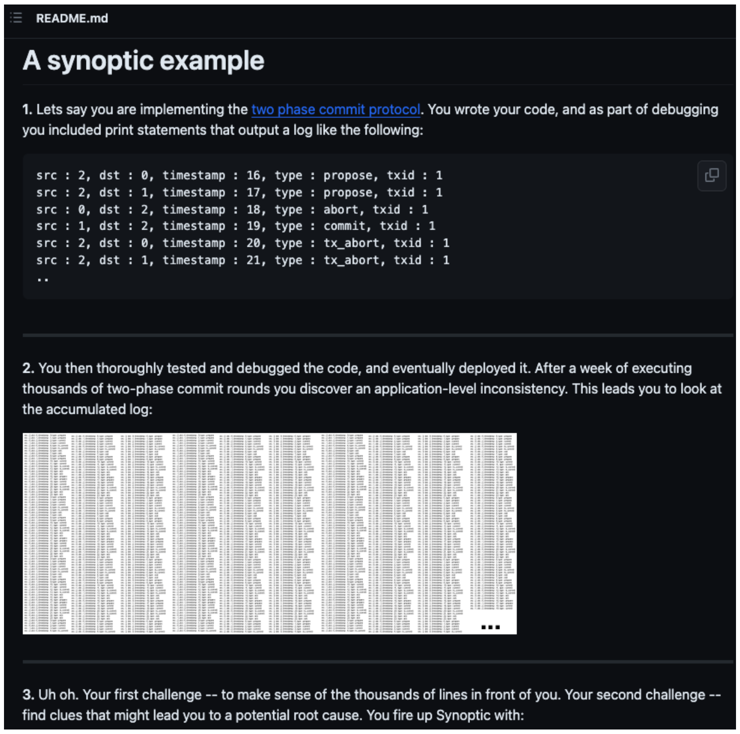
IN-CLASS-4 EXERCISE

Team: CodeCraftersH - Members: Harshita Loganathan, Hemangani Nagarajan

Question 1

The Synoptic build adheres to best design practices, including the presence of thorough documentation. It includes a README file that provides clear explanations of usage, accompanied by examples.



README.md

A synoptic example

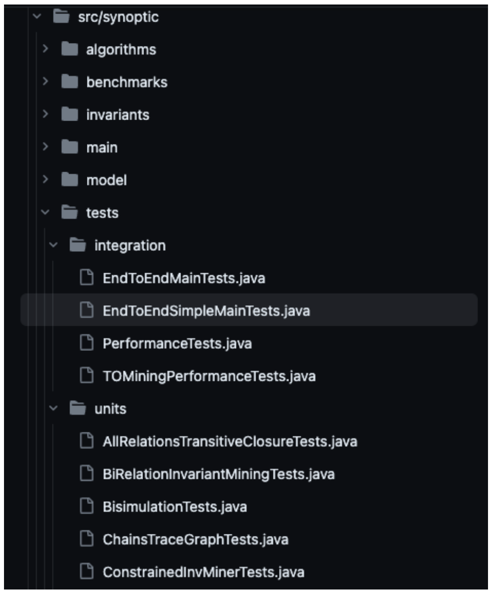
1. Lets say you are implementing the two phase commit protocol. You wrote your code, and as part of debugging you included print statements that output a log like the following:

```
src : 2, dst : 0, timestamp : 16, type : propose, txid : 1
src : 2, dst : 1, timestamp : 17, type : propose, txid : 1
src : 0, dst : 2, timestamp : 18, type : abort, txid : 1
src : 1, dst : 2, timestamp : 19, type : commit, txid : 1
src : 2, dst : 0, timestamp : 20, type : tx_abort, txid : 1
src : 2, dst : 1, timestamp : 21, type : tx_abort, txid : 1
..
```

2. You then thoroughly tested and debugged the code, and eventually deployed it. After a week of executing thousands of two-phase commit rounds you discover an application-level inconsistency. This leads you to look at the accumulated log:

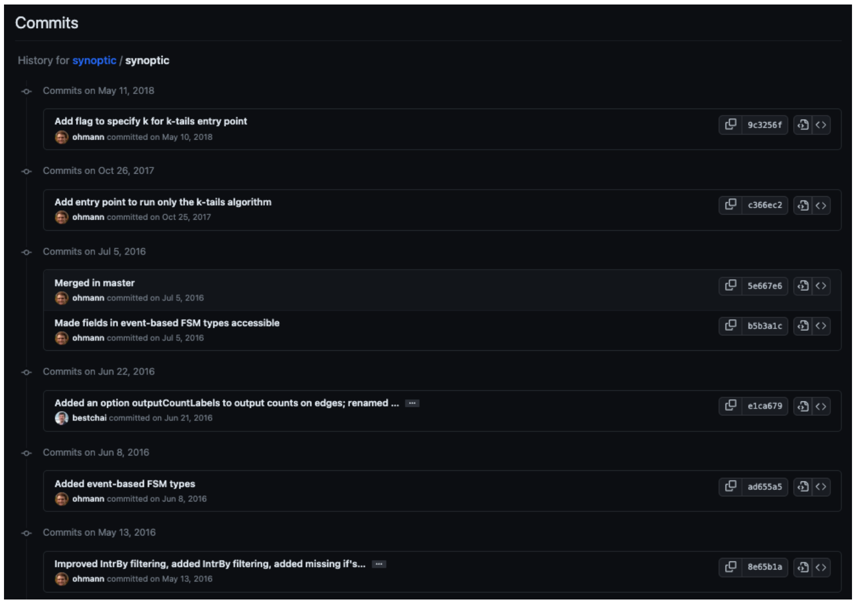
3. Uh oh. Your first challenge -- to make sense of the thousands of lines in front of you. Your second challenge -- find clues that might lead you to a potential root cause. You fire up Synoptic with:

The Synoptic build incorporates both unit and integration test cases that encompass a range of scenarios, ensuring it has good testability.



```
src/synoptic
  algorithms
  benchmarks
  invariants
  main
  model
  tests
    integration
      EndToEndMainTests.java
      EndToEndSimpleMainTests.java
      PerformanceTests.java
      TOMiningPerformanceTests.java
    units
      AllRelationsTransitiveClosureTests.java
      BiRelationInvariantMiningTests.java
      BisimulationTests.java
      ChainsTraceGraphTests.java
      ConstrainedInvMinerTests.java
```

The Synoptic build effectively manages versions using Git, accompanied by descriptive commit messages for clear understanding. This practice facilitates easier debugging.



Commits

History for synoptic / synoptic

Commits on May 11, 2018

Add flag to specify k for k-tails entry point
ohmann committed on May 10, 2018 — 9c3256f

Commits on Oct 26, 2017

Add entry point to run only the k-tails algorithm
ohmann committed on Oct 25, 2017 — c366ec2

Commits on Jul 5, 2016

Merged in master
ohmann committed on Jul 5, 2016 — 5e667e6

Made fields in event-based FSM types accessible
ohmann committed on Jul 5, 2016 — b5b3a1c

Commits on Jun 22, 2016

Added an option outputCountLabels to output counts on edges; renamed ...
bestchai committed on Jun 21, 2016 — e1ca679

Commits on Jun 8, 2016

Added event-based FSM types
ohmann committed on Jun 8, 2016 — ad655a5

Commits on May 13, 2016

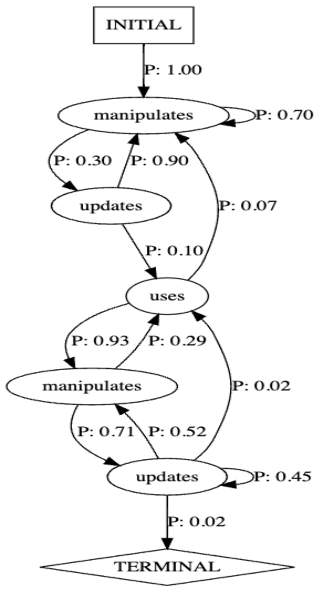
Improved IntrBy filtering, added IntrBy filtering, added missing if's...
ohmann committed on May 13, 2016 — 8e65b1a

Question 2

In order to integrate tracing statements into version 1, one approach was to locate model, view, and controller components in the **TicTacToeBlock** and **TicTacToeGame** class files. Then, tracing statements were affixed in the appropriate manner, employing terminology such as "manipulates," "uses," and "updates." Log statements that included the value 'manipulates' were integrated into model setter functions and assignment operations in the same manner as the variables 'player' and 'movesLeft', both of which are considered model components. Furthermore, log statements pertaining to 'updates' were integrated into the code responsible for modifying the view.

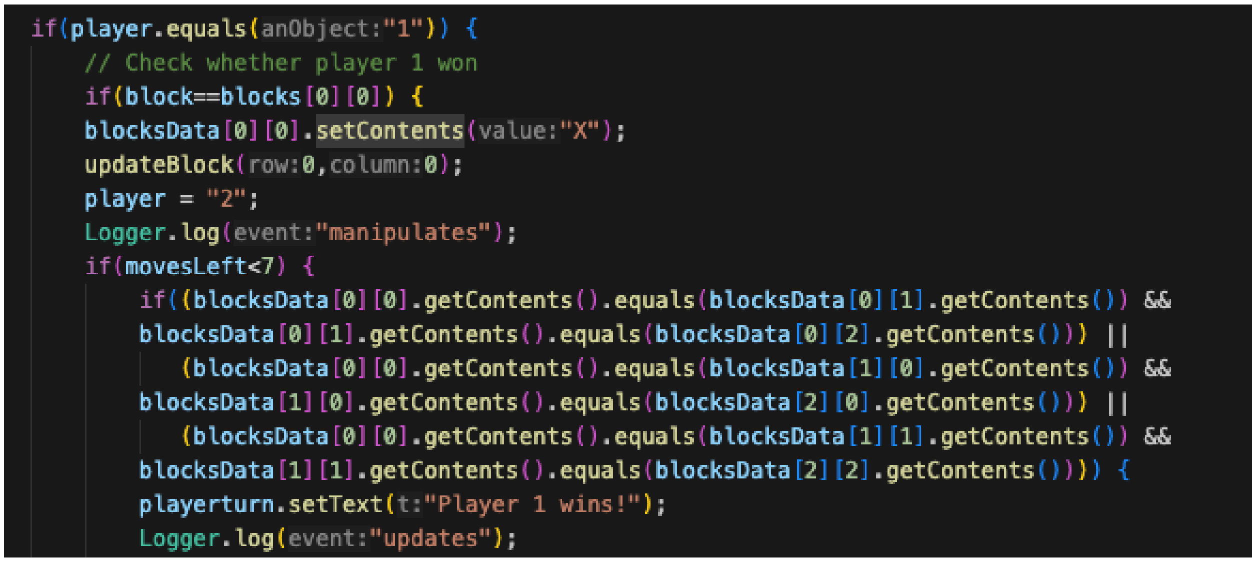
Question 3

Various scenarios were traced, encompassing victories for both Player 1 and Player 2, as well as draws. Resetting the game, either during or after its conclusion, and non-participation in a contest were also considered. Wins could occur diagonally, vertically, or horizontally. The FSA diagram remained unaffected by the inclusion of click functionality on tiles, and sampling stopped once all trace scenarios were explored. The final FSA obtained is presented alongside.



Question 4

- **Expected Behaviour:** Each game initiates from the initial state and advances to the ‘manipulates’ state upon invoking the set functions, which initializes each value and the view gets updated.
- **Unexpected behavior:** This occurs when the ‘updates’ event is initiated, subsequently passed to the ‘manipulates’ event, and yet another ‘updates’ event is generated. Please see the below image. Furthermore, version 1 allows the user to re-select a tile that was previously selected.


```
if(player.equals(anObject:"1")) {
    // Check whether player 1 won
    if(block==blocks[0][0]) {
    blocksData[0][0].setContents(value:"X");
    updateBlock(row:0,column:0);
    player = "2";
    Logger.log(event:"manipulates");
    if(movesLeft<7) {
        if((blocksData[0][0].getContents().equals(blocksData[0][1].getContents()) &&
        blocksData[0][1].getContents().equals(blocksData[0][2].getContents())) ||
           (blocksData[0][0].getContents().equals(blocksData[1][0].getContents()) &&
        blocksData[1][0].getContents().equals(blocksData[2][0].getContents())) ||
           (blocksData[0][0].getContents().equals(blocksData[1][1].getContents()) &&
        blocksData[1][1].getContents().equals(blocksData[2][2].getContents()))) {
        playerturn.setText(t:"Player 1 wins!");
        Logger.log(event:"updates");
```

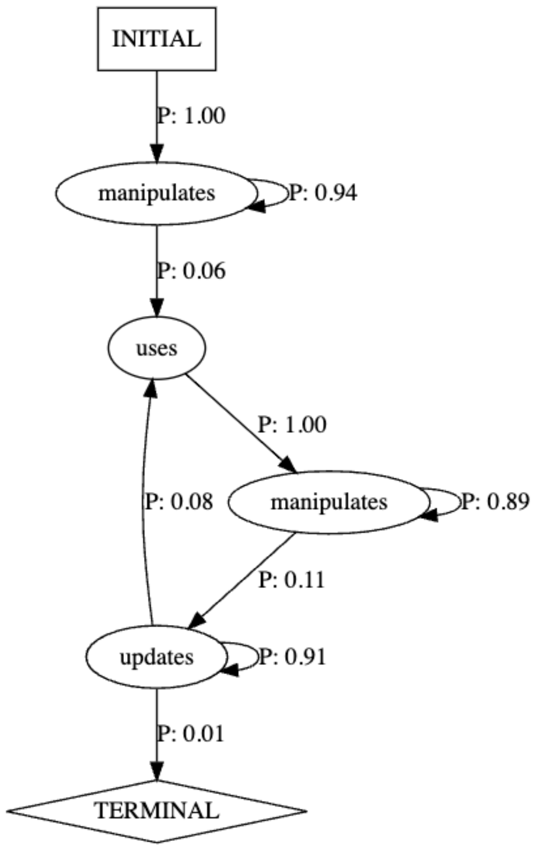
Question 5

Tracing statements were easier to incorporate into the second version of the application. As a result of the implementation of MVC architecture, it was easier to distinguish between locations of the code where the controller modified the model and where the view was altered, as opposed to version 1, which was a single file.

Question 6

The FSA adheres to the MVC architecture pattern by invoking the controller from the initial state, which **manipulates** the model and **updates** the views.

No significant deviations were observed in the sampling methodology when comparing application versions 1 and 2. A range of potential outcomes were discussed, including player 1 and player 2 emerging victorious, achieving victories longitudinally and vertically, and situations in which the game is restarted. The sole distinction is that in the second version, re-clicking on an object is not considered a legal action; therefore, such instances are not taken into account when sampling traces.



Question 7

- **Automated:** The sampling procedure can be automated by generating numbers from 1 to 9 inclusively at random for each participant and maintaining a record of tiles that have been previously selected. The game concludes with a stalemate if no participant wins or if all nine numbers have been selected.
  Additionally, the commands used to generate traces and FSAs can be automated through the creation of a script. FSAs can be generated from the testing records that are also included during the QA testing phase.

- **Semi - Automated:** In a semi-automated system, the input for the sequence of actions for Player 1 and Player 2 consists of a comprehensive catalog of scenarios that are reviewed in advance. It is possible to automate a portion of the procedure by generating FSAs for these cases through the execution of a script.

Question 8

One potential drawback of the random number generation method is its inability to account for a variety of scenarios, such as Player 1 or Player 2 emerging victorious in various manners or the match culminating in a draw. As a result of the cases' stochastic nature, it is possible for similar traces to be generated, leading to the omission of rare or even significant cases.

Semi-automation relies on a predefined catalog of scenarios, limiting the system's adaptability to unforeseen situations or dynamic changes.

Question 9

In order to compare two inferred models, it is necessary to determine which states and transitions are identical in both models, followed by the identification of states and transitions that are deemed redundant and omitted from one of the models. A comparison of the identifiers will reveal two equivalent states. Two transitions that possess an identical initial and final state may be considered equivalent. However, this must be confirmed through an examination of neighboring states and transitions.

In this context, both FSAs consist of three states: "uses," "modifies," and "updates." By comparing these states, we confirm their similarity in both models. Then, we focus on the transitions: in version 1, a transition from "updates" to "manipulates" exists, which is absent in version 2. This method not only clarifies the structural differences but also serves as an effective tool for debugging, analysis, and education in understanding state machines and model inference.

Question 10

In software engineering, model inference is particularly useful in analyzing and visualizing the behavior of complex software systems. This process starts with developers inserting tracing statements into their software, such as a gaming application. These statements log specific events during execution, for instance, in a Tic-Tac-Toe game, logging events like moves, wins, or player switches. When the software runs, these traces are collected, providing a rich dataset. Tools like Synoptic can then infer a model from these traces, effectively constructing a state-transition graph that represents the software's possible states and transitions. This model is invaluable for developers as it offers an intuitive, visual understanding of the system's behavior, facilitating easier debugging, refinement, and documentation. This method becomes particularly beneficial in large-scale or complex projects, where understanding interactions and behaviors of different components is crucial. By providing a clear visualization of the software's behavior, it aids in efficient debugging and quality assurance, ensuring the system behaves as intended and helping to identify edge cases or unanticipated event sequences.