WHAT'S THE FUZZ ABOUT TESTING?

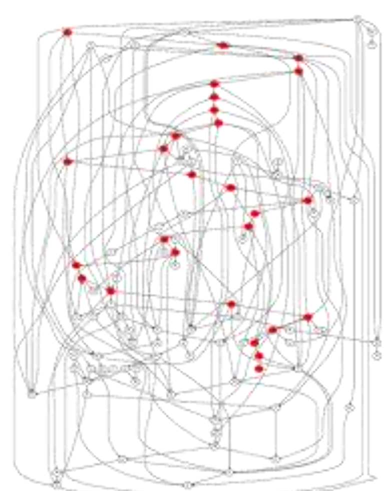
Gerard Holzmann
gh@jpl.nasa.gov

LARS
NASA/JPL Laboratory for Reliable Software

"Whatever can happen will happen if we make trials enough."
Augustus De Morgan (1866)

QUESTIONS

1. How good is Unit Testing with *100% MC/DC Coverage*?
2. Is *Fuzz testing* (Randomized Testing) better?
3. What if we use *Testing with Perfect Recall*?
4. How can we exploit *Parallelism*?

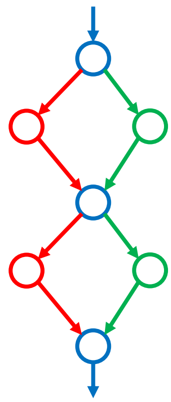


2

THREE SMALL EXAMPLES
1: CONDITIONALS

```
int *p;

void
fct(int x, int y)
{
   if (x)
   {     p = &x;
   }
   if (y)
   {    *p = y;
   }
}
```



```
void
test_main(void)
{
    fct(0,0);
    fct(1,1);
}
```

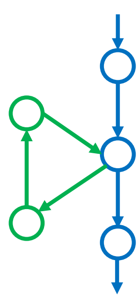
this test achieves 100% MC/DC coverage, yet it misses the bug that is revealed with another test: foo(0,1)
it covers just 50% of the execution paths in the control-flow graph

3

2: LOOPS

```
void
fct(int x, int y)
{   int i, a[4];

    for (i = 0; i < x+y; i++)
    {  a[i] = i;
    }
}
```



```
void
test_main(void)
{
    fct(1,1);
}
```

this test achieves 100% MC/DC coverage, yet it misses the array indexing bug that is revealed with, for instance, foo(1,3)
it covers just 1 of 2^{31} theoretically possible execution paths

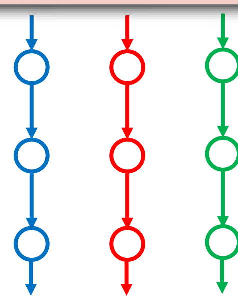
4

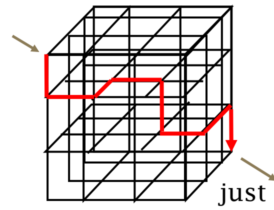
3: THREADS

So maybe MC/DC coverage is not the best metric to aim for.
Is Random Test Selection (Fuzz Testing) better?

```
int   x,  y,  r;
int  *p, *q, *z;
int **a;

thread_1() // initialize
{
    p = &x;
    q = &y;
    z = &r;
}
```





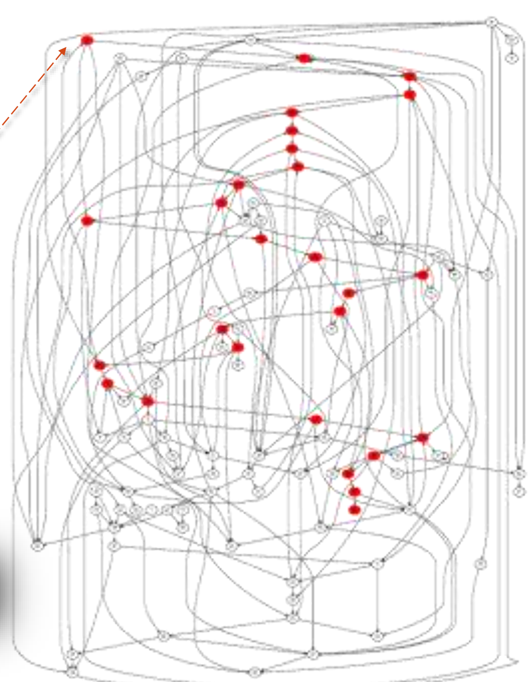

```
thread_2() // swap *p and *q
{
    r = *p;
   *p = *q;
   *q = r;
}
```

```
thread_3() // access z via a and p
{
      a = &p;
     *a = z;
    **a = 12;
}
```

5

AN EXAMPLE

- 83 nodes are reachable from the node labeled *S1*
- How many *random tests* would we have to do to be sure that all 83 reachable nodes are visited at least once?

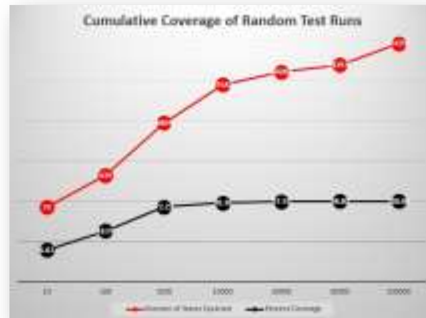


a sample graph with 100 nodes and 200 edges

6

N RANDOM TESTS OF 500 STEPS
STATES VISITED VS # UNIQUE STATES

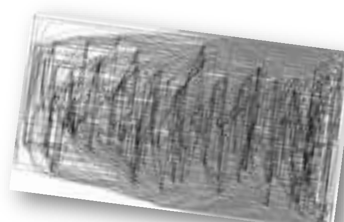
nr of tests	visited states	unique states	percent coverage	time
10	70	5	6%	1 second
100	439	15	18%	3 seconds
1,000	8,804	60	72%	1 minute
10,000	79,582	75	90%	6 minutes
20,000	166,066	81	97%	12 minutes
30,000	243,978	82	99%	17 minutes
100,000	834,707	83	100%	52 minutes



(the x-axis is a logscale)

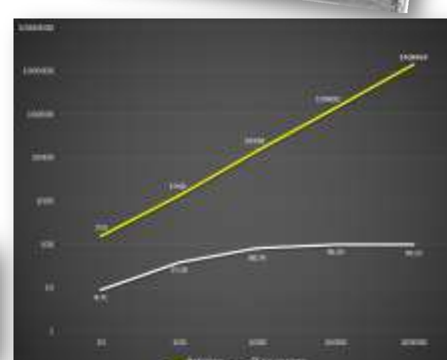
7

THE SAME TEST FOR A GRAPH OF 1000 NODES, 781 REACHABLE



nr of tests	visited states	unique states	percent coverage	time (sec)
10	153	68	9%	1
100	1,340	291	37%	6
1,000	14,338	631	81%	124
10,000	139,692	754	96%	640
100,000	1,408,469	775	99%	93120

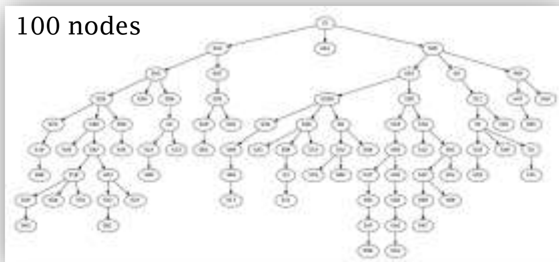
random test is unbiased, but does increasing amounts of duplicate work as it progresses, making it hard to push coverage to 100%

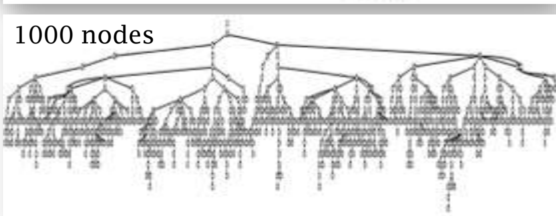


⟶ nr of random tests

8

WHAT IF YOU COULD REMEMBER WHERE YOU'VE BEEN: WITH DFS OR BFS





nr of tests	visited states	unique states	percent coverage
1	83	83	100%

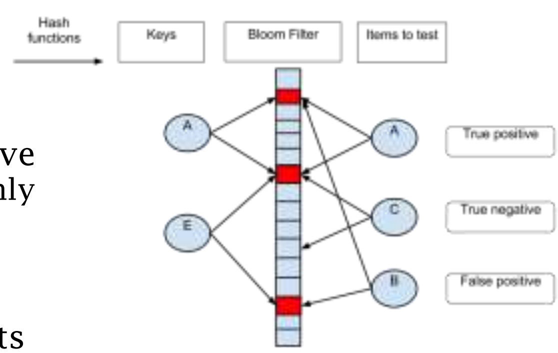
nr of tests	visited states	unique states	percent coverage
1	781	781	100%

a breadth-first search (BFS) with *Perfect Recall* in either graph visits *all* reachable nodes and explores *all* execution paths *without duplication* in seconds

9

DOES THE RECALL HAVE TO BE PERFECT?

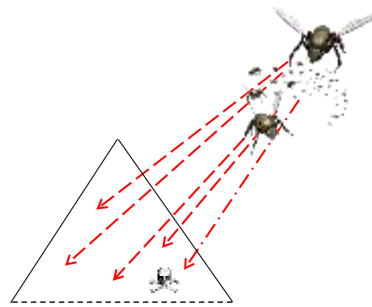
- What if storing all reachable states (for ***perfect*** recall) takes too much memory?
- That's Okay: Recall does not have to be completely Perfect: it is only meant to *reduce* the amount of duplicate work
- It often suffices to store just a hash-signature (or just a few bits using a fixed size Bloom filter)



10

CAN IT BE FAST TOO?

- For large problems, a full DFS or BFS search could require excessive amounts of *time*
- But, we can *parallelize* the tests if we can split up the search space, using … randomization
  - *leading to a technique for Swarm Testing*



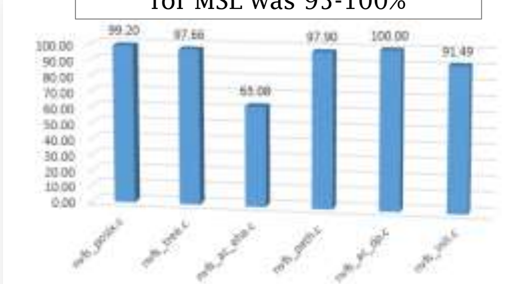
method:

(1) use N parallel search engines (hundreds, thousands)
(2) define a small memory bound M for each search
(3) randomize the DFS within each search engine

AN EXPERIMENT

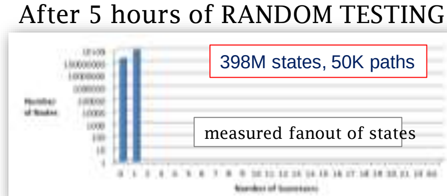
NVFS UNIT TEST SUITE

The arget Statement Coverage (%) for MSL was 95-100%

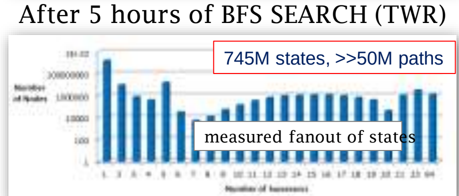


We measured the number of unique system states reached in all the above NVFS unit tests combined as
35,796 unique states (plus 1,175 duplicates)
and an estimated number of 100 distinct execution paths

After 5 hours of RANDOM TESTING



After 5 hours of BFS SEARCH (TWR)



Despite ~98% statement coverage, the Unit Tests explored **3 orders of magnitude** fewer states than either Random or TWR.
Testing with Recall (TWR) performed best.

SO WHAT IS THE FUZZ?

- There's one downside: to use *Testing with Recall* the application must be instrument-ed so its *state* can be captured and remembered
- But if you do this you can:
  - dramatically increase test coverage
  - and, you can also perform much stronger checks, up to full linear temporal logic model checking of source code

13

THANK YOU

"A random element is rather useful when we are searching for a solution of some problem."

A.M. Turing, "Computing machinery and intelligence," Oxford University Press, MIND (the Journal of the Mind Association), Vol. LIX, no. 236, pp. 433-60, (1950).

LARS NASA/JPL Laboratory for Reliable Software

14

MC/DC TESTING

Modified condition/decision coverage – Every point of entry and exit in the program has been invoked at least once, every condition in a decision in the program has taken all possible outcomes at least once, every decision in the program has taken all possible outcomes at least once, and each condition in a decision has been shown to independently affect that decision's outcome. A condition is shown to independently affect a decision's outcome by: (1) varying just that condition while holding fixed all other possible conditions, or (2) varying just that condition while holding fixed all other possible conditions that could affect the outcome.