Never Trust Any Published Algorithm

Mark Gritter
@markgritter
Tintri, Inc.
Minnebar 2015
April 11, 2015

Why should you care?

- New algorithms are invented all the time.
- Correct implementation of algorithms matters a lot.
- The techniques which find bugs in algorithms are widely applicable.

An algorithm for analyzing log message types

minnesota + tech + design + drinks

What’s the idea?

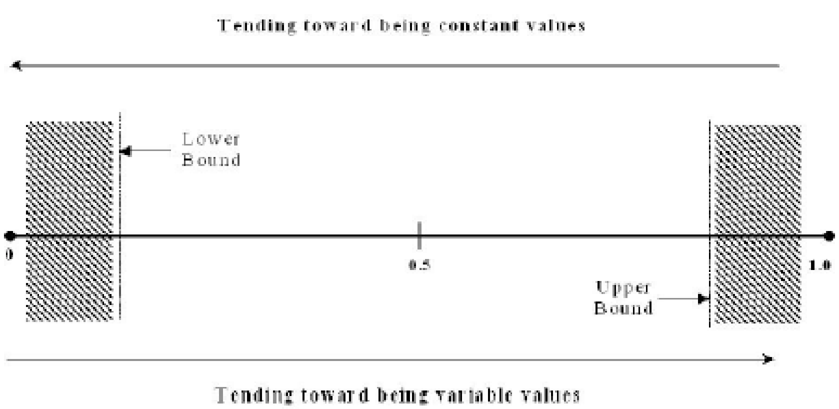
- Take a set of log output
- Heuristically partition them into messages of the same type
- Recover the original “definition” of the event

```
Device eth0 link down.
Device eth1 link up 100 Mbps.
Device eth1 link up 1 Gbps.
Device eth0 high error rate 2324/s.
```

```
log( LOG_INFO, “Device %s link down.\n” );
log( LOG_INFO, “Device %s link up %d %s. );
log( LOG_ERROR, “Device %s high error rate %d/s” );
```

One of the Heuristics

- Looking at two columns and trying to decide which is the variable.
- What's wrong with this?



Algorithm 4 Get_Rank_Position Function

Input: Set S of token values from the M side of a $1 - M$ or $M - 1$ mapping of a log file partition.
Real number *lower_bound*.
Real number *upper_bound*.
Output: Integer *split_rank*. *split_rank* can have values of either 1 or 2.

```
Distance = Cardinality of S / #Lines_that_match_S
if Distance ≤ lower_bound then
  if Mapping is 1-M then
    split_rank = 2
  else
    split_rank = 1 {Mapping is M-1}
  end if
else if Distance ≥ upper_bound then
  if Mapping is 1-M then
    split_rank = 1
  else
    split_rank = 2 {Mapping is M-1}
  end if
else
  if Mapping is 1-M then
    split_rank = 1
  else
    split_rank = 2 {Mapping is M-1}
  end if
end if
Return(split_rank)
```

How did this happen?

- This algorithm was likely only implemented once, by a single person.

- Reviewers and shepherds are busy, have their own priorities, and certainly don’t try to write their own code.
- Maybe the code used in the experiments described does something sensible or maybe it doesn’t--- 404 when I tried to download.
- Unfortunately this is typical of many published algorithms.

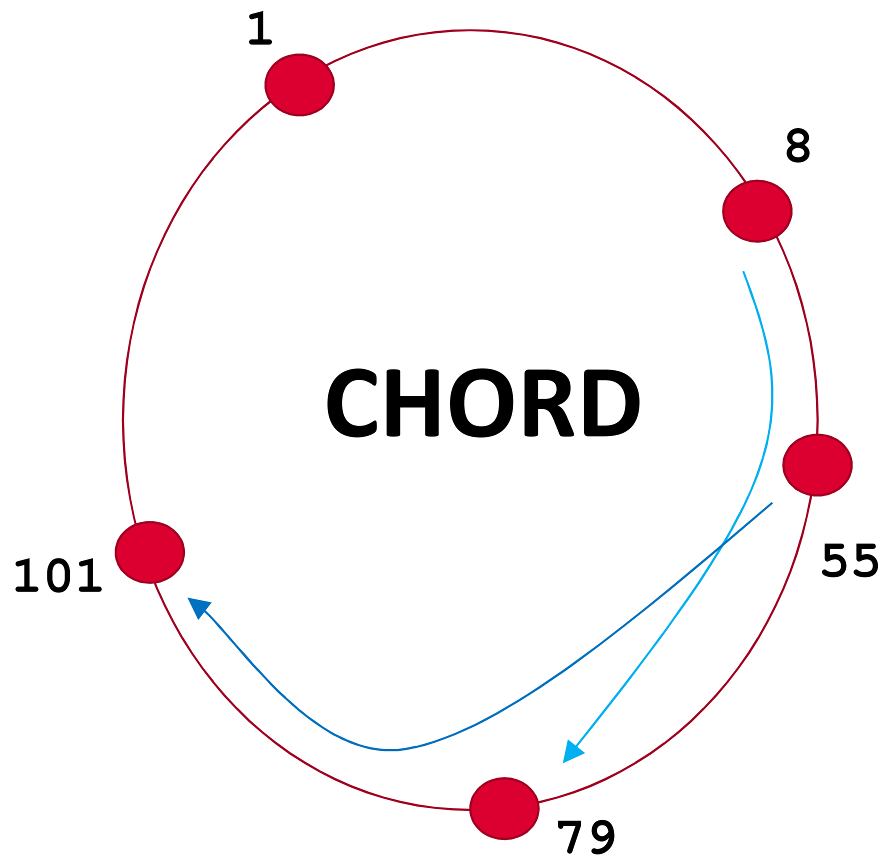
Lesson for development

- If only one person understands the algorithm, it probably doesn't work correctly.
- Have two different people prototype your core algorithm!
  - Most of the effort is in release hardening, maintenance, and testing anyway.

An algorithm for distributed hash tables

What's the idea?

- Use peer-to-peer technology to build an ad-hoc, scalable, robust key-value store.
  - Nodes can join, fail, etc., but properties of the network remain invariant.
- Award-winning paper with proof of correctness, implemented and studied many times (at least 10!)
  - Originally published in 2001
  - 2011 SIGCOMM Test-of-Time Award



What breaks?

- Nearly everything! *None* of the seven claimed invariants actually hold.
- Pamela Zave, “Using Lightweight Modeling to Understand Chord”, SIGCOMM Computer Communication Review, April 2012

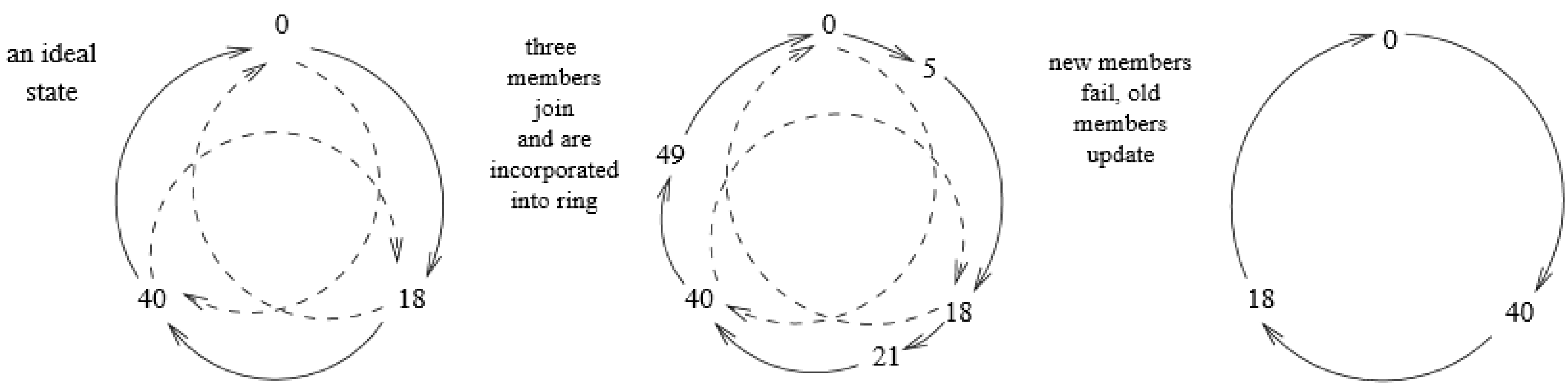


Figure 7: Three stages (left to right) creating a counterexample to *OrderedRing*.

How was the problem found?

- Dr. Zave used Alloy, a tool for model checking
  - First have to translate a loose description into a formal model
  - Exactly specify the invariants
  - Then the tool looks for counterexamples

```
pred OneOrderedRing [t: Time] {
let ringMembers =
{ n: Node | n in n.(^(bestSucc.t)) } |
    some ringMembers                            -- at least one ring
&& (all disj n1, n2: ringMembers |
      n1 in n2.(^(bestSucc.t)) )                -- not two
&& (all disj n1, n2, n3: ringMembers |
      n2 = n1.bestSucc.t => ! Between[n1,n3,n2]
                                                -- ring is globally ordered
)
```

How did this happen?

- Informal reasoning about failure conditions.
  - "Distributed systems frequently do not work the way we expect them to."
  - Easier to focus on examples you know work rather than finding examples which don't work.
- None of the implementations concretely defined which version of the algorithm they were implementing. If they found problems, that knowledge didn't reach the broader community.
- Even though tools were available, nobody bothered to check.

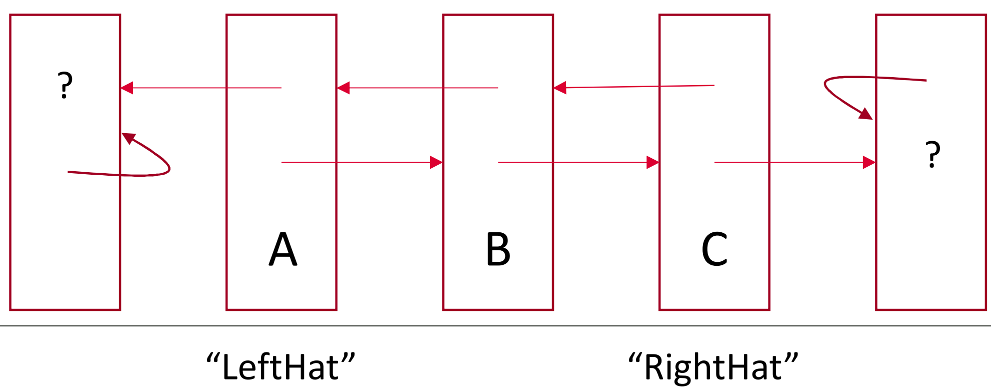
Lesson for Development

- A proof is only as good as the amount of checking that proof receives.
  - Nobody gets published for reviewing somebody else's proof.
- If correctness matters to you, find a way to check your design.
  - Model checkers
  - Fuzzers
  - Coverage tools

An algorithm for a nonblocking deque (double-ended queue)

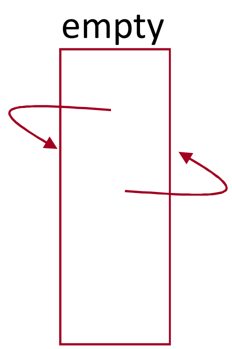
What’s the idea?

- The Snark algorithm uses DCAS (double-compare-and-swap) to implement a lock-free double-ended queue.
- Published in 2001 by David Detlefs, Christine Flood, Garthwaite, Paul Martin, Nir Shavit and Guy Steele (yes, that one) as an improvement over earlier fixed-sized deques.
  - Includes a “sketch” proof of correctness based on possible states, with 7 lemmas and 5 theorems.



What breaks?

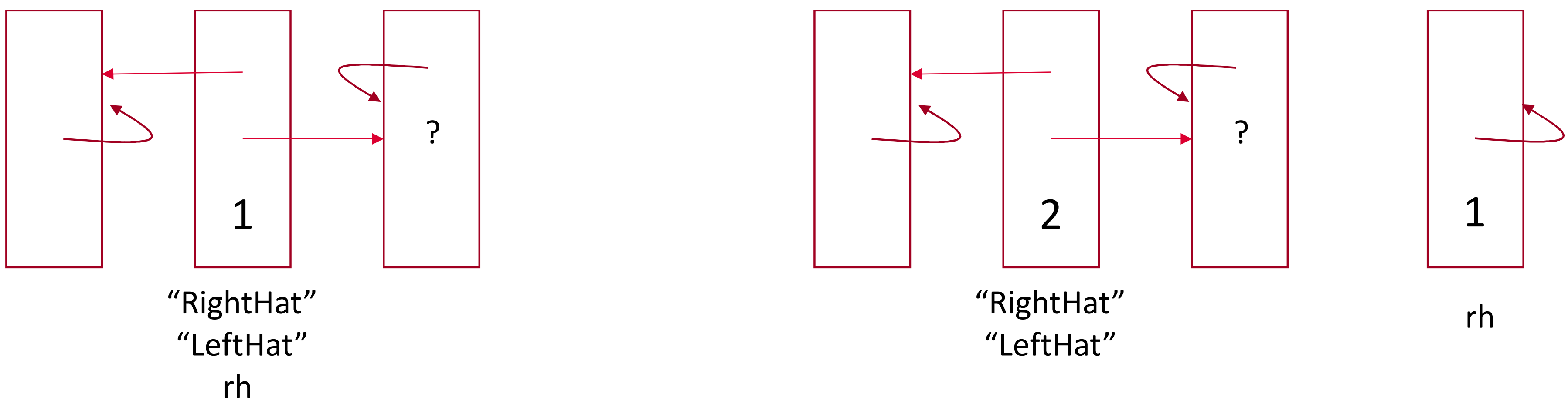
- Element can be removed twice.
- Pop can fail when deque is never empty during its run.




```
val popRight() {
  while (true) {
    rh = RightHat;
    lh = LeftHat;
    if (rh->R == rh) return "empty";
    if (rh == lh) {
      if (DCAS(&RightHat, &LeftHat,
                rh, lh, Dummy, Dummy))
        return rh->V;
    } else {
      rhL = rh->L;
      if (DCAS(&RightHat, &rh->L,
                rh, rhL, rhL, rh)) {
        result = rh->V;
        rh->R = Dummy;
        return result;
      }
    }
  }
}
```

Failed pop when nonempty (linearizability)

1. Thread A starts popRight while not empty, loads “rh”
2. Thread A delayed, other processes pushRight and popLeft so that “rh” has been removed.
3. Thread A resumes, sees “rh->r = rh” due to remove and assumes empty



How was the problem found?

- Model checking, again.
- The original authors and a few more produced a revised paper three years later explaining the error.
  - Used the PVS Specification and Verification System

Lesson for Development

- Again: a proof is only as good as the amount of checking that proof receives.
  - If there’s a proof, there should be a formal model too.
- Heuristically, most bugs have small examples.
  - Just three operations in this case!
  - Can you do exhaustive testing instead of a proof?
- Even experts in concurrency get it wrong
  - Use delay points to get better coverage during stress testing.

An algorithm for sorting

minnebar
minnesota + tech + design + drinks

What's the idea?

- Timsort: a combination of merge sort and insertion sort
  - Designed to perform well on real-world data
  - Originally developed for Python in 2002
  - Ported to Java as `java.util.Collections.sort` and `java.util.Arrays.sort`
- Find existing "runs" that are already sorted within the array
  - If the run is not long enough, use insertion sort to extend it
  - Merge pairs of runs during the sort, and a final merge over all runs at the end
  - Auxiliary data structure is used to track the length of the runs

Claimed Invariant

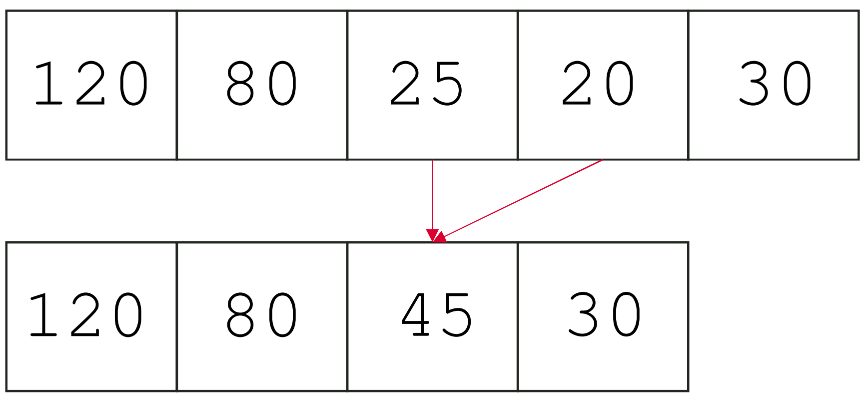
- runLen[n-2] > runLen[n-1] + runLen[n]
- runLen[n-1] > runLen[n]

(Ensures run lengths grow faster than Fibonacci sequence)

```
private void mergeCollapse() {
  while (stackSize > 1) {
    int n = stackSize - 2;
    if (n > 0 && runLen[n-1] <= runLen[n] + runLen[n+1]) {
      if (runLen[n - 1] < runLen[n + 1])
         n--;
      mergeAt(n);
    } else if (runLen[n] <= runLen[n + 1]) {
      mergeAt(n);
    } else {
      break; // Invariant is established
    }
  }
}
```

What breaks?

- The code doesn’t actually preserve the invariant.
  - This leads to more slots in runLength[] being used than expected
  - and causes Java to throw OutOfBoundsException



But 80 + 45 > 125 --- oops!

How was the problem found?

- Stijn de Gouw, Jurriaan Rot, Frank S. de Boer, Richard Bubel, and Reiner Hähnle
  - http://envisage-project.eu/proving-android-java-and-python-sorting-algorithm-is-broken-and-how-to-fix-it/
  - Attempting to prove correctness and failed! (They first proved that counting sort and radix sort were correct.)
- Annotate code with preconditions/postconditions and invariants
  - Use verification tool KeY (an interactive theorem prover) to prove they hold
  - Then reason about worst-case to find counterexample

Aftermath

- The authors submitted bug reports to both Python and Java
  - Python implemented a fix to the algorithm (following the proof of correctness)
    - Even though a counterexample would already have been infeasibly large
    - Question asked: is the extra check required worth it in terms of performance?
      - Yes, number of times this procedure is run is logarithmic in input size.
  - Java bumped up the runLength array size to compensate

Lesson for Development

- Automated tools are still viewed as “research” and not yet ready for prime time
  - … despite years of experience finding bugs in real-world network protocols and algorithms.
  - “Modern formal specification languages and formal verification tools are **not** cryptic and super-hard to learn. Usability and automation are improving constantly. But we need more people to try, test and use our formal tools. Yes, it costs a little effort to start formally specifying and verifying stuff, but not more than, say, learning how to use a compiler framework or a build tool. We are talking days/weeks, not months/years. Will **you** take up the challenge?” – de Gouw et al
- Just because millions of people use your algorithm doesn’t mean it’s safe.
  - Admittedly most people don’t sort 67-million-element arrays

Your thoughts?

- Most errors are a lot more basic than this.
  - If we can't even get simple tasks right, how can we perform complicated ones?
  - You should probably read **`thedailywtf.com`**
- Do you have any examples to share?
  - Of broken algorithms?
  - Of using formal methods?
- All engineering is about trade-offs. We hate to admit it, but correctness isn't always the top priority.

Bonus content

- Is this the Sieve of Eratosthenes?
- People have claimed it is for about 30 years…

```
primes = sieve [2..]
sieve (p : xs) = p : sieve [x | x <- xs, x ‘mod‘ p > 0]
```

- Melissa E. O’Neill, “The Genuine Sieve of Eratosthenes”, Journal of Functional Programming, January 2009
- https://www.cs.hmc.edu/~oneill/papers/

Last-Minute Bonus Content

- “Clustering of Time Series Subsequences is Meaningless”, Eamonn Keogh and Jessica Lin, 2005
- http://www.cs.ucr.edu/~eamonn/meaningless.pdf
- Cites more than 25 papers based on a data-mining technique that produces random results.