Lecture 22
Markov Analysis of Software Specifications
Analysis of Software Artifacts
Somesh Jha
Paper

- J.A. Whittaker and J.H. Poore
  Markov Analysis of Software Specifications
  ACM Transactions on Software Engineering Methodology
  Vol 2, No 1, January 1993, pages 93-106
View a Program as a Graph

• view a system as a state machine with three classes of states: *invocation*, *usage*, and *termination*

• system starts in the *invocation* state and then moves to one of the usage states because of stimuli from the user or the environment

• system transitions between usage states because of user/environment stimuli

• after the system is done, it moves to the *termination* state
View of the System

Figure 1: Software Usage Model
How to Obtain the State Machine?

• this can be derived from the system specification

• we will assume from here on that we have the state-machine model of the software system
The Markov Chain

• we want to turn the state machine into a Markov chain

• how does one do that?

• assign probabilities to transitions such that sum of the probabilities of all the transitions emanating from a state \( s \) is one

• this is called the *statistical phase*
Payoff

• what is the pay back from building the Markov chain?

• helps us select test data

• calculate expected latency of the system

• helps us with performance optimization of a system
Uninformed Approach

• use this approach in absence of data

• each transition from a state \( s \) has equal probability

• if a state \( s \) has \( k \) transitions \( T_1, \ldots, T_k \) emanating from it, each transition \( T_i \) has probability \( \frac{1}{k} \)
Problem with the Uninformed Approach

- does not model the actual usage of the system
- in practice, there are few paths through the system which are traversed a lot
- we want the probabilities to reflect the actual usage of the system
- for example, when I access my brokerage account, I read news more often than making trades
How to get the Probabilities?

- build prototypes and track the paths through the prototypes as some users (also known as *guinea pigs*) use it

- assume that prototyping produces $N$ paths through the system
  - this is also called *corpus*

- suppose a state $s$ of the system appears $i$ times in the corpus

- suppose a transition $s \rightarrow s'$ appears $j$ times in the corpus

- probability of the transition $s \rightarrow s'$ is $\frac{j}{i}$
Markov chain completed

• assume that we have turned the state-machine into a Markov chain

• now we move on to generating test cases

• generate test cases by random sampling of the Markov chain
Example Markov Chain

![Sample Markov Chain Diagram](attachment:image.png)

Figure 2: Sample Markov Chain
Generating Test Cases

• suppose we want to generate $N$ test cases

• a complete path is simply a path from the invocation state to the termination state

• we present a procedure to generate a random complete path through the Markov chain

• in order to generate $N$ complete paths, just repeat the procedure $N$ times
Generating a Path by Random Sampling

• initially, the path $\pi$ has the *invocation* state
• let $s$ denote the last state of the path $\pi$
• if $s$ is the *termination state*, we stop
• otherwise, let $s_1, \ldots, s_k$ be the successors of state $s$
• pick a state out of the set \{ $s_1, \ldots, s_k$ \} such that probability of picking state $s_i$ is the probability of the transition $s \rightarrow s_i$
• append $s_i$ to the path $\pi$ and repeat the steps
Characteristic of the Test Cases

• consider a path \( \pi = s_0, s_1, \cdots, s_k \)

• probability of this path is given by the following number:

\[
\prod_{i=0}^{k-1} p(s_i \rightarrow s_{i+1})
\]

• this probability corresponds to the likelihood that the path \( \pi \) will be followed during the normal operation of the system

• assuming we have limited resources for testing we can pick test-cases based on the usage of the system

  – pick \( N \) most likely paths
Expected Performance

- suppose we have a latency (denoted by $L(s \rightarrow s')$) for each transition in the Markov chain

- remember that a state-machine is an abstraction of the system and so each transition in the state-machine can mean a complex set of operations at the concrete level.

- we want to find the expected latency of the system based on the Markov chain or usage model
Algorithm for Expected Performance

- generate $N$ paths $\pi_1, \ldots, \pi_N$ through the Markov chain using random sampling

- latency of the path $\pi$ (denoted by $L(\pi)$) is simply the sum of the latency of the transitions on the path

- expected latency of the system is simply the average latency given by the expression below:

$$\frac{1}{N} \sum_{i=1}^{N} L(\pi_i)$$

- for large $N$, the expression given above converges to the true latency
Performance optimization

- for performance optimization, a designer might want to look at paths with high probability through the system

- these paths are called *hot paths*

- it is easy to generate the $M$ complete paths through the system that have higher probability than any other complete paths

- a designer or an architect can look at these hot paths and make sure that at least the system has *low latency* along these paths
Transition Matrix $M$

- Suppose we have $n$ states in our Markov chain

- Markov chain can be represented as $n \times n$ matrix

- The $i, j$-th entry of the matrix is the probability of transitioning from state $i$ to $j$

- If there is no transition from state $i$ to $j$, that entry is 0
Usage distribution

• Suppose the following limit exists:

\[ L = \lim_{n \to \infty} M^n \]

• each row of \( L \) adds up to one
  there are some technical conditions required for the following to be true

• what does each entry the first row mean?

• let the \( i \)-th entry of the first row be \( p_i \)
  we call this the usage probability of state \( i \).
High Usage states

• identify, *high usage* states, i.e., states with high *usage probability*

• intuitively, during the operation of the system states with high usage will be encountered more often

• this can identify critical states of a system