Self-Verifying Dining Philosophers*

Peter Welch and Neil Brown
School of Computing, University of Kent, UK

IFIP WG2.4, Cape May, 30th. September, 2011

* About 10 of them *

The story of The Dining Philosophers is due to Edsger Dijkstra – one of the founding fathers of Computer Science.

It illustrates a classic problem in concurrency: how to share resources safely between competing consumers.

http://www.cs.utexas.edu/users/EWD/ewd03xx/EWD310.PDF

A new, really really neat, solution (Neil Brown / PHW)

A new, really neat, solution (Neil Brown / PHW)

A new, really neat, solution (Neil Brown / PHW)

A new, really neat, solution (Neil Brown / PHW)
null
The previous model check verifies properties of a college with precisely 5 philosophers. The FDR2 model check is almost instant.

Scaling to 10 philosophers puts a strain on my laptop – it gets very hot and takes a few minutes. Scaling to 20 fails.

In the FDR2 manual, Bill Roscoe explains how to verify a college with 10^20 philosophers … we had better follow his guidelines … and tackle the black art of compression in model checking …

With our simpler college, we want to beat that scale! Further, we would like to verify a college of any number of philosophers … using induction.

The first guideline is not to build the philosophers and forks as separate sub-systems and, then, the college as their parallel combination. This is what we did and it doesn’t let us use inductive reasoning very easily.

Instead, first build a philosopher-fork pair. Next, build chains of philosopher-fork pairs using recursion (e.g. a chain of length n is a chain of length (n-1) plus one more pair). Using induction, verify properties of the chain, for any n.

Finally, add one more pair that connects both ends of a chain and get the college. Verify the college using the already verified properties of the chain.

There are two further points that are needed: hiding and compression.

First, note that the thinking and eating reports from the philosophers play no role in the deadlock/livelock properties of the college. Each philosopher engages on its own thinking and eating channels with the environment of the college. The forks do not engage with those channels.

Therefore, no thinking or eating report can block the operations of the college. Verifying deadlock and livelock freedom in a college with the thinking and eating events hidden will also verify the result for a college that doesn’t hide them.
We can ask for the size of the labelled transition system (state machine) generated by FDR … We can ask for the size of the labelled transition system (state machine) generated by FDR …

We can ask for the size of the labelled transition system (state machine) generated by FDR …

We can ask for the size of the labelled transition system (state machine) generated by FDR …

We can ask for the size of the labelled transition system (state machine) generated by FDR …

We can ask for the size of the labelled transition system (state machine) generated by FDR …
We have to declare lots of events in the CSP code? Now we are OK! ☺☺☺

But it really should not be up to us to declare and use this infinite set of names that cannot be expressed in the CSP code? Not doing so seems to break the semantics of hiding … ???

We have to declare lots of events that are not possible in the CSP code …

Now build a chain … using recursion

An event called eatBarMiddle inside the instance of the CSP code gets confused with the eatBarMiddle event in the CSP code.

Generating the FDR code for this requires an extra care … (because FDR2 does something it shouldn’t – clairvoyance)

The following does not work correctly,…
How similar are they and might they deadlock?

\[ \text{VERIFY \ PROC\ Chain\ (VAL\ VERIFY\ INF\ size)} \]
\[ \text{VAL\ VERIFY\ INF\ length;} \]
\[ \text{CHAIN\ (55, eatBarRight, eatBarLeft)} \]
\[ \text{CHAIN\ (44, eatBarRight, eatBarLeft)} \]

ChainChain ((55, eatBarRight, eatBarLeft))

ChainChain ((44, eatBarRight, eatBarLeft))

Let \(M(0)\) be the hypothesis that:

\[ \text{Chain\ (44, eatBarRight, eatBarLeft)} \]

Clearly \(M(4)\) and, by model checking, \(M(5)\).

\[ \text{H(4)} \]

\[ \text{H(5)} \]

\[ i \geq 4 \]

\[ H(i+1) \]

\[ \text{EQUIVALENT.F} \]

\[ \text{EQUIVALENT.F} \]

We have \(M(4)\) and \(M(5)\). Suppose \(M(4)\) for any \(i \geq 4\). Consider:

\[ \text{CHAIN\ (55, eatBarRight, eatBarLeft)} \]

This reduces to:

\[ \text{BARRIER\ eatBarMiddle} \]

\[ \text{PhiFork\ (eatBarMiddle, eatBarLeft)} \]

By \(M(4)\), this is \(\text{EQUIVALENT.F}\) to:

\[ \text{CHAIN\ (44, eatBarRight, eatBarLeft)} \]

Which, by \(M(5)\), is \(\text{EQUIVALENT.F}\) to:

\[ \text{CHAIN\ (44, eatBarRight, eatBarMiddle)} \]

But what about Colleges?

All chains of (non-reporting) \(\text{philosopher-fork}\) pairs with lengths equal to or greater than 4 are failures equivalent. Further, all such chains are deadlock-free (since model checking gave us that directly for chains of lengths 1 through 4).

But what about Colleges?

We can immediately deduce that all \(\text{CollegeChain}\) with size equal to or greater than 5 are failures equivalent (since their \(\text{Chain}\) sub-components have lengths equal to or greater than 4 and are failures equivalent).

Hence, all \(\text{CollegeChain}\) with size equal to or greater than 2 are deadlock-free. Of course, with no reporting, they are hopelessly livelocked!

So what about reporting Colleges?

An earlier argument showed that a deadlock-free result for a college with external reports hidden implies a deadlock-free result for a college with external reports (since the external reporting cannot cause internal blocking). So all reporting colleges of any size are deadlock-free.

The following argument shows that a college with external reports is also livelock-free.

From simple code inspection, a \(\text{Phil}\) process cannot engage in two \(\text{eatBar}\) events (internal) without an (external) intervening report.

This could be model-checked, using techniques discussed earlier, if it was felt necessary!
Finally, the Brute Force Approach

--- A chain of (length^level) philosopher-fork pairs.
VERIFY PROC Chain2 (VAL VERIFY INF level, length)!
PAR id = 1 FOR length -- 2
NORMALISE (level -- 1, length, eatBar\[id -- 1\], eatBar\[id\])
VERIFY DEADLOCK.FREE.F Chain2 (1000, 10, _)
VERIFY Deadlock.FREE.F Chain2 (010, 10, _)

Finally, the Brute Force Approach

--- A chain of (length^level) philosopher-fork pairs.
VERIFY PROC Chain2 (VAL VERIFY INF level, length)!
PAR id = 1 FOR length -- 2
NORMALISE (level -- 1, length, eatBar\[id -- 1\], eatBar\[id\])
VERIFY DEADLOCK.FREE.F Chain2 (2500, 10, _)
VERIFY Deadlock.FREE.F Chain2 (500, 10, _)

Finally, the Brute Force Approach

--- A chain of (length^level) philosopher-fork pairs.
VERIFY PROC Chain2 (VAL VERIFY INF level, length)!
PAR id = 1 FOR length -- 2
NORMALISE (level -- 1, length, eatBar\[id -- 1\], eatBar\[id\])
VERIFY DEADLOCK.FREE.F Chain2 (2500, 10, _)
VERIFY Deadlock.FREE.F Chain2 (500, 10, _)

Summary

A new, simple and symmetric solution to the Dining Philosophers has been presented.

That it is deadlock and livelock free has been verified ... trivially for up to 8 philosophers (just push a button).
Verifying this for 10^2000 philosophers required some creative refactoring of the program code ... but then only took around 10 seconds on my new laptop, ☺. Rather large systems can be verified (but we were lucky here).
Verifying this for any number of philosophers required very similar (and simpler) refactoring of the program code and a simple induction argument. The heavy-lifting verification of the base and induction steps took no noticeable time.
And all we did was program ... ☺ ☺ ☺
A dynamic concurrency model built into its core design with full denotational semantics (based on the CSP traces/failures/divergences model).

No data race hazards (eliminated by compiler aliasing analysis).

Deterministic concurrency by default. Non-determinism is introduced only by explicit use of special features (e.g., choice, shared channels).

The fastest and most effective multicore scheduler on the planet (maybe).

Program verification by programming (and a little thinking).

Simple to learn, simple to use (e.g., 90 min Lego Robots occam workshop).

A tiny user base ... to be fixed (???) ... how (???) ... when (???) ...

But occam-π needs rationalising.

It's time, again, for Occam's razor.