Logic Problems in Computing

Presented by Brad Kain

27April 2017

Logic Problems in Computing

Review of well-known logic problems and use in software development

Computer scientists have used logic problems to understand and explain challenges in the design and implementation of complex systems. From operating systems to distributed systems, the use of logic problems has allowed software designers to create more effective algorithms. This presentation reviews several well-known logic problems and applications in design, including the *Two Generals, Byzantine Generals, Monty Hall,* and *Dining Philosophers* problems.

Keywords: logic, computing, algorithms, byzantine generals, monty hall, dining philosophers

Two Generals Problem

- Two armies, each led by a general, are encamped in separate valleys near a city
- The generals have agreed to attack, but must attack at the same time to be successful
- Messages to ‘attack’ or ‘retreat’ can only be sent between the armies through a valley controlled by the city defenders
- Messenger might be captured and the message lost

E. A. Akkoyunlu, K. Ekanadham, and R. V. Huber, "Some Constraints and Trade-offs in the Design of Network Communications”, State University of New York, Stonybrook, 1975.

Two Generals Solution

- There are no deterministic solutions, since any sequence of messages will always result in uncertainty by the receiving general
  - Send a series of i messages and respond with a confirmation that the message was received
- A probabilistic approach cannot guarantee certainty
  - Send k messages and act on messages received as long as a level of confidence is reached
- Special case of the *Byzantine Generals Problem*

Byzantine Generals Problem

- Byzantine armies, each led by a general, are encamped near a city
- The generals have agreed to attack, but must attack in sufficient number to be successful
- Messages to ‘attack’ or ‘retreat’ can be sent between the armies
- However, some number of generals are traitors and will not send a false message to cause a defeat

Lamport, L.; Shostak, R.; Pease, M., "The Byzantine Generals Problem”, *ACM Transactions on Programming Languages and Systems*. **4** (3): 382–401, 1982.

Byzantine Generals Solution

- Key assumption is whether to allow messages to be ‘forged’
- Solution exists as long as the number of traitorous generals is less than 1/3 of the total number of generals
- Systems are said to exhibit Byzantine Fault Tolerance (BFT) when it exhibits this ability to operate under limited failures
  - Practical Byzantine Fault Tolerance" (PBFT) algorithm (Liskov, Castro)
  - Upright open source library
  - Bitcoin distributed ledger/proof of work mechanism

Monty Hall Problem

- Based on the game show of the 1960s
- Contest chooses one of three doors each of which conceals a prize of a brand new car or goats
- The host – Monty Hall – opens one of the remaining doors to reveal a goat and allows the contestant to choose another door
- Should the contestant switch doors?
- Functionally equivalent to the 'Three Prisoners Problem'

"The Monty Hall Problem", Steve Selvin (letter), *The American Statistician,* 12 May 1975.

Monty Hall Solution

- Key to the solution is the observation that the host provides additional information by revealing one of the two goats
  - Switching wins 2/3 times and not switching wins only 1/3 times
  - if the contestant initially picks a goat (2 of 3 doors), the contestant *will* win the car by switching because the other goat can no longer be picked, whereas if the contestant initially picks the car (1 of 3 doors), the contestant *will not* win the car by switching
- Simple solution however controversial, but can be demonstrated in simulations to be correct
- Variants based on the behavior of the host are common

Example– Contestant Chooses Door 1

Behind door 1	Behind door 2	Behind door 3	Result if staying at door #1	Result if switching to the door offered
Car	Goat	Goat	**Wins car**	Wins goat
Goat	**Car**	Goat	Wins goat	**Wins car**
Goat	Goat	**Car**	Wins goat	**Wins car**

Dining Philosophers Problem

- Five silent philosophers sit at a round table with bowls of spaghetti and want to eat and think
- Forks are placed between each pair of adjacent philosophers
- Philosopher requires two forks to eat, but can take a fork as available
- Philosopher eats for a period of time and returns the forks to table
- What is a solution that allows all philosophers to eat?
  - Avoid *Deadlock* or *Resource Starvation*

Dijkstra, E. W., "Hierarchical ordering of sequential processes", June 1971, Acta Informatica 1(2): 115–138.

Dining Philosophers Solution

- *Resource hierarchy* – assign a partial ordering on resources and a rule that a process acquires resources based on lowest order first
- *Arbitrator or Waiter* – introduce an entity that grants permission for a process to acquire resources
- Chandry/Misra – uses messages to coordinate resources
  1. For every pair of philosophers contending for a resource, create a fork and give it to the philosopher with the lower ID. Each fork can either be *dirty* or *clean.* Initially, all forks are dirty.
  2. When a philosopher wants to use a set of resources (*i.e.* eat), said philosopher must obtain the forks from their contending neighbors. For all such forks the philosopher does not have, they send a request message.
  3. When a philosopher with a fork receives a request message, they keep the fork if it is clean, but give it up when it is dirty. If the philosopher sends the fork over, they clean the fork before doing so.
  4. After a philosopher is done eating, all their forks become dirty. If another philosopher had previously requested one of the forks, the philosopher that has just finished eating cleans the fork and sends it.