Proving correctness of multithread algorithms

Asked 16 years, 3 months ago Modified 16 years, 3 months ago

Viewed 3k times

13

Multithread algorithms are notably hard to design/debug/prove. Dekker's algorithm is a prime example of how hard it can be to design a correct synchronized algorithm. Tanenbaum's Modern operating systems is filled with examples in its IPC section. Does anyone have a good reference (books, articles) for this? Thanks!

algorithm multithreading theory correctness proof

Share

Improve this question

Follow

edited Sep 17, 2008 at 22:54

asked Sep 16, 2008 at 16:44

Leandro

161 ● 2 ● 8

6 Answers

Sorted by: Highest score (default)

It is impossible to prove anything without building upon guarentees, so the first thing you want to do is to get familiar with the memory model of your target platform; Java and x86 both have solid and standardized memory models - I'm not so sure about CLR, but if all else fails, you'll have build upon the memory model of your target CPU architecture. The exception to this rule is if you intend to use a language that does does not allow any shared mutable state at all - I've heard Erlang is like that.

The first problem of concurrency is shared mutable state.

That can be fixed by:

- Making state immutable
- Not sharing state
- Guarding shared mutable state by the *same* lock (two different locks cannot guard the same piece of state, unless you *always* use exactly these two locks)

The second problem of concurrency is safe publication. How do you make data available to other threads? How do you perform a hand-over? You'll the solution to this problem in the memory model, and (hopefully) in the API. Java, for instance, has many ways to publish state and the java.util.concurrent package contains tools specifically designed to handle inter-thread communication.

The third (and harder) problem of concurrency is locking. Mismanaged lock-ordering is the source of dead-locks.

You can analytically prove, building upon the memory model guarentees, whether or not dead-locks are possible in your code. However, you need to design and write your code with that in mind, otherwise the complexity of the code can quickly render such an analysis impossible to perform in practice.

Then, once you have, or before you do, prove the correct use of concurrency, you will have to prove single-threaded correctness. The set of bugs that can occur in a concurrent code base is equal to the set of single-threaded program bugs, plus all the possible concurrency bugs.

Share Improve this answer

Follow

answered Sep 16, 2008 at 17:53

Chris Vest

8,672 ● 3 ● 38 ● 43

4

[The Pi-Calculus, A Theory of Mobile Processes](#) is a good place to begin.

Share Improve this answer

Follow

answered Sep 17, 2008 at 22:35

Apocalisp

35k ● 8 ● 108 ● 158

This is the original paper its based on: era.ed.ac.uk/bitstream/handle/1842/6569/Sangiorgi1993.pdf . I don't normally say this as I really appreciate the lineage of computers and how suprisingly relevant so many older texts

are but this one is a bit too abstract (phd thesis), didn't age well / difficult to understand today (speaks from a lisp-inspired reasoning of computer logic and functioning), and tries to generalize everything. So, I'd advise seeking another source. – Jack G Jul 12 at 17:46

4

"Principles of Concurrent and Distributed Programming", M. Ben-Ari
ISBN-13: 978-0-321-31283-9
They have in on safari books online for reading:
http://my.safaribooksonline.com/9780321312839

Share Improve this answer

Follow

answered Sep 18, 2008 at 15:09

jdkoftinoff
2,421 ● 1 ● 18 ● 18

That was pretty much exactly what I was looking for. Thanks! – Leandro Sep 20, 2008 at 19:23

@Leandro: If it was "pretty much exactly" what you were looking for, then you can 'accept' the answer. – Chris Vest Sep 26, 2008 at 22:26

2

Short answer: it's hard.

There was some really good work in the DEC SRC Modula-3 and larch stuff from the late 1980's.

e.g.

- Thread synchronization: A formal specification (1991) by A D Birrell, J V Guttag, J J Horning, R Levin
  System Programming with Modula-3, chapter 5
- Extended static checking (1998) by David L. Detlefs, David L. Detlefs, K. Rustan, K. Rustan, M. Leino, M. Leino, Greg Nelson, Greg Nelson, James B. Saxe, James B. Saxe

Some of the good ideas from Modula-3 are making it into the Java world, e.g. JML, though "JML is currently limited to sequential specification" says the intro.

Share Improve this answer Follow

answered Sep 16, 2008 at 19:08

dckc
61 • 4

The gatekeeper.dec.com link is broken, but the DEC SRC report on Extended static checking is available at research.microsoft.com/en-us/um/people/leino/papers/src-159.pdf . I agree ESC was really good work.
– Daira-Emma Hopwood May 26, 2012 at 18:21

2

I don't have any concrete references, but you might want to look into the Owicki-Gries theory (if you like theorem proving) or process theory/algebra (for which there are also various model-checking tools available).

Share Improve this answer

Follow

answered Sep 17, 2008 at 22:31

mweerden

14k ● 5 ● 34 ● 32

0

@Just in case: I is. But from what i learnt, doing so for a non trivial algorithm is a major pain. I leave that sort of a thing for brainier people. I learnt what i know from Parallel Program Design: A Foundation (1988) by K M Chandy, J Misra

Share Improve this answer

Follow

answered Sep 16, 2008 at 17:38

Mostlyharmless

2,283 ● 19 ● 28