And the truth will make you spin

Mathew, Cherry G.
cherry@NetBSD.org

AsiaBSDcon 2024
Taipei
Taiwan
March 24, 2024

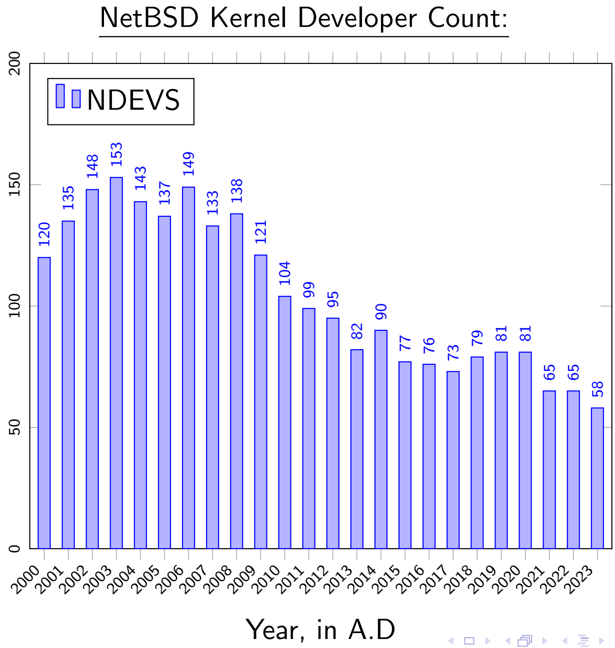
Design Driven Development using the spin verifier.

Audience:

A Software practitioner:

- Dealing with concurrent execution and distributed state. Eg: OS developers.
- Who finds current software system design approaches inadequate.
- For whom descriptive documentation is irksome and inadequate.
- Deal with design issues (for eg: as an "architect")
- Deal with implementation issues (for eg: as an "engineer")

Motivations:



Knowledge Management:

- Problem:

Knowledge Management:

- Problem:
  - Design crowdsourcing not viable

Knowledge Management:

- Problem:
  - Design crowdsourcing not viable
- Proposed Solution:

Knowledge Management:

- Problem:
  - Design crowdsourcing not viable
- Proposed Solution:
  - Formal Specification

Knowledge Management:

- Problem:
  - Design crowdsourcing not viable
    - Multiple design opinions about the same code.
    - Documentation/code can drift.
    - Greybeard memory can fade.
    - Unit Testing can only probe points in design space.
- Proposed Solution:
  - Formal Specification
    - Automated verification by model checking.
    - Invariants serve as design **Canon**.

Formal Specification: a trivial example

Consider the following C code:

```
#include <stdio.h>
#include <assert.h>

int j, i, array[10];

void
printarray(void)
{
        for (j = 0; j < 10; j++) {
                i = j;
                printf("array[%d] == %d\n", i, array[i]);
        }
}
```

Formal Specification: a trivial example

Questions such as:

- Why 10 elements, and not 9 or 11 or 1000 ?
- Where is the number of elements specified ?
- What are the edge cases for i and j ?

Formal Specification: a trivial spin example

Specification:

```
#define ARRAYSIZE ARRAYMAX

int j, i, array[ARRAYSIZE];

active proctype printarray()
{
  for (j : 0 .. (ARRAYSIZE - 1)) {
    i = j;
    printf("array[d] == %d\n", i, array[i]);
  }
}
```

Formal Specification: a trivial spin example

Specification State:

```
int j, i, array[ARRAYSIZE];
```

Formal Specification: a trivial spin example

Specification Model:

```
active proctype printarray()
{
  for (j : 0 .. (ARRAYSIZE - 1)) {
    i = j;
    printf("array[d] == %d\n", i, array[i]);
  }
}
```

Formal Specification: a trivial spin example

Specification Invariants:

```
/* Monitors the progress of state variables */
int j, i, array[ARRAYSIZE];
/* Written in "LTL" - Linear Temporal Logic */
ltl  /* Canon */
{
  true
  && (always (ARRAYSIZE == ARRAYMAX))
  && (always ((i >= 0) && i <= (ARRAYMAX - 1)))
  && (eventually always (i == (ARRAYMAX - 1)))
}
```

(D-Cubed) Design Driven Development:

- Inspired from Test Driven Development

(D-Cubed) Design Driven Development:

- Inspired from Test Driven Development
- Back to the “Drawing board”

(D-Cubed) Design Driven Development:

- Inspired from Test Driven Development
- Back to the "Drawing board"
- Paradigm shift from: "start digging" $\Rightarrow$ "start designing"

(D-Cubed) Design Driven Development:

- Inspired from Test Driven Development
- Back to the "Drawing board"
- Paradigm shift from: "start digging" $\Rightarrow$ "start designing"
- "Drawing board" is formal design

(D-Cubed) Design Driven Development:

- Inspired from Test Driven Development
- Back to the “Drawing board”
- Paradigm shift from: “start digging” $\Rightarrow$ “start designing”
- ”Drawing board” is formal design
- Verification/consistency of designs can be automated.

(D-Cubed) - process:

- Define scope - "Hub" as unit of design scope.

(D-Cubed) - process:

- Define scope - "Hub" as unit of design scope.
- Build Formal Specification. (Spin is useful on NetBSD)

(D-Cubed) - process:

- Define scope - "Hub" as unit of design scope.
- Build Formal Specification. (Spin is useful on NetBSD)
  - Model state space and transition logic.
  - Write invariants/properties for the state space.
  - Consistency checking/verification.

(D-Cubed) - process:

- Define scope - "Hub" as unit of design scope.
- Build Formal Specification. (Spin is useful on NetBSD)
  - Model state space and transition logic.
  - Write invariants/properties for the state space.
  - Consistency checking/verification.
- Implement model. (C is used on NetBSD)

(D-Cubed) - process:

- Define scope - "Hub" as unit of design scope.
- Build Formal Specification. (Spin is useful on NetBSD)
  - Model state space and transition logic.
  - Write invariants/properties for the state space.
  - Consistency checking/verification.
- Implement model. (C is used on NetBSD)
- Extract the model from Implementation (Modex/spin)

(D-Cubed) - process:

- Define scope - "Hub" as unit of design scope.
- Build Formal Specification. (Spin is useful on NetBSD)
  - Model state space and transition logic.
  - Write invariants/properties for the state space.
  - Consistency checking/verification.
- Implement model. (C is used on NetBSD)
- Extract the model from Implementation (Modex/spin)
- Fidelity checking

(D-Cubed) - process:

- Define scope - "Hub" as unit of design scope.
- Build Formal Specification. (Spin is useful on NetBSD)
  - Model state space and transition logic.
  - Write invariants/properties for the state space.
  - Consistency checking/verification.
- Implement model. (C is used on NetBSD)
- Extract the model from Implementation (Modex/spin)
- Fidelity checking
- Iterate

(D-Cubed) - case study - Adaptive Replacement Cache

"ARC: A SELF-TUNING, LOW OVERHEAD REPLACEMENT CACHE" by Megiddo et. al.
https://www.usenix.org/legacy/events/fast03/tech/full_papers/megiddo/megiddo.pdf

ARC(c)

INPUT: The request stream $x_1, x_2, \ldots, x_t, \ldots$.
INITIALIZATION: Set $p = 0$ and set the LRU lists T_1, B_1, T_2, and B_2 to empty.

For every $t \geq 1$ and any x_t, one and only one of the following four cases must occur.

Case I: x_t is in T_1 or T_2. A cache hit has occurred in ARC(c) and DBL($2c$).
Move x_t to MRU position in T_2.

Case II: x_t is in B_1. A cache miss (resp. hit) has occurred in ARC(c) (resp. DBL($2c$)).

$$\boxed{\text{ADAPTATION:}}\ \text{Update } p = \min\{p + \delta_1, c\} \text{ where } \delta_1 = \begin{cases} 1 & \text{if } |B_1| \geq |B_2| \\ |B_2|/|B_1| & \text{otherwise.} \end{cases}$$

REPLACE(x_t, p). Move x_t from B_1 to the MRU position in T_2 (also fetch x_t to the cache).

Case III: x_t is in B_2. A cache miss (resp. hit) has occurred in ARC(c) (resp. DBL($2c$)).

$$\boxed{\text{ADAPTATION:}}\ \text{Update } p = \max\{p - \delta_2, 0\} \text{ where } \delta_2 = \begin{cases} 1 & \text{if } |B_2| \geq |B_1| \\ |B_1|/|B_2| & \text{otherwise.} \end{cases}$$

REPLACE(x_t, p). Move x_t from B_2 to the MRU position in T_2 (also fetch x_t to the cache).

Case IV: x_t is not in $T_1 \cup B_1 \cup T_2 \cup B_2$. A cache miss has occurred in ARC(c) and DBL($2c$).

Case A: $L_1 = T_1 \cup B_1$ has exactly c pages.
    If ($|T_1| < c$)
        Delete LRU page in B_1. REPLACE(x_t, p).
    else
        Here B_1 is empty. Delete LRU page in T_1 (also remove it from the cache).
    endif
Case B: $L_1 = T_1 \cup B_1$ has less than c pages.
    If ($|T_1| + |T_2| + |B_1| + |B_2| \geq c$)
        Delete LRU page in B_2, if ($|T_1| + |T_2| + |B_1| + |B_2| = 2c$).
        REPLACE(x_t, p).
    endif
Finally, fetch x_t to the cache and move it to MRU position in T_1.

Subroutine REPLACE(x_t, p)
If (($|T_1|$ is not empty) and (($|T_1|$ exceeds the target p) or (x_t is in B_2 and $|T_1| = p$)))
    Delete the LRU page in T_1 (also remove it from the cache), and move it to MRU position in B_1.
else
    Delete the LRU page in T_2 (also remove it from the cache), and move it to MRU position in B_2.
endif

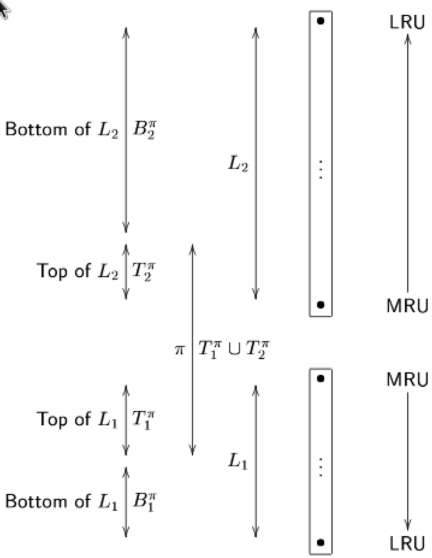
(D-Cubed) - case study - Adaptive Replacement Cache

mail to tech-kern@ and code listing:

`https://mail-index.netbsd.org/tech-kern/2023/09/28/msg029203.html`

(D-Cubed) - case study - Adaptive Replacement Cache

ARC Caches and state variables.



State Variables

- Buffers T1 U B1 == L1
- Buffers T2 U B2 == L2
- variable p - "Tunable Parameter"
- C - half the size of the Cache

(D-Cubed) - case study - Adaptive Replacement Cache

Specification Invariants:

```
ltl
{
        /* c.f Section I. B, on page 3 of paper */
        always ((lengthof(T1) +
                 lengthof(B1) +
                 lengthof(T2) +
                 lengthof(B2)) <= (2 * C))

        /* Reading together Section III. A., on page
             7, and
         * Section III. B., on pages 7,8
         */
        && always ((lengthof(T1) + lengthof(B1)) <=
              C)
        && always ((lengthof(T2) + lengthof(B2)) <=
              (2 * C))

        /* Section III. B, Remark III.1 */
        && always ((lengthof(T1) + lengthof(T2)) <=
              C)

        /* TODO: III B, A.1 */

        /* III B, A.2 */
        && always (((lengthof(T1) +
                  lengthof(B1) +
                  lengthof(T2) +
                  lengthof(B2)) < C)
                 implies ((lengthof(B1) == 0) &&
                          lengthof(B2) == 0))
```

```
        /* III B, A.3 */
        && always (((lengthof(T1) +
                  lengthof(B1) +
                  lengthof(T2) +
                  lengthof(B2)) >= C)
                 implies ((lengthof(T1) +
                          lengthof(T2)) == C))

        /* TODO: III B, A.4 */

        /* TODO: III B, A.5 */

        /* IV A. */
        && always (p <= C)

        /*
         * Force spin to generate a "good" input
              trace (See: arc.drv)
         * The handwavy reasoning here is that an
              absolutely full ARC
         * would have had to exercise all codepaths
              to get there.
         */
        && always !(true /* Syntactic glue */
                && lengthof(T1) == C
                && lengthof(B1) == C
                && lengthof(T2) == C
                && lengthof(B2) == C
              )
}
```

(D-Cubed) - case study - Adaptive Replacement Cache

Specification Invariants:

On LTL:

- assert() checks for current status of variable ***NOW***.
- LTL checks along the entire life of the state machine.

(D-Cubed) - case study - Adaptive Replacement Cache

Specification Invariants:

"Propositional Logic".
for example:

```
int x;

...

void
test(void)
{
    assert(x == SOMEVALUE);
}

/*
 * Implies x should be that value at that
 * specific execution point.
 */
```

(D-Cubed) - case study - Adaptive Replacement Cache

Specification Invariants:

LTL - or Linear Temporal Logic
for example:

```
int x;

...

ltl
{
    always (x == SOMEVALUE)
}

/*
 * Implies x should be that value throughout
 * execution.
 */
```

(D-Cubed) - Model Extraction

The spin companion "Model Extractor"(modex) can extract a model implicit within C code. This extraction is guided by a bespoke language "prx" which modex uses.

for example:

```
%F test.c
%X -n test

/*
 * Extract model from test.c:test()
 */
```

Fidelity Checking:

Does:

```
ltl
{
    always (x == SOMEVALUE)
}
```

Still pass ?

(D-Cubed) - Model Extraction

Model Extraction:

Extraction gives us a spin model file with the following content:

```
// Generated by MODEX Version 2.11 - 3 November 2017
// Sat 23 Mar 2024 10:38:18 PM IST from test.prx

int x;
proctype p_test( )
{
      c_code [(now.x==SOMEVALUE)] { ; };
}
```

We can now use a common driver to drive this “Hub” being checked.

```
init {
    pid n;
    n = run p_test();

    (n == _nr_pr); /* Wait for p_test() to exit */
}
```

(D-Cubed) - Model Driver

Spin as implementation driver:

- modex parser is flaky
- hook up spin to drive test() directly.

```
int x;
proctype p_test( )
{
    c_code {
    int x;
    x = now.x;
    test();
    }
}

...

$ spin -D SOMEVALUE=1 -a test.drv
$ cc -D SOMEVALUE=1 -o test pan.c test.c
$ ./test
```

(D-Cubed) - case study - Adaptive Replacement Cache

Specification Invariants:

Pros	**Cons**
- Explicit design visibility	- Dev time can be ~2.5x
- Debugging reduced by ~90%	- Model/Implementation sync overhead
- Can ask new falsifiable questions via LTL	- Poorly crafted LTL can blur design clarity
- Can integrate into CI	- poorly crafted constraints can stall CI

(D-Cubed) - differences with MBSE/Systems Modelling:

- Requirements are at the State Machine level
- No code generation
- Fidelity checking
- Integrated with CI

(D-Cubed) - TODO for Spin/Modex on NetBSD

- Modex is flaky - re-write parser for C99
- Harness needs (language) re-design

(D-Cubed) - first steps for NetBSD. (WIP)

- Alternative method, without Modex
  (because of broken C-lang parser).
- Existing NetBSD code:
  - spin as "driver" for "Rump"-ed C code.
  - standalone verification possible.
  - glue code instead of modex.
- Pro: Existing code can be dropin verified.
- Con: Extracted model replaced by glue code updating model state on behalf of C code. Verification blindspot.

(D-Cubed) - introducing “SpinOS”

- Capture design models of various “Hub”s in NetBSD
- Record Invariants as design documentation
- Comprehensive formal design of a real world OS
- Fidelity checking to keep model “grounded”
- Can be used as basis for D-Cubed based development in several OSs.
- Please join the project! (Send me email, for now).

(D-Cubed) Roadmap:

- Develop SpinOS as canonical model for NetBSD.
- Integrate SpinOS elements into NetBSD CI
- Auto-generate documentation (man pages for eg:) from LTL.
- RAG - Online Oracle for greybeard style Q&A

(D-Cubed) Questions ?:

Fediverse:

`@c@bow.st`

$\Leftarrow$

Scan QR Code for consulting.