ECE/CS 541
Computer System Analysis:
Intro to state-space methods

Mohammad A. Noureddine
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign

Fall 2018

Learning Objectives

- Or what is this course about?

- At the start of the semester, you should have
  - Basic programming skills (C++, Python, etc.)
  - Basic understanding of probability theory (ECE313 or equivalent)

- At the end of the semester, you should be able to
  - Understand different system modeling approaches
    - Combinatorial methods, state-space methods, etc.
  - Understand different model analysis methods
    - Analytic/numeric methods, simulation
  - Understand the basics of discrete event simulation
  - Design simulation experiments and analyze their results
  - Gain hands-on experience with different modeling and analysis tools

Announcements and Reminders

- Probability quiz on **September 20, 2018**
  - No calculators or cheat sheets
  - Will provide pdf and cdf expansions if needed
- Will post HW1 solutions today
- HW2 coming up soon
  - Covers combinatorial methods
- List of possible projects and ideas on the website soon
  - How soon? Hopefully by the end of the week!

Lineage Driven Fault Inject (LDFI)

- So far, we've been thinking about how our system might fail
  - How do we fail our system?
  - Building RBDs, fault trees, reliability graphs, etc.

- But we have a treasure trove of our system did not fail
  - i.e., how our system gave us "good outcomes"

- Transformation the question from "could a bad thing ever happen"
  - Use narrower "how did *this* good thing happen?"

- Answers can provide rich information about the different paths that a successful request can take within our system
  - Use the answers to prune out scenarios that do not really matter

Example

- Consider the following example:
  - "Good outcome" = all acknowledged writes are durably stored.

- Consider a write that was durably stored
  - Q: Why was that write durably stored?
  - A: because it is stored on two replicas: *repA* and *repB*.

- Keep going backwards
  - Q: Why was the write stored on *repA*
  - A: because the client issued one or more broadcast requests to store a write

- Identified 4 important events that contributed to the good outcome of a durable write

$$E \equiv \{RepA, RepB, Bcast1, Bcast1\}$$

Lineage Graph

- Backward reasoning brings us to a lineage graph for that durable write
- Space of possible failure scenarios is 2^E
  - But not all are interesting
  - Failing *RepA* and *Bcast2* tells us nothing
- Random strategy cannot tell us that!
- Which failure scenarios are then interesting?
  - Build a fault tree

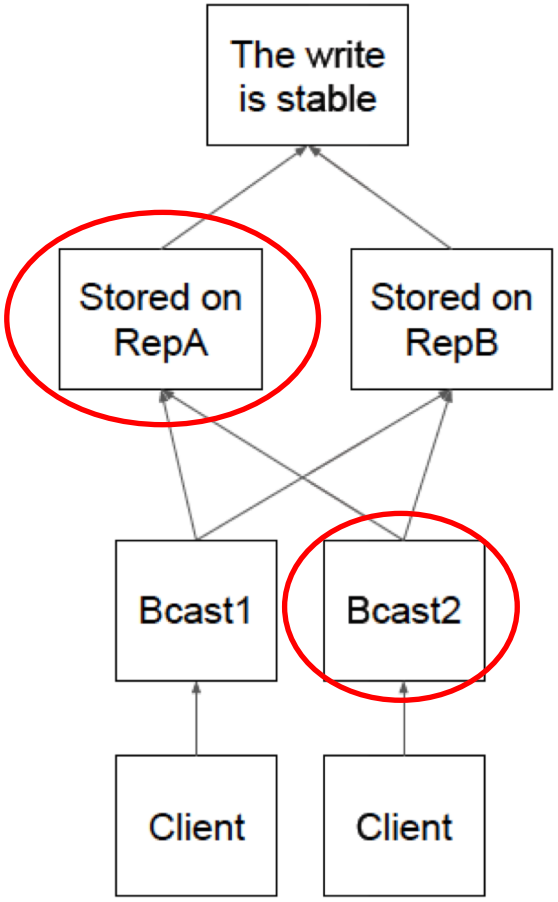


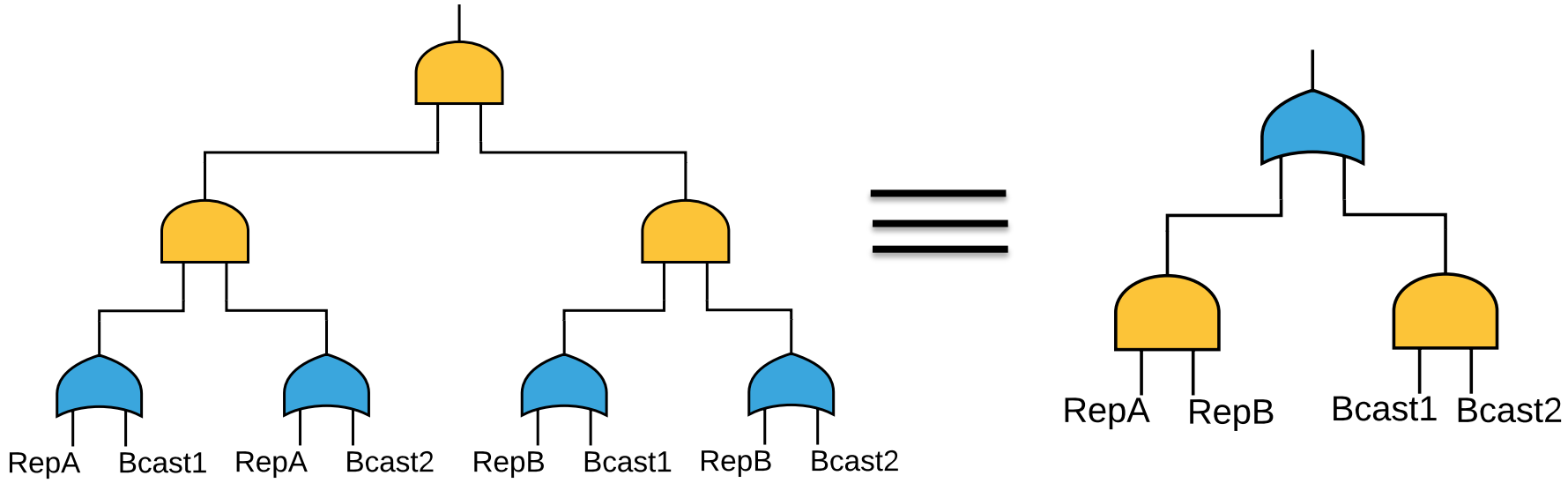
Figure 1. A simple lineage graph

Build the Fault Tree

- They don't actually build the fault tree
  - They build the equivalent *Conjunctive Normal Form* (CNF) expression
  - CNF: product of sums

$$(RepA \vee Bcast1) \wedge (RepA \vee Bcast2) \wedge (RepB \vee Bcast1) \wedge (RepB \vee Bcast2)$$

Path in lineage graph



Min set of useful scenarios

- We can now obtain the minimal solution to the CNF formula that we generated
  - Use off-the-shelf SAT solvers

- We see that the only two scenarios that we care about are

$$\{\{repA, repB\}, \{Bcast1, Bcast2\}\}$$

- Outcome of one execution might not reveal all the dependencies
  - Run the failure scenario, one of two things will happen
    - A new execution path will be revealed
      - Update the fault tree and rerun
    - System fails and you have uncovered a fault tolerance bug

LDFI Process

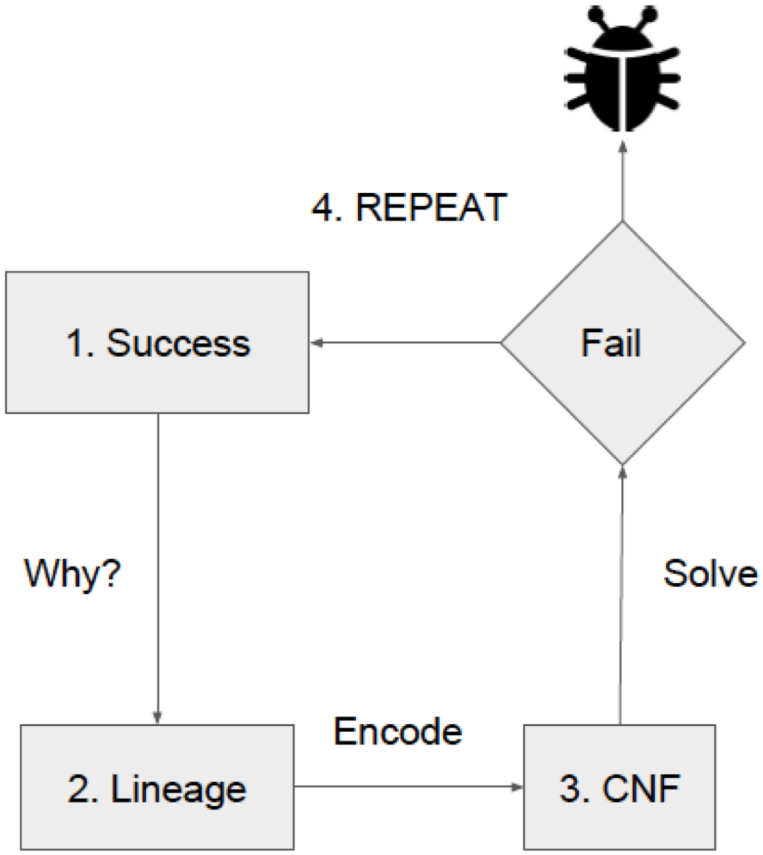


Figure 2. Overview of LDFI.

LDFI Process

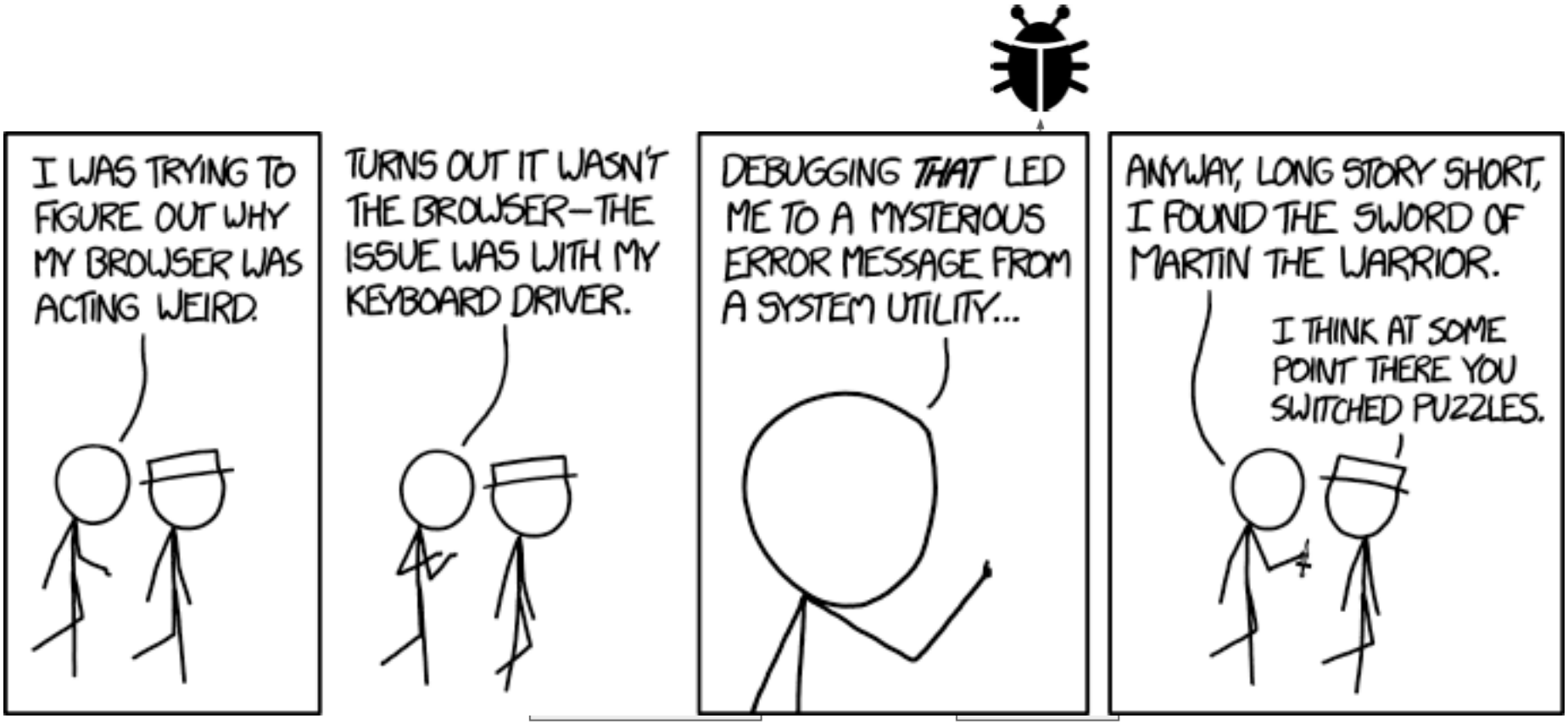


Figure 2. Overview of LDFI.

Results

- Implemented at Netflix to find fault tolerance bugs
- Paper provide interesting details about the challenges they faced and how they overcame them
  - I do recommend reading the paper
- LDFI at Netflix covered the failure space after doing 200 experiments
  - Number of possible scenarios in considered case study is 2^{100}
- **Revealed 11 new critical failures** that could prevent a customer from loading the initial Netflix homepage

Further Reading

- Systems are becoming large, distributed and complex
- Our reliability process is not scalable to such systems
- So how do we build fault trees
  - Let the computers do it – Use machine learning
- **LIFT: Learning Fault Trees from Observational Data**
  - Meike Nauta et al.
  - Appeared at QEST 2018
  - Available on the course website
- Use failure datasets to generate fault trees and use them for analysis
- Interesting project ideas!!!

Objectives for this Module

- Define and classify random processes
- Define Markov processes with focus on Markov chains
- Relax the independence assumption for modeling
  - Discrete Time Markov Chains (DTMC) modeling
  - Continuous Time Markov Chains (CTMS) modeling
    - Motivate queuing theory!
- Understand limitation of Markovian modeling
  - Higher level formalism (Petri-Nets, SANs)
- Markov chains in practice at Google