CS 520
In-class exercise 4
Model inference

Due: **Wednesday, April 21, 2021, 9:00 AM** via Moodle. This in-class exercise is a group submission. This means that **each group only needs to submit their solution once** and also that every student in a group will get the same grade. You will work with students within your group, but not with students from other groups. Multiple groups' submissions may not be created jointly. Late assignments will be accepted for extenuating circumstances.

Overview and goal

The high-level goal of this exercise is to learn how automated model inference works and its limitations, then to apply it to real-world traces to infer models of real-world phenomena, and learn something interesting from the resulting models.

Background

Model inference uses a set of observations of how a process executes to the produce a model of everything the process can do. For example, imagine watching ten different people, each, bake a pie, and writing down every step each of them takes. What you end up with is ten traces of executions of the pie baking process. Feed these ten traces into a model inference tool, and it will produce a model of pie baking. The model has a start state and an end state, and everything in between is some way of describing different possible traces. Every trace through the model (from the start state to the end state) is a way to bake a pie. Typically (but depending on the tool), this model will include the ten traces you already observed, but it may include others as well. These others are generalizations of the observed traces.

Of course, model inference doesn't just work for pie baking. The more typical approach is to execute a software system many times and record logs of these executions. These logs (traces) could be something like every method that executes, or it could be the logging information developers chose to put into the system. Feeding these log traces into a model inference tool produces a model of possible system behavior, some observed and some unobserved.

We will be using the Synoptic model inference tool documented here:
Paper: http://people.cs.umass.edu/~brun/pubs/pubs/Beschastnikh11fse.pdf
Git repo: https://github.com/ModelInference/synoptic.git

What to do

Forming groups

1. Team up in groups of size 3, 4, or 5. (If you cannot find enough members, raise your hand and ask the instructor.)
2. Create a new group on Moodle (see "In-class exercise 4: Group selection"), and add all group members.

Set up
Make sure that you have Git, Apache Ant, Java, and Graphviz installed:

- Git: https://git-scm.com/
- Apache Ant: http://ant.apache.org/
- Java: https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html // The Synoptic model inference tool needs the Java 8 regular expression library.
- Graphviz: https://graphviz.org/ // This could optionally be used to convert the inferred model to different formats

For version 1 (v1.0.0) of the RowGameApp from homework 1:

1. Clone the `cs520-Spring2020` git repository (if you have a clone from a previous assignment, delete it, or clone to a new location):

   ```
   git clone https://github.com/LASER-UMASS/cs520-Spring2020 rowGameApp-v01
   ```

   Change into the rowGameApp-v01 folder: *cd rowGameApp-v01*

   Switch to the first version (v1.0.0) from homework 1: *git checkout v1.0.0*

   Read the *threeinarow* `README.md` file: `cat threeinarow/README.md`

2. Install the synoptic model inference tool

   cd threeinarow/lib

   Source build:
   git clone https://github.com/ModelInference/synoptic.git
   cat synoptic/README.md // Follow these build instructions

   Binary build:
   Download from here: *https://drive.google.com/drive/folders/12svY1LVbjgmtNYCmi2GT_g0YMukju1Yb?usp=sharing*

3. Compile and run the application using the `main` method in the `RowGameApp.java` file. This will invoke the GUI. Play with it and see the logger print out tracing statements to STDOUT.

   ```
   cd .. // Should be rowGameApp-v01/threeinarow
   ant clean
   ant compile
   java -DLogger.tracing=true -cp bin ThreeInARowGame
   ```

For the second version (v2.0.0) of the RowGameApp from homework 2:

1. Clone the `cs520-Spring2020` git repository again:

   ```
   git clone https://github.com/LASER-UMASS/cs520-Spring2020 rowGameApp-v02
   ```

   Change into the rowGameApp-v02 folder: *cd rowGameApp-v02*

   Switch to the first version (v2.0.0) from homework 2: *git checkout v2.0.0*

   Read the *threeinarow* `README.md` file: `cat threeinarow/README.md`

2. Install the synoptic model inference tool

   cd threeinarow/lib
   mkdir synoptic

cd synoptic
mkdir lib
// Copy the rowGameApp-v01/threeinarow/lib/synoptic/lib/.jar lib*

3. Compile and run the application using the `main` method in the `RowGameApp.java` file. This will invoke the GUI. Play with it and see the logger print out tracing statements to STDOUT.

```
cd .. // Should be rowGameApp-v02/threeinarow
ant clean
ant compile
java -Dlogger.Logger.tracing=true -cp bin RowGameApp
```

Sampling traces and inferring a model from them

You should develop a technique for sampling the traces of the RowGameApp. Keep in mind that lots of things can affect model inference:

- How many traces there are.
- How long the traces are.
- How diverse the traces are.

For the RowGameApp, the inferred model should use the following events: **uses**, **manipulates**, **updates**. Thus, your tracing statements will need to be for these events.

For version 1 of the RowGameApp, here are the proposed steps:

1. Add tracing statements to the RowGameApp. For instance, the ThreeInARowGame class delegates to the Logger class to print tracing statements for the **uses** events.
2. Generate a sample trace of this App: *./sampleTrace.sh UNIQUE_ID*. For instance, *./sampleTrace.sh 03*.
3. Run the model inference tool on your set of sampled traces from the App to produce the inferred model of it: *./modelInference.sh*. Could optionally convert the inferred model from a dot file to PDF: *./saveInferredModelAsPDF.sh*. Also run the model inference tool on your set of sampled traces from the App to produce the invariants of it: *./saveInvariants.sh*.
4. Inspect the inferred model of the RowGameApp to decide whether or not to generate additional traces from the App. If so, repeat Steps 2, 3, and 4. If not, continue to Step 5.
5. Commit the classes with the tracing statements, the traces, and the final inferred model (as PDF or PNG) as well as the final invariants (as TXT).

For version 2 of the RowGameApp, repeat the above steps.
For both versions, take notes about how you approach steps 1, 2 and 4.

Questions

Using the Synoptic documentation, your notes, and the results, answer the following questions:

1. For the Synoptic build, identify 2 design principles or best programming practices that are satisfied. For each design principle or best practice satisfied, illustrate with an example from the build.

2. Briefly explain how you added the tracing statements to the first version of the RowGameApp (Step 1 above).

3. Briefly explain your technique for sampling the traces of the first version of the RowGameApp (Step 2 above). This explanation should include how you decided to stop sampling.

4. For your inferred model of the first version of the RowGameApp, briefly describe one expected behavior. Briefly describe one unexpected behavior.

5. Was it easier to add the tracing statements to the second version of the RowGameApp (Step 1 above). Why or why not?

6. Your inferred model of the second version of the RowGameApp should be the MVC architecture pattern. (If the **manipulates** event were split into two events: **sets**, **gets**), the inferred model should actually be the PAC variation of this pattern.) Did you need to change your technique for sampling the traces of the RowGameApp (Step 2 above) to produce that inferred model? Briefly explain why or why not.

7. Propose a semi-automated of fully automated technique for sampling the traces of the RowGameApp (Step 2 above).

8. What is the primary disadvantage of any such trace sampling technique?

9. Propose one technique for illustrating the differences between two inferred models (represented as FSAs).

10. What is one use case where model inference would be helpful to the developers?

Deliverables

Your submission, via Moodle, must be a single (one per group) archive (`.zip` or `.tar.gz`) file with name `<group name>-inclass4.<zip/tar.gz>`, containing:

1. `answers.pdf or answers.txt`: A PDF or plain-text file with your answers to the above 10 questions. *List all group members on top of this file*.

2. rowGameApp-v01: Your copy of the *threeinarow* repository, with your tracing statements, RowGameApp traces, the invariants (as a TXT file), and the inferred model (as a PDF or PNG file). For example, on a Linux-based machine (e.g., MacOS), you can use the terminal from the *threeinarow* directory and run the command

   tar -vczf rowGameApp-v01.repo.tar.gz .git

3. rowGameApp-v02: Your copy of the *threeinarow* repository, with your tracing statements, RowGameApp traces, the invariants (as a TXT file), and the inferred model (as a PDF or PNG file). For example, on a Linux-based machine (e.g., MacOS), you can use the terminal from the *threeinarow* directory and run the command

   tar -vczf rowGameApp-v02.repo.tar.gz .git