dependency of an authorities with the agreement and the dependency of an authorities with the agreement and the dependency of the authorities and the agreement and the ag |

The full model

| | ls_store a = case a of Storm→T _ → F | | |
|---|--|--|--|
| $a \stackrel{\leftarrow}{\rightarrow} b = (a, b) \in r$ | is force $a = case \ a$ of Frence | trachment reports = some flowed in report Virginisie price is some flowed in report Virginisie price is | while, date, effort, are actions through location hind sequenced before additional synchronized with date dependency control dependency toggene before : [As it happens whiten like (A) : As its "While, like, like," it string through location-less dependency happens before a bit "While, like, like," it string through location-less dependency happens before a bit "While, like, like," it is that through location is a like the like and like the like |
| $a \land b \equiv (a, b) \in r$ | leglock_or_unlock a = leglock a v legmbork a | | |
| $a \stackrel{\leftarrow}{>} b = (a,b) \notin r$ | Icatenicaction a :: | $\begin{array}{c} \text{release} \text{-} \text{experience} = \lambda_{ad} \xrightarrow{\text{descensions}} b = \\ \text{le_{a}t_{a}atomic_{a}boutloom} b \land \end{array}$ | visible-sequence of obserfators and is visible-sequence of obserfators and was lead to ≡ (c. var_lead to |
| <u>-</u> | ligatomic Jonal a V ligatomic ptore a V ligatomic grave a | λ_{a} (b) λ_{a} λ_{b} λ_{a} λ_{b} λ_{a} | (2) Augmentator (2) /. (2) Augmentator (2) Augmentator (3) Augmentator (4) Au |
| $a \stackrel{\leftarrow}{\rightarrow} b \stackrel{\rightarrow}{\rightarrow} c \equiv a \stackrel{\leftarrow}{\rightarrow} b \wedge b \stackrel{\rightarrow}{\rightarrow} c$ | iclosducatore a = iclosd a ∨ icatore a | $(\forall c. \lambda_{ac} \xrightarrow{matilization analog} c \xrightarrow{matilization analog} b \Longrightarrow \iota_{\lambda_{ac}} \underbrace{\iota_{ac}}_{\lambda_{ac}} (\forall c. \lambda_{ac}))$ | ⇒ -(b - (b |
| | is rood a = | | $m_0(m_0 p, f = (p, 2n \in L, (p = f = p))$ |
| relation, over a ref = domain ref \subseteq a \land range ref \subseteq a | le ptomic load a V le ptomic runn a V le Joad a | ninam-sequence set actions through location-kind sequenced-before additional-synchronized-with data-dependency control-dependency modification-order :: minus_sequence actions through location-kind sequenced-before additional-synchronized-with data-dependency control-dependency modification-order a bit | visible proposes at side effects = visible proposes at side effects = News host b. |
| $\stackrel{nl}{\longrightarrow} _{r}\equiv nnl\ \cap\ (\alpha\times\alpha)$ | is_write a :: is_atomic_store a \lor is_atomic_rraw a \lor is_atom a | hypothetical release_sequence = a hypothetical colors sequence b b = | (b, W in-at-actesia-location b then (upon-local) U |
| $rel _{a} = rel \cap (a \times a)$ | | local attention location b A ((b = a) V | Coldinary given with an internal and companied to the coldinary given with a coldinary given with a coldinary given and to the coldinary given a coldinary g |
| \xrightarrow{ab} L = $aaf \cap (a \times ab)$ | lo_acquire a = (case memory_order a of Scout: mere_ord → | (n_derance a b \(\rangle \rangle \) a multivation arrive \(b \rangle \) (\(\chi_{\pm} \) a multivation arrive \(\chi_{\pm} \) \(\chi_{\p | 07 |
| | (mem_ord ∈ {Mo_wregens, Mo_wreg_ms, Mo_wreg_cer} ∧ (b. rand a ∨ b. frace all ∨ | n_decent a c))) | siable segences and self-reference at ratios through functions are fastered under a distinct of segences of self-reference and self-research and self-reference and self-research and self-research and self-research and self-reference and self-research and self-reference and self-refere |
| $nel_{ja}=nel\cap(a\times a)$ | (*ZNLS STATE STATE CONTENTS AND CONTENTS AND CONTENTS AND CONTENTS AND CONTENTS AND AND C | hypothetical release sequences set actions through location-kind sequenced-before additional-synchronized-with data-dependency control-dependency modification-order: | condetent yearly from yeapping = condetent yearly from yeapping = |
| $\operatorname{strict}_{\underline{a}} percedur \ \text{ord} = \operatorname{invellective} \ \text{ord} \wedge \operatorname{trans} \ \text{ord}$ | Nove → ic lock a) | hypothetical release sequence actions through location-kind sequenced-before additional-synchronized-with data-dependency control-dependency modification-order a b) | (b) (i.e. on $b \land b \land a$ in a simulation a) (iii) (|
| total_cover s and = relation_cover s and A | ls_consume a :: ls_read a \((memory_order a :: Some Mo_consume) \) | synchronization_with = 2 = (** = 2 = ***) b = (* = additional synchronization, for thread create etc **) | thus $(z_{m_1}, z_{m_2}, z_{m_3}) + \delta \wedge z_{m_2} \leq \delta)$ thus $(z_{m_1}, z_{m_2}, z_{m_3}) \wedge \delta \wedge z_{m_2} \leq \delta)$ |
| $(\forall x \in \epsilon. \ \forall y \in \epsilon. \ x \xrightarrow{ad} y \lor y \xrightarrow{ad} x \lor (x = y))$ | | 3 additional aprohessional stable 3 V | (m,(a,aab,b),(a,aab,aab,aab,aab,b) = (a,aab,aab,aab,aab,aab,aab,aab,aab,aab,a |
| strict_total_order_over s and = | (case assumpty_order a of Sourcemen.ord | (some_location Dh h $a \in actions h$ h $b = actions h$ ($(*-mates upschreakminion - *)$) $(u-minks h$ $h \in action h$ $a \mapsto b$) v | then $(\zeta ^2, van)^2$ $(v''' v''' v''' v''' v''' v''' v''' v'$ |
| strict_preorder and \(\lambda\) total_ever a and | mem_ord (Mo_periors_Mo_acq_rest, Mo_peri_cer) (to_verior_s v in_fence a) Nover_s t_mands a) | (* - release/acquire werchronization - *) | eta ~(2x 0))) \cdot \ |
| $x \xrightarrow{rod}_{pool} y =$ $pool} x \wedge x \xrightarrow{soc} y \wedge \neg (\exists x. pool} x \wedge x \xrightarrow{soc} x \xrightarrow{soc} y)$ | None → is_malock a) | (is_arbinos a \(\hat{\chi_acception}\) b \\ \((\lambda \chi_acception\) \(\hat{\chi}\) \\ \((\lambda \chi_acception\) \(\hat{\chi}\) \\ \((\lambda \chi_acception\) \\\ | ((c, a) ∈ \(\frac{A}{\pi}\) ((c) \(\frac{A}{\pi}\) \(\frac{A}{\pi}\) ((d) \(\frac{A}{\pi}\) |
| | is_sequent a = (memory_serder a = Sonm: Mo_semp_sem) | (* – feece synchronization – *) (Indiance a / Indiance a | y = manufactation of h (n at atomic decention b ⇒ (x - y) ∀ x millionared → y) ∧. |
| $x \stackrel{\text{sd}}{\Longrightarrow} y =$ $x \stackrel{\text{sd}}{\Longrightarrow} y \wedge \neg (\exists x. x \stackrel{\text{sd}}{\Longrightarrow} x \stackrel{\text{sd}}{\Longrightarrow} y)$ | location_kind = Murrox | (lix lly, some heatins xy /. is considerately as 'h, deatinis' artists y /h (a retin x /s. x appendiction x /s.) x appendiction x /s. x appendiction x /s. | (* noc 2008 f.) = 70 2 |
| well-founded $r = wf r$ | Nox_stome Atome | $2 x \wedge y b \wedge$ $(2x \times \frac{y_{year-balled}}{y_{year}} x y))) \vee$ | Vc. c [∞] b ∧ |
| | actions_respect_location_kinds = actions_respect_location_kinds = | (is_frace a ∧ is_triknee a ∧ is_acquire b ∧ is_acquire b ∧ | lagration a A name_location in the A lagrationing discretion to \(\sum_{in} \) (cm \) (a) \(\sum_{in} \) (cm \) (cm \) (*non-collect*) |
| type_ibbeev action_id : string | Va. case location a of Soon: /→ | (Br. mare-location x b A. le constit action x b A. a quessionables x A. a page of the constituents of t | (Victor) (Vi |
| type_abboov thread_id : string | (case incarior-kind I of MITTEX > k_block_per_pulsock a Non_xyrounc -> k_block_per_pulsoce a | $\frac{3}{(2x \cdot x)} \xrightarrow{\text{hymiltotial arbitrars ansurance}} x \xrightarrow{d_1} b))) \lor$ | c 2 A A Court of the second boundary boundary and the second bound |
| type_abbrev location : string | Arounc → is_load_or_store a ∨ is_atomic_action a) None → T | (is_atomic_action a / is_minuse a / is_finese b / is_acquire b / i | The Committee of the Co |
| type_abbon val : string | | (2x seems backling a x A A A battering backling x A A x seems backling b A (2x x seems seems y → x x)))))) | $(v_i, b_i) \in \mathcal{A}_{i-1}$ is actional curve b $v_i = a_i$ is $a_i = a_i$ in $b_i = a_i$. |
| | case location a of Sone: I (location-kind I = Bd) None: F | (3x, x → → x → x))))) | $(t(x_i)) \in \stackrel{d}{\sim} t_i \text{ longuest } b$ $\Longrightarrow (v_i)_{i=1}^{n} (t(x_i))_{i=1}^{n} (t(x_i))$ |
| memory_code_cours = Mo_MENQ_CET MO_MENA_CET | NOME → F | synchronizes_with_est actions (Locusian Income), location-lind sequenced define additional-synchronized with data-dependency control-dependency of modification-order sc release-sequence hypothetical-velase-sequence : | $A = 2a$. System commoderation $b \in B \setminus A$ |
| Модильник Модильник Модильнык | le_at_mutex_location a :: le_at_location_kind a Murrox | ryschronium_with actions through location-kind sequenced-before additional-synchronized-with data-dependency control-dependency of modification-order screlates-sequence hypothetical-educate sequence a b) | (*-Free restrictions *) (*-20.25 *) |
| Mo_ecquest | is_at_non_stonic_location s = | $a((-i) \xrightarrow{\text{supermed latery}} b) = \underbrace{a^{\text{supermed latery}}}_{b} b =$ | $(x,y(x,b)) = \max_{x \in \mathcal{X}} (x,y(x,b))$ (by the $x \in \mathcal{X}$ by $y(x,y(x,b)) = y(x,y(x,b))$ |
| action :: Lock of action id thread id location | lean docation kind a Non-arrounc | | by write a b complexation a b b a b b a b |
| Unitoric of action id thread id location Account action of action id thread id memory order enum location val | ls_at_stenic_location a :: ls_at_location_bind a Arounc | carrieridependency_ta_set actions through. location-kind sequenced before additional-spechenolous-with data-dependency control-dependency of = carrieri_dependency_ta_cations through location-kind sequenced before additional-spechenolous-with data-dependency control-dependency of a b) | $\Rightarrow (y = z) \lor z$ $\xrightarrow{\text{miditation modes}} y) \land (^{*}2.3.4^{*})$ |
| Aronne prome of action id thread id memory order enum location val Aronne pure of action id thread id memory order enum location val val Loan of action id thread id location val | | describer wheel letter | Constitution (by the Constitution of the Const |
| Strong of action,id thread,id location val Fronce of action,id thread,id memory,order,enum | same_thread a b = (thread_id_id_iof a = thread_id_iof b) | dependency action d_solice = x = | Secrette a A connectication a 2 h x 5 h A is natural nearties 0) |
| (action_id_of (Locx aid) = aid) ∧ | threadwise, relation, over x set $=$ relation, over x set $(Y(x,b) \in rel.$ some thread x $b)$ | (2s, 3 - short sequence, $\rho \stackrel{d}{\rightarrow} b) \land$ (b - series adoptedness $b \stackrel{d}{\rightarrow} (b = d)))$ | $\Rightarrow (y = a) \lor x \xrightarrow{\text{confinements}} y) \land$ $(*20.55 *)$ |
| (action_id_id (Union; aid _) = aid) \(\) (action_id_id (Aroun_io.on aid) = aid) \(\) (action_id_id (Aroun_io.on aid) = aid) \(\) | some_location $s b = (\text{location } s = \text{location } b)$ | dependency_ardered_before_met actions through location-kind sequenced-before additional-synchronized-with data-dependency control-dependency of modification-order release-sequence carrier-a-dependency-to :: | (**) A 1-2 $\frac{(r_1, r_2)}{r_1} = \frac{r_1}{r_2} \frac{r_2}{r_2} \frac{r_2}{r$ |
| (action_id_ad (Aronne_aron aid) = aid) ∧ (action_id_ad (Loon aid) = aid) ∧ (action_id_ad (Sroom aid) = aid) ∧ (action_id_ad (Fronc aid) = aid) ∧ | locations of actions = {t. lin. (location n = Sourc t)} | dependency_andronLinding actions through location-kind sequenced-before additional-synchronized-with data-dependency control-dependency of modification-order release-sequence carriers-dependency to a b) | h_{sp} -decoding action in h consultance is a $h \land x \in \mathcal{A}$ and h consultance is h and h consultance in h and h consultance is h and h consultance in h consultance |
| (action island (Frence and) = and) | | simple_happens_before = \(\frac{\text{simple_happens_before}}{\text{c}}\) = | $\Rightarrow (z=z) \lor z \xrightarrow{\text{millionis sole}} z)$ |
| $(\operatorname{thread}_a\operatorname{id}_a\operatorname{id}(\operatorname{Lock}_a\operatorname{rid}_a)=\operatorname{rid}) \wedge \\ (\operatorname{thread}_a\operatorname{id}_a\operatorname{id}(\operatorname{Uncock}_a\operatorname{rid}_a)=\operatorname{rid}) \wedge $ | well_formed_action s = case s of Arounc_sono _ mem_ord _ → mem_ord ∈ | (supercord about) continuous miles) (| $\ \mathcal{L}_{\mathcal{L}_{i,j}}\ _{L^{\infty}(\mathbb{R}^{N})} \leq \frac{1}{N} \ \mathcal{L}_{i,j}\ _{L^{\infty}(\mathbb{R}^{N}$ |
| (thread_id_of (ATOMIC_LOADidd) = idd) ∧ (thread_id_of (ATOMIC_LTOREidd) = idd) ∧ | $\{Mo. nelaxed, Mo. acquere, Mo. acquere, Mo. constant}\}$ Arordic storic mem.ord → mem.ord ∈ | Consistent_utimple_happens_before aft0 :: tentilentes_of AMA_1 | V = |
| | | | |

We can reason about C concurrency!



Shared memory

```
int a = 1;
int b = 0;
```

Thread 1

```
int s;
for (s=0; s!=4; s++) {
   if (a==1)
     return NULL;
   for (b=0; b>=26; ++b)
   ;
}
```

Thread 2

```
b = 42;
printf("%d\n", b);
```

Thread 2 is not affected by Thread 1 and vice-versa

This program is data-race free

This program must print 42

Shared memory

```
int a = 1;
```

This is a compiler bug

```
for (s=0; s!=4; s++) {
    if (a==1)
      return NULL;
    for (b=0; b>=26; ++b)
    ;
}
```

Thread 2 is not affected by Thread 1 and vice-versa

This program is data-race free

This program must print 42

Shared memory

```
int a = 1;
```

This is a concurrency compiler bug

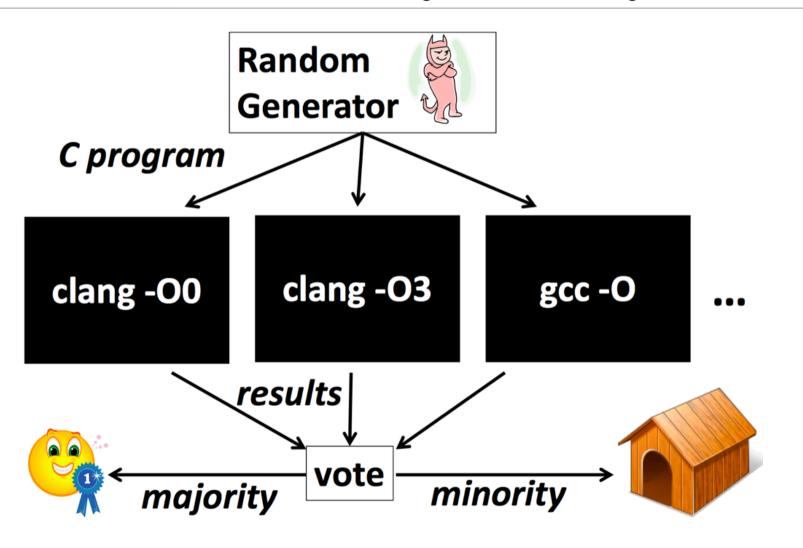
Thread 2 is not affected by Thread 1 and vice-versa

This program is data-race free

This program must print 42

Compiler testing: state of the art

Yang, Chen, Eide, Regehr - PLDI 2011



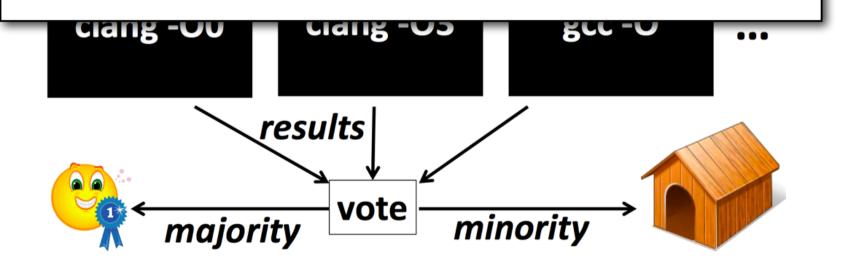
Compiler testing: state of the art

Yang, Chen, Eide, Regehr - PLDI 2011

Random

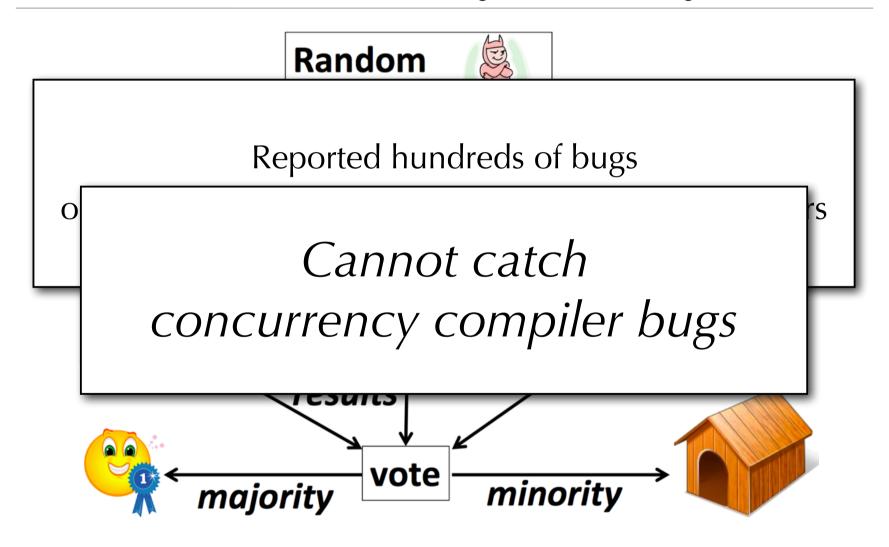


Reported hundreds of bugs on various versions of gcc, clang and other compilers



Compiler testing: state of the art

Yang, Chen, Eide, Regehr - PLDI 2011



Hunting concurrency compiler bugs?

How to deal with non-determinism?

How to generate non-racy interesting programs?

How to capture all the behaviours of concurrent programs?

A compiler can optimise away behaviours:

how to test for correctness?

limit case: two compilers generate correct code with disjoint final states

Idea

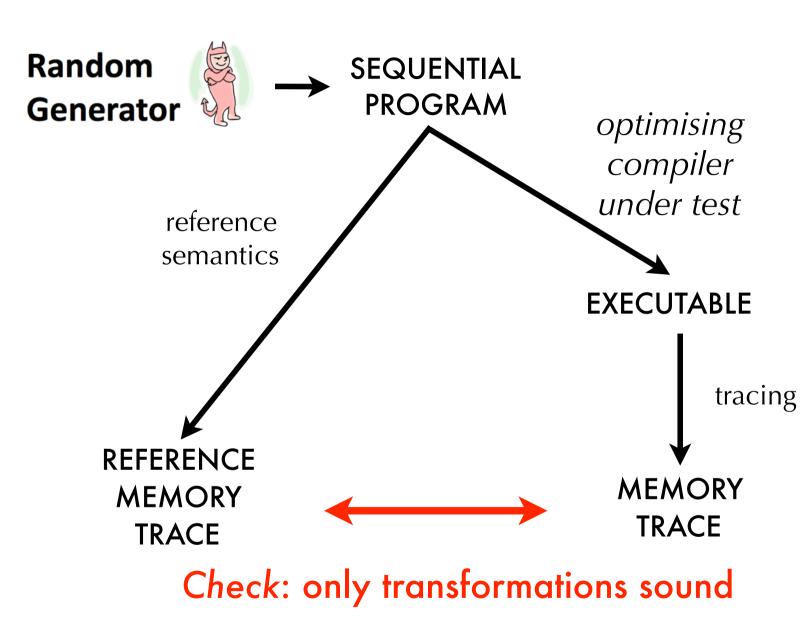
C/C++ compilers support separate compilation Functions can be called in arbitrary non-racy concurrent contexts

1

C/C++ compilers can only apply transformations sound with respect to an arbitrary non-racy concurrent context

Hunt concurrency compiler bugs

search for transformations of sequential code not sound in an arbitrary non-racy context

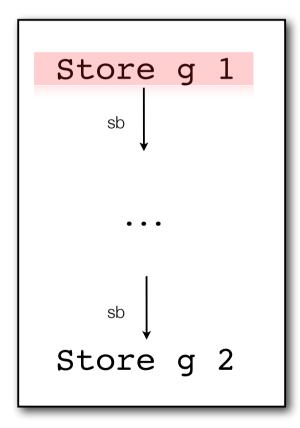


in any concurrent non-racy context

Soundness of compiler optimisations in the C11/C++11 memory model



Elimination of overwritten writes



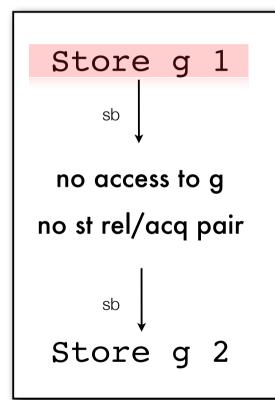
Under which conditions is it correct to eliminate the first store?

A same-thread release-acquire pair is a pair of a release action followed by an acquire action in program order.

An action is a *release* if it is a possible source of a synchronisation unlock mutex, release or seq_cst atomic write

An action is an *acquire* if it is a possible target of a synchronisation lock mutex, acquire or seq_cst atomic read

Elimination of overwritten writes



It is safe to eliminate the first store if there are:

- 1. no intervening accesses to g
- 2. no intervening same-thread release-acquire pair

The soundness condition

Shared memory

```
g = 0; atomic f1 = f2 = 0;
```

Thread 1

```
g = 1;
f1.store(1,RELEASE);
while(f2.load(ACQUIRE)==0);
g = 2;
```

The soundness condition

Shared memory

```
g = 0; atomic f1 = f2 = 0;
```

The soundness condition

Shared memory

$$g = 0$$
; atomic $f1 = f2 = 0$;

```
Thread 1 candidate overwritten write

g = 1;

f1.store(1,RELEASE); same-thread release-acquire pair while(f2.load(ACQUIRE)==0);

g = 2;
```

Shared memory

```
g = 0; atomic f1 = f2 = 0;
```

Thread 1

```
g = 1;
f1.store(1,RELEASE);
while(f2.load(ACQUIRE)==0);
g = 2;
```

Thread 2

```
while(f1.load(ACQUIRE)==0);
printf("%d", g);
f2.store(1,RELEASE);
```

Shared memory

```
g = 0; atomic f1 = f2 = 0;
```

Thread 1

Thread 2

```
g = 1;
f1.store(1,RELEASE); while(f1.load(ACQUIRE)==0);
while(f2.load(ACQUIRE)==0);
g = 2;
while(f1.load(ACQUIRE)==0);
f2.store(1,RELEASE);
```

Thread 2 is non-racy

Shared memory

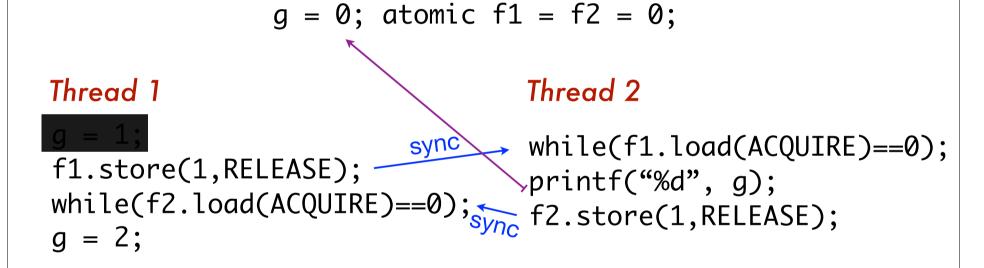
```
g = 0; atomic f1 = f2 = 0;
```

Thread 1

Thread 2

Thread 2 is non-racy
The program should only print 1

Shared memory



Thread 2 is non-racy
The program should only print 1

If we perform overwritten write elimination it prints 0

Shared memory

```
g = 0; atomic f1 = f2 = 0;
```

Thread 1

```
g = 1;
f1.store(1,RELEASE);
while(f2.load(ACQUIRE)==0);
g = 2;
```

Thread 2

```
while(f1.load(ACQUIRE)==0);
printf("%d", g);
f2.store(1,RELEASE);
```

Shared memory

$$g = 0$$
; atomic $f1 = f2 = 0$;

Thread 1

g = 1; f1.store(1,RELEASE);

g = 2;

Thread 2

```
while(f1.load(ACQUIRE)==0);
printf("%d", g);
f2.store(1,RELEASE);
```

Shared memory

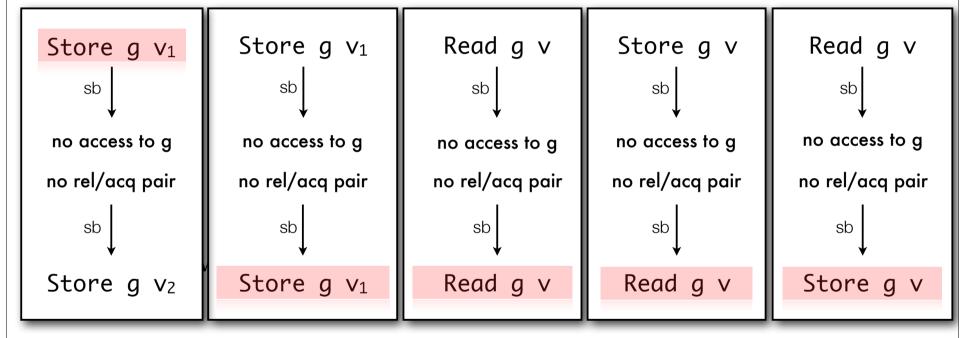
$$g = 0$$
; atomic $f1 = f2 = 0$;

Thread 1

Thread 2

If only a release (or acquire) is present, then all discriminating contexts *are racy*. It is sound to optimise the overwritten write.

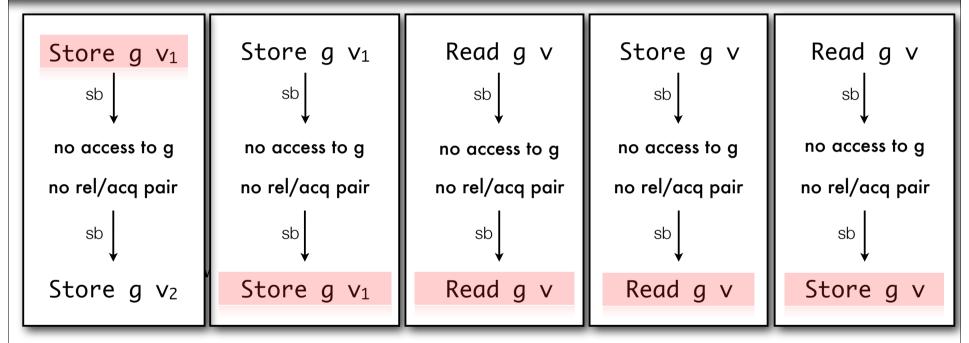
Eliminations: bestiary



Overwritten-Write Write-after-Write Read-after-Read Read-after-Write Write-after-Read

Reads which are not used (via data or control dependencies) to decide a write or synchronisation event are also eliminable (*irrelevant reads*).

Also correctness statements for reorderings, merging, and introductions of events.

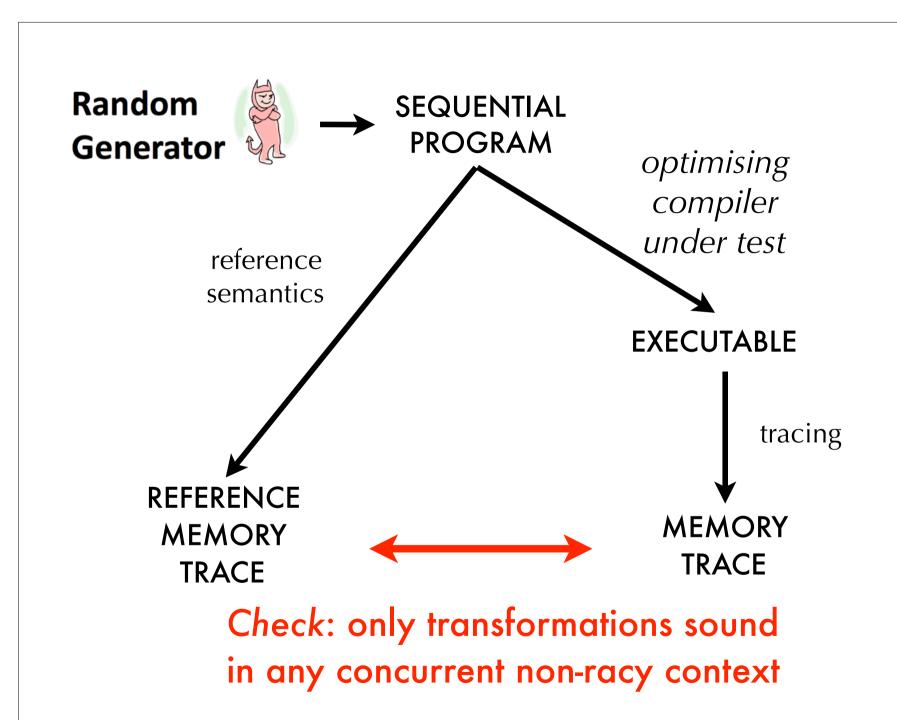


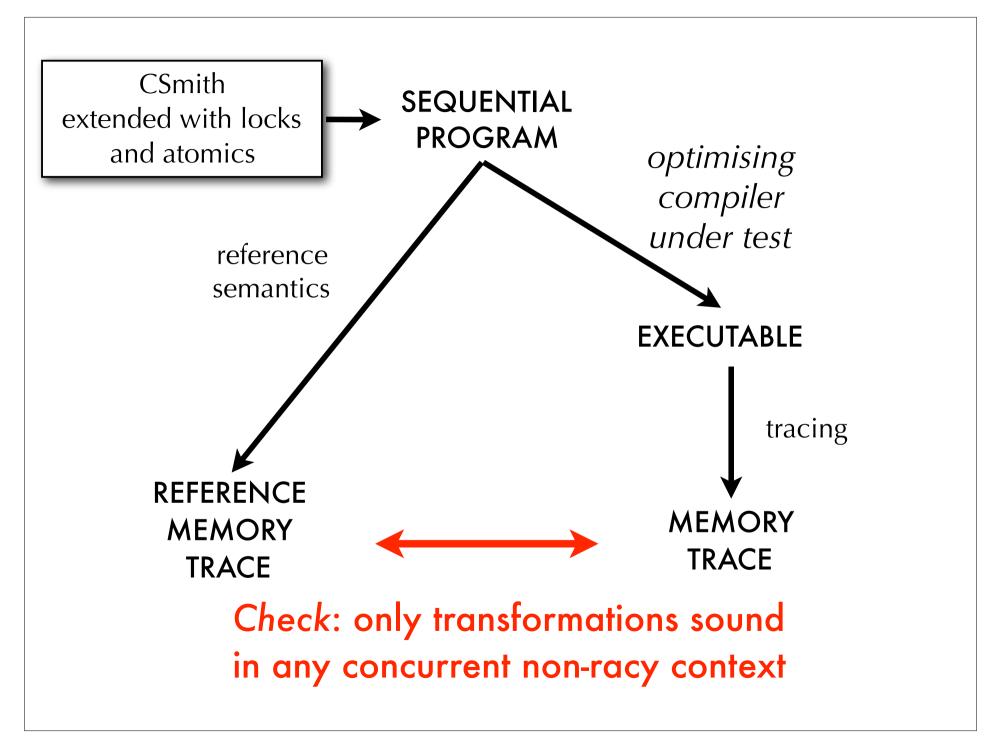
Overwritten-Write Write-after-Write Read-after-Read Read-after-Write Write-after-Read

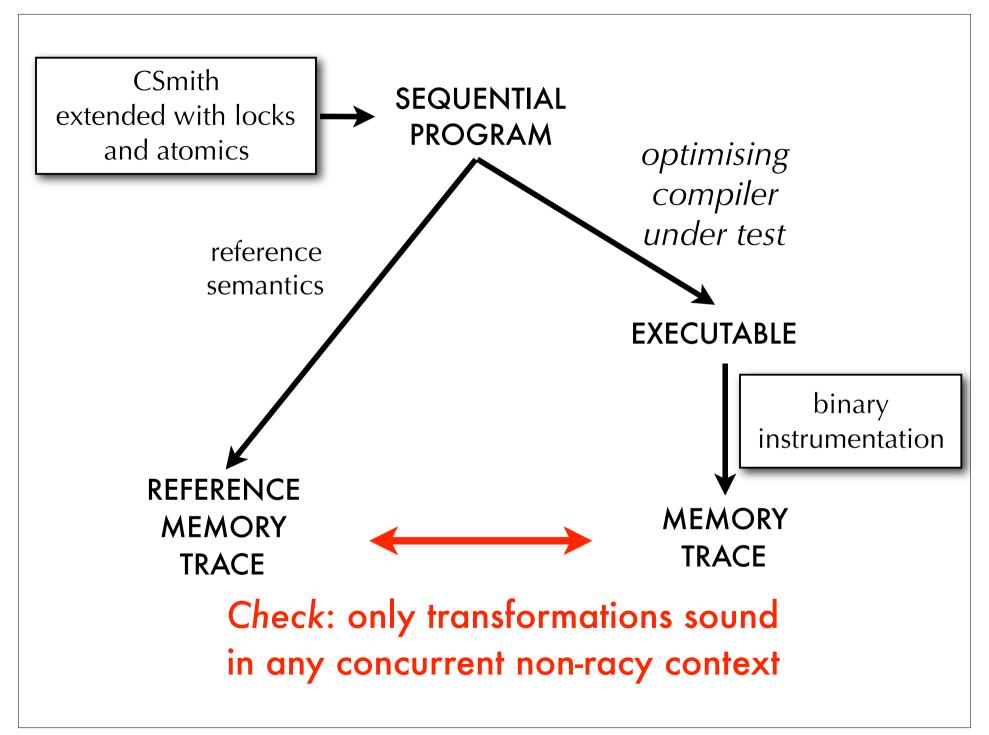
Reads which are not used (via data or control dependencies) to decide a write or synchronisation event are also eliminable (*irrelevant reads*).

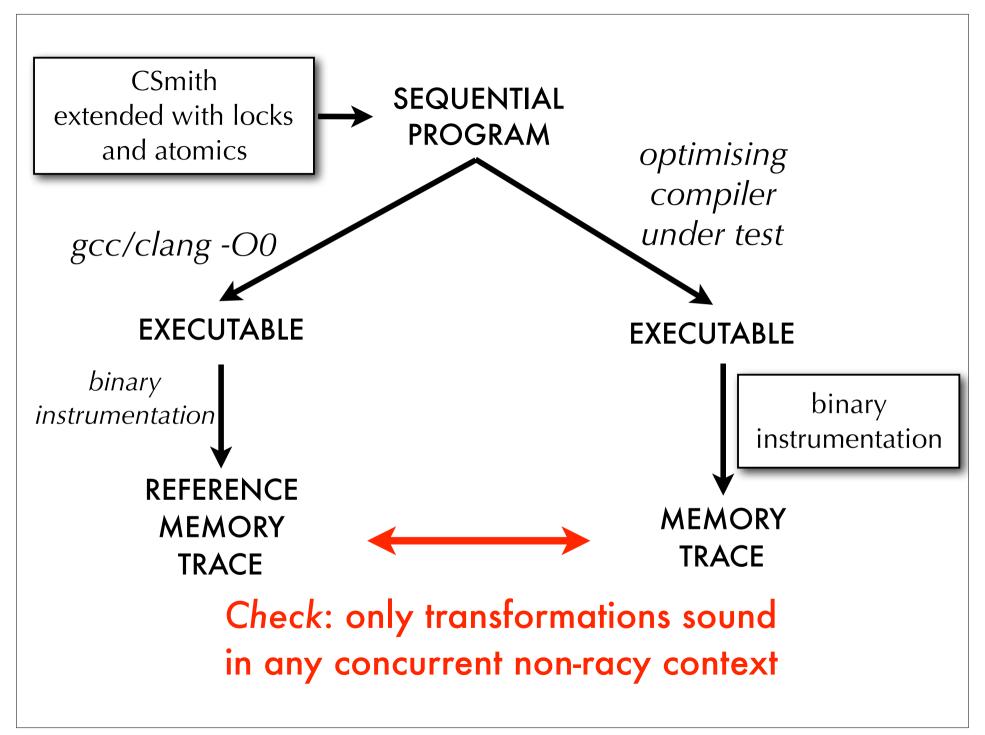
From theory to the Cmmtest tool

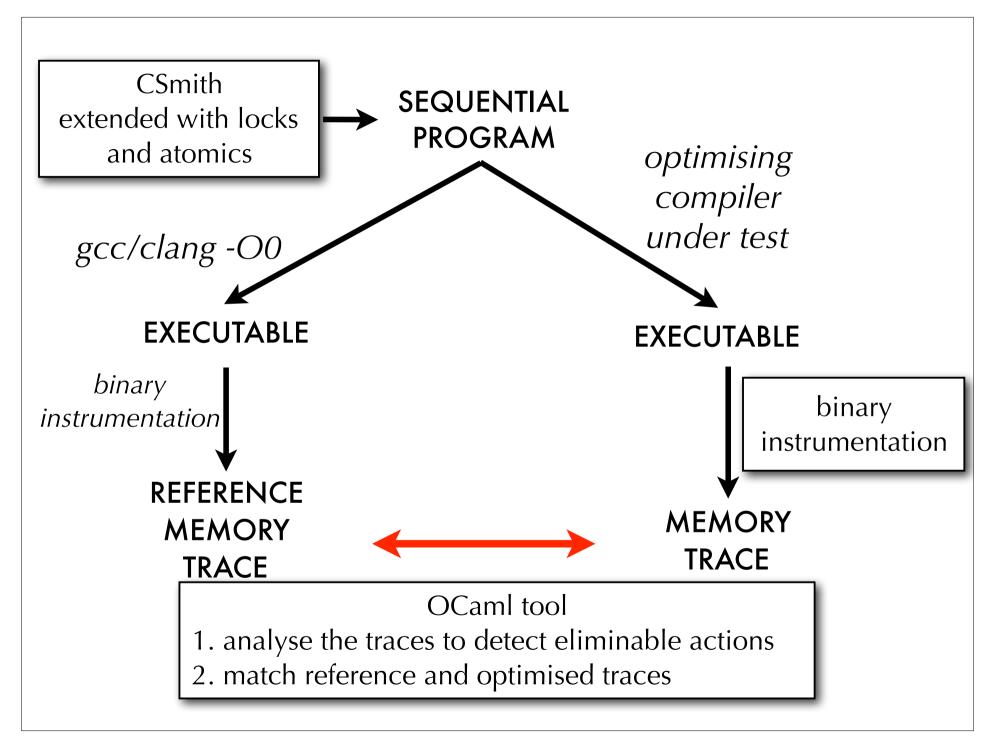












```
const unsigned int g3 = OUL;
long long g4 = 0x1;
int g6 = 6L;
volatile unsigned int g5 = 1UL;

void func_1(void){
   int *18 = &g6;
   int 136 = 0x5E9D070FL;
   unsigned int l107 = 0xAA37C3ACL;
   g4 &= g3;
   g5++;
   int *l102 = &l36;
   for (g6 = 4; g6 < (-3); g6 += 1);
   l102 = &g6;
   *l102 = ((*l8) && (l107 << 7)*(*l102));
}</pre>
```

Start with a randomly generated well-defined program

```
Init g3 0
Init g4 1
Init g5 1
Init g6 6
```

```
void func_1(void){
    int *l8 = &g6;
    int l36 = 0x5E9D070FL;
    unsigned int l107 = 0xAA37C3ACL;
    g4 &= g3;
    g5++;
    int *l102 = &l36;
    for (g6 = 4; g6 < (-3); g6 += 1);
    l102 = &g6;
    *l102 = ((*l8) && (l107 << 7)*(*l102));
}</pre>
```

```
void func_1(void){
Init g3 0
                           int *18 = &q6;
                           int 136 = 0x5E9D070FL;
Init g4 1
                           unsigned int 1107 = 0xAA37C3ACL;
Init g5 1
                           q4 \&= q3;
                           g5++;
Init g6 6
                           int *1102 = \&136;
                           for (g6 = 4; g6 < (-3); g6 += 1);
                           1102 = \&g6;
                           *l102 = ((*l8) && (l107 << 7)*(*l102));
      reference
      semantics __
       Load g4 1
       Store q4 0
       Load g5 1
       Store g5 2
       Store g6 4
       Load g6 4
       Load g6 4
       Load g6 4
       Store g6 1
       Load g4 0
```

```
void func_1(void){
Init g3 0
                          int *18 = \&g6;
                          int 136 = 0x5E9D070FL;
Init g4 1
                          unsigned int l107 = 0xAA37C3ACL;
Init g5 1
                          q4 \&= q3;
                          q5++;
Init g6 6
                          int *1102 = &136;
                          for (g6 = 4; g6 < (-3); g6 += 1);
                          1102 = &q6;
                          *1102 = ((*18) \&\& (1107 << 7)*(*1102));
      reference
                                       gcc -O2 memory trace
     semantics
       Load g4 1
       Store q4 0
                                       Load g5 1
       Load g5 1
                                        Store g4 0
       Store g5 2
                                       Store g6 1
       Store g6 4
                                        Store g5 2
       Load g6 4
                                       Load g4 0
       Load g6 4
       Load g6 4
       Store g6 1
       Load g4 0
```

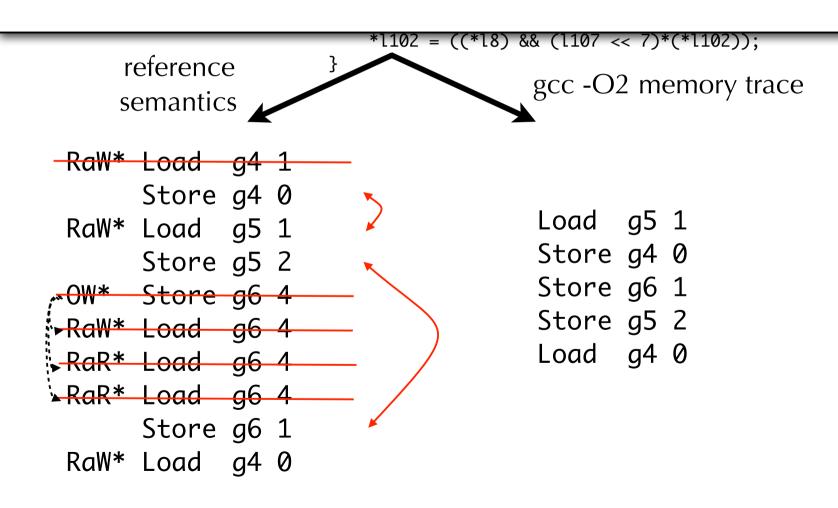
```
void func_1(void){
Init g3 0
                         int *18 = \&g6;
                         int 136 = 0x5E9D070FL;
Init g4 1
                         unsigned int 1107 = 0xAA37C3ACL;
Init g5 1
                         q4 \&= q3;
                         q5++;
Init g6 6
                         int *1102 = &136;
                         for (g6 = 4; g6 < (-3); g6 += 1);
                         1102 = &q6;
                         *1102 = ((*18) \&\& (1107 << 7)*(*1102));
      reference
                                      gcc -O2 memory trace
     semantics
 RaW* Load g4 1
       Store g4 0
                                       Load g5 1
 RaW* Load g5 1
                                       Store g4 0
       Store g5 2
                                       Store g6 1
.~OW* Store g6 4
                                       Store g5 2
≻RaW* Load g6 4
                                       Load g4 0
►RaR* Load g6 4
≽RaR* Load g6 4
       Store q6 1
 RaW* Load g4 0
```

```
void func_1(void){
Init g3 0
                           int *18 = &a6:
                           int 136 = 0x5E9D070FL;
Init g4 1
                           unsigned int 1107 = 0xAA37C3ACL;
Init g5 1
                           q4 \&= q3;
                           q5++;
Init g6 6
                           int *1102 = &136;
                           for (g6 = 4; g6 < (-3); g6 += 1);
                           1102 = &q6;
                           *1102 = ((*18) \&\& (1107 << 7)*(*1102));
      reference
                                         gcc -O2 memory trace
      semantics
RaW* Load g4 1
       Store q4 0
                                         Load g5 1
 RaW* Load q5 1
                                         Store g4 0
       Store g5 2
                                         Store g6 1
 <del>·OW* Store g6 4</del>
                                         Store g5 2
⊱<del>RaW* Load g6 4</del>
                                         Load g4 0
RaR* Load g6 4
'<u>- RaR* Load g6 4</u>
       Store g6 1
 RaW* Load g4 0
```

void func_1(void){
 int *18 = &g6;

Init g3 0

Can match applying only correct eliminations and reorderings



If we focus on the miscompiled initial example...

```
int a = 1; int s;
int b = 0;
                for (s=0; s!=4; s++) {
                  if (a==1)
                    return NULL;
                  for (b=0; b>=26; ++b)
     reference
     semantics
    Load a 1
```

```
int a = 1; int s;
                for (s=0; s!=4; s++) {
int b = 0;
                  if (a==1)
                    return NULL;
                  for (b=0; b>=26; ++b)
     reference
                                  gcc -O2 memory trace
     semantics
    Load a 1
                                 Load a 1
                                 Load b 0
                                 Store b 0
```

Cannot match some events —— detect compiler bug

reference semantics

Load a 1

gcc -O2 memory trace

Load a 1

Load b 0

Store b 0

Applications

2013 - 2015



1. Testing C compilers (GCC, Clang, ICC)

Some concurrency compiler bugs found in the latest version of GCC.

Store introductions performed by loop invariant motion or if-conversion optimisations.

Remark: these bugs break the Posix thread model too.

All promptly fixed.

2. Checking compiler invariants

GCC internal invariant: never reorder with an atomic access

Baked this invariant into the tool and found a counterexample...
...not a bug, but fixed anyway

3. Detecting unexpected behaviours

uint16_t g uint16_t g for (;
$$g==0$$
; $g=-$); \longrightarrow $g=0$;

Correct or not?

3. Detecting unexpected behaviours

uint16_t g uint16_t g for (;
$$g==0$$
; $g=-$); \longrightarrow $g=0$;

If g is initialised with 0, a load gets replaced by a store:

The introduced store cannot be observed by a non-racy context. Still, arguable if a compiler should do this or not.

3. Detecting unexpected behaviours

uint16_t g uint16_t g for (;
$$g==0$$
; $g=-$); \longrightarrow $g=0$;

If g is initialised with 0, a load gets replaced by a store:

False positives in Thread Sanitizer

The formalisation of the C11 memory model enables compiler testing... what else?



Proving the correctness of mappings for atomics

https://www.cl.cam.ac.uk/~pes20/cpp/cpp0xmappings.html

| r, | | |
|----|---------------------------|---|
| | C/C++11 Operation | ARM implementation |
| | Load Relaxed: | ldr |
| | Load Consume: | Idr + preserve dependencies until next kill_dependency OR Idr; teq; beq; isb OR Idr; dmb |
| | Load Acquire: | ldr; teq; beq; isb OR ldr; dmb |
| i | Load Seq Cst: | ldr; dmb |
| | Store Relaxed: | str |
| | Store Release: | dmb; str |
| | Store Seq Cst: | dmb; str; dmb |
| | Cmpxchg Relaxed (32 bit): | _loop: ldrex roldval, [rptr]; mov rres, 0; teq roldval, rold; strexeq rres, rnewval, [rptr]; teq rres, 0; bne _loop |
| | Cmpxchg Acquire (32 bit): | _loop: ldrex roldval, [rptr]; mov rres, 0; teq roldval, rold; strexeq rres, rnewval, [rptr]; teq rres, 0; bne _loop; isb |
| | Cmpxchg Release (32 bit): | dmb; _loop: ldrex roldval, [rptr]; mov rres, 0; teq roldval, rold; strexeq rres, rnewval, [rptr]; teq rres, 0; bne _loop; |
| | Cmpxchg AcqRel (32 bit): | dmb; _loop: ldrex roldval, [rptr]; mov rres, 0; teq roldval, rold; strexeq rres, rnewval, [rptr]; teq rres, 0; bne _loop; isb |
| | Cmpxchg SeqCst (32 bit): | dmb; _loop: ldrex roldval, [rptr]; mov rres, 0; teq roldval, rold; strexeq rres, rnewval, [rptr]; teq rres, 0; bne _loop; dmb |
| | Acquire Fence: | dmb |
| | Release Fence: | dmb |
| | AcqRel Fence: | dmb |
| | SeqCst Fence: | dmb |
| | | |

Inform new optimisations e.g. the work by Robin Morisset on the Arm LLVM backend

```
while (flag.load(acquire))
 {}
  .loop
   ldr r0, [r1]
   dmb ish
   bnz .loop
  .loop
   ldr r0, [r1]
   bnz .loop
    dmb ish
```









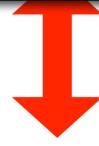


A word on CompCertTSO

Idea: the programming language memory model faithfully mimics the processor model.

Our we might want radically different programming languages!

(Radically different language = radically different challenges?)



A semantic preserving compiler, CompCertTSO



Intel processors implement the x86-TSO MM

Resources



http://www.cl.cam.ac.uk/~pes20/weakmemory/index.html

Starting point:

J. Sevcik

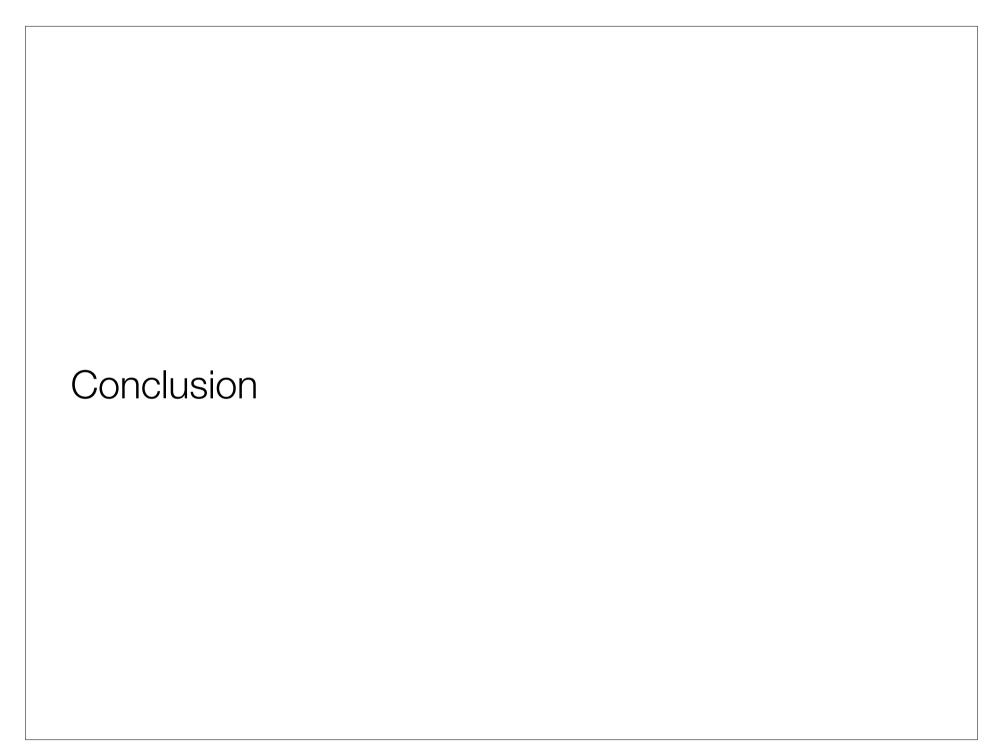
Safe Optimisations for Shared Memory Concurrent Programs

PLDI 2011

H. Bohem

Threads Cannot Be Implemented as a Library

PLDI 2005



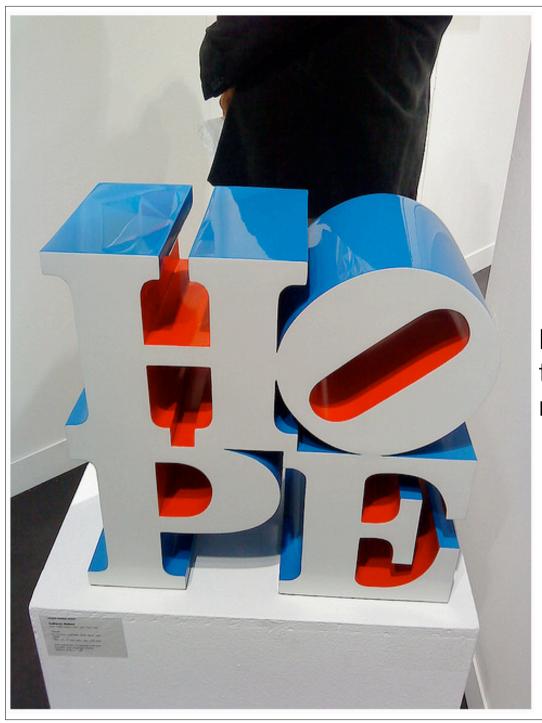
Syllabus

In these lectures we have covered the hardware models of two modern computer architectures (x86 and Power/ARM - at least for a large subset of their instruction set).

We have seen how compiler optimisations can also break concurrent programs and the importance of defining the memory model of highlevel programming languages.

We have also introduced some proof methods to reason about concurrency.

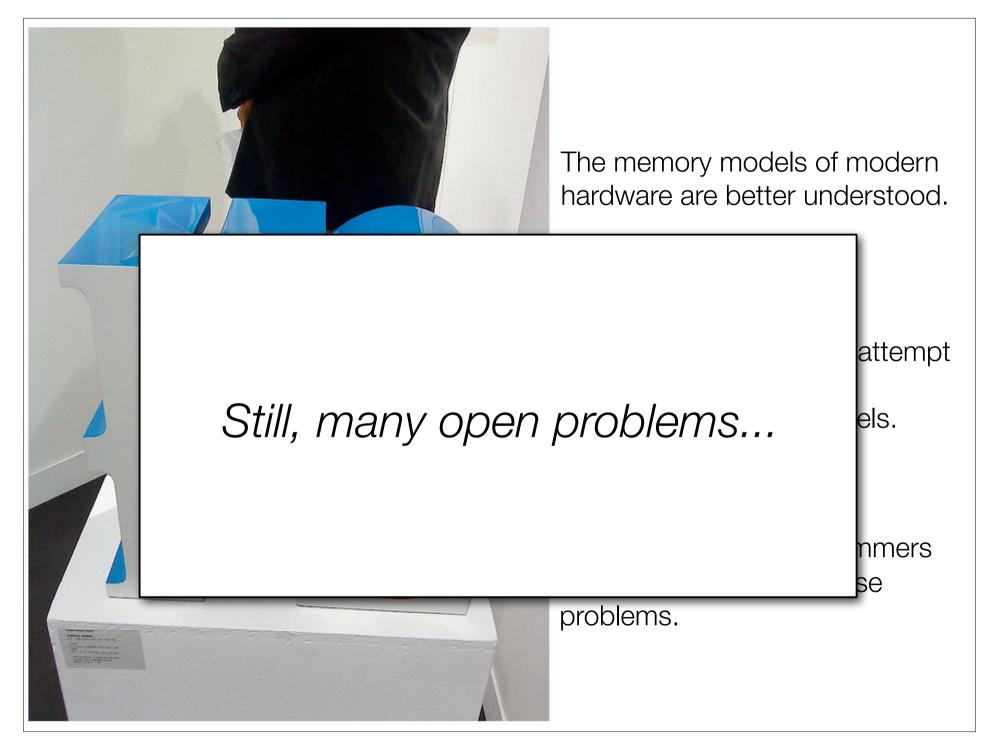
After these lectures, you might have the feeling that multicore programming is a mess and things can't just work.



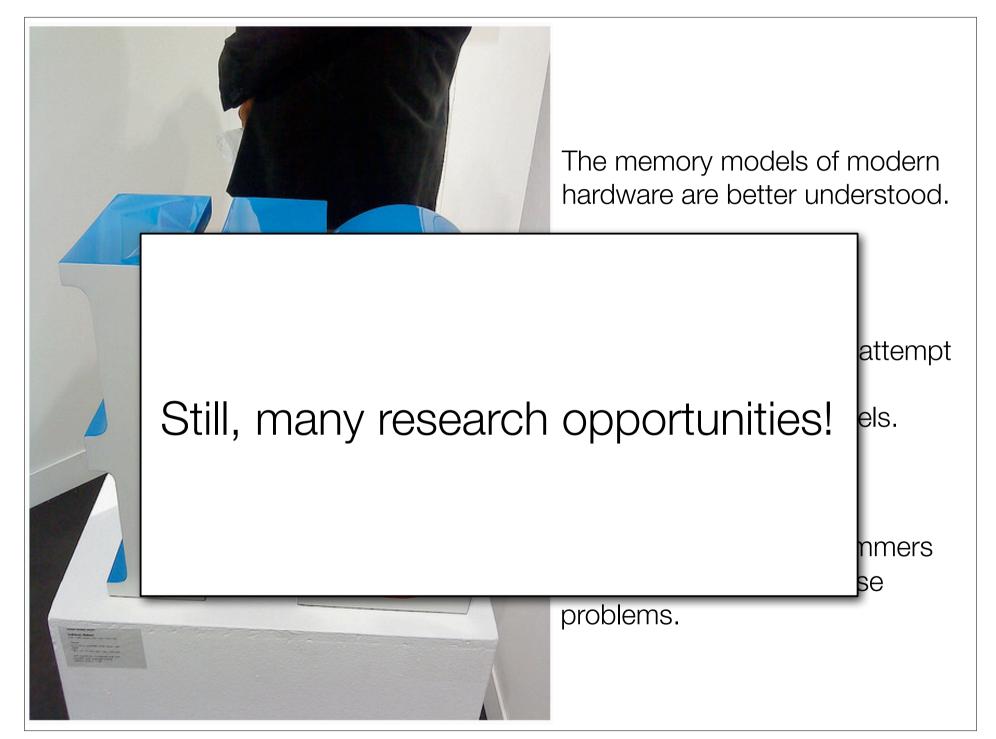
The memory models of modern hardware are better understood.

Programming languages attempt to specify and implement reasonable memory models.

Researchers and programmers are now interested in these problems.



120-2



120-3