LearnLib Tutorial

Testing Techniques 2024 - 2025

The purpose of this tutorial is to help you with installing and running all necessary tools for learning models with LearnLib.

Pre-requisites

You need the following programs to follow along:

IDE

The `learning-project/` used in this tutorial is a Java Maven project which every IDE nowadays should be able to import.

In this tutorial we use Eclipse, and therefore the `learning-project/` used in this tutorial is already imported in Eclipse, and contains the Eclipse specific project configuration files. Eclipse is available at `https://www.eclipse.org/`. When installing Eclipse it also asks the version of Java SDK to install with it. Advised is the latest Long Term Support(LTS) version: Java 21.

At the end of this document there are also instructions for importing this Maven project into IntelliJ IDEA and Visual Studio Code.

If you are going to use another IDE then eclipse it must be noted that the `learning-project/` used in this tutorial enforces compliance with Java 8 because we use an older learnlib library in the project. Java supports backwards compability, so a Java 21 compiler can be used in Java 8 compliant mode. So make sure you apply this compliance in your specific IDE, because without it you will get compile problems in the project. But normally when importanting a Maven project this should be applied correctly in your IDE.

GraphViz *or* Graphviz Visual Editor

Needed to visualize the learned models. Learnlib will write its learned model to textual `.dot` files. A `.dot` file can be opened in any text editor. However we can better look at the model if it is visualized. We have two options to do this visualization:

- With Graphviz's `dot` command a `.dot` file can be converted to a visualization in a pdf file. When double clicking a `.dot` the model is automatically visualized by the Graphviz viewer.
- Instead of installing Graphviz's `dot` command, you can visualize `.dot` files online at `http://magjac.com/graphviz-visual-editor/`.

Comparison: Using the online tool means that no installation is required, which is convenient if for some reason you do not want to install GraphViz. But its usage requires copying and pasting the `.dot` file from a text editor to

the online tool. However with Graphviz the conversion `.dot` → `.pdf` is automatically done from the learnlib project, and we can double click any `.dot` to get the model visualized by the Graphviz viewer. With GraphViz installed the workflow is more convenient, and it is therefore recommended to install it.

Graphviz installation and configuration:

- GraphViz installation instructions: `http://graphviz.org/download/`.
- Make sure that the path to the `bin` folder of Graphiz is in your PATH variable. The easiest solution is by setting the PATH variable in Eclipse:
  - open menu: Run → Run Configurations...
  - open the Run Configuration for your program
  - go to the Environment TAB, and add:

    ```
    PATH=/my/graphviz/bin/directory:${env_var:PATH}
    ```

    Note: on a Mac or Linux machine we can use 'which dot' command in the terminal to find the graphviz bin path.

1 Setup the learning project in Eclipse

1. Open the given `learning-project` project in Eclipse.
   (a) open menu: File → Open Projects from File System...
   (b) click "Directory..." button on the right
   (c) select the `learning-project` directory
   (d) press "Finish" button

   Note: the `learning-project` uses maven, which by default is installed within Eclipse.

2. Have a look at `ExampleExperiment.java`, and run its main method (by clicking on the run button while the file is open). If all works fine, it learns a model of the hardcoded SUT `ExampleSUL.java`. The result is written to `learnedModel.dot`. In the Eclipse's Package Explorer left-click on the project name and choose "Refresh" to let the new files show up in the `results/` subfolder!
   If graphviz is installed and available on the PATH environment variable, then it also automatically generates a layout of the `learnedModel.dot` file in the `pdf` file `learnedModel.pdf`. See instructions above to set PATH in the Eclipse Run Configuration.

   Compare the SUL in `ExampleExperiment.java` with the learned model.

3. Finally have a look at `src/basiclearner/BasicLearner.java`, to see how you can run and parametrize experiments. Note that using two enums `LearningMethod` and `TestingMethod` we can configure the way we do the learning in a `BasicLearner.runControlledExperiment`:

```
public enum LearningMethod {
 LStar, RivestSchapire, TTT, KearnsVazirani
}
public enum TestingMethod {
 RandomWalk, WMethod, WpMethod, UserQueries
}
```

   In the `main` method in `ExampleExperiment.java` we choose at an high level which learning and testing method we use in the learning setup. This setup lets us easily switch between the different methods. Note that the default testing method is set to `UserQueries`, which means that the user acts as the equivalence oracle: have a look at the hypothesis, and try to think of one.

2 Setup the learner to learn a SUL running in a website

We will now learn the behaviour of a website with learnlib. We do this by running the website in a browser, and performing any input from the learner using the *Selenium WebDriver*. Selenium is an open-source tool that automates web browsers. It provides a single interface that lets you write test scripts in programming languages like Ruby, Java, NodeJS, PHP, Perl, Python, and C#, among others. The *Selenium WebDriver* let you write java-programs to automate GUI interactions in a browser.

2.1 Setup *Selenium WebDriver*

To get the *Selenium WebDriver* to work it needs to install an API at the java client side, and a plugin on the browser 'GUI' side:

1. Install Selenium WebDriver's Java Library, either
   - via *Maven*, if you are familiar with this:
     - goto `https://mvnrepository.com/`
     - search for "org.seleniumhq.selenium:selenium-java"
     - import the maven xml code from the website as a dependency in your pom.xml using the IDE as editor

     Note: current learning project already has selenium version 3.141.59 setup as dependency in the pom.xml file.

- or manually from `https://www.selenium.dev/downloads/`. Download the webdriver **for Java** under the header *Selenium Client & WebDriver Language Bindings*, and add the jars from this zip as a library to your project.

2. Install Selenium WebDriver's "Driver".
   The "Driver" is an executable server that opens up a browser instance and runs the commands from the client on the browser. For each browser version, such as Chrome or Firefox, there is a specific driver. Google develops and maintains Chromedriver for Selenium to support automation on Chromium/Chrome. Thus to use Selenium with the Chrome browser you must download the Chrome webdriver. It is **strongly advice** to use Chrome, because the code in the project is already setup for this browser. For other browsers you need to adapt it.

   The standard Chrome browser nowadays get updated often, which forces testers to also update the chrome webdriver to match the browser version. To help testers google decided to ship a special Chrome browser for testing which is fixed in version and does not get updated. Once this browser and its chrome webdriver is installed we can keep using it for a long time providing us a stable test setup.

   To install the Chrome webdriver:

   - go to `https://googlechromelabs.github.io/chrome-for-testing/`
   - download the 'stable' 'chrome' for testing from that page for your specific operating system. Install that on your operating system.
   - download the 'stable' 'chromedriver' with the same version as your 'chrome' for testing browser. Unzip the downloaded file and put the `chromedriver` file in the `webdriver/` folder in the learning project. The installed location is configured in the project such that the project can start it itself automatically. For more info see the next section.

   To use Selenium in firefox, you need the *geckodriver*.

2.2 Configure Selenium in the learning project's java code

The Chocolate Bar Machine SUL is a webpage, which you can open in any browser. The webpage is locate at `sul/chocolate_bar_machine/website.html`. Open this file in a browser to play with the SUL.

1. The `SeleniumSUL` class, in the `seleniumsul` package in the project, is a SUL-adapter class used by the learner using Selenium to interact with the Chocolate Bar Machine SUL in the browser. The `SeleniumSUL` class gets inputs from the learner and performs them on the Chocolate Bar Machine SUL in the browser. To interact with the browser the Chocolate

Bar Machine SUL uses the Selenium Java API to send inputs and receive outputs from/to the browser.

2. The `SeleniumChocolateBarMachineLearner.java`, in the `seleniumsul` package in the project, configures the Selenium parameters and then calls `BasicLearner.runControlledExperiment` to start learning using the `BasicLearner` class. When configuring Selenium make sure it contains the right

   - path to `sul/chocolate_bar_machine/website.html` containing the Chocolate Bar Machine website,
   - path to the browser-specific driver. The code currently uses chromedriver for Chrome/Chromium, and
   - java-class for the driver for your browser. The code currently uses `ChromeDriver`.

   For the Chrome browser SeleniumChocolateBarMachineLearner.java contains something like:

   ```
   String sulURI = "file:///path/to/chocolate_bar_machine/website.html";
   String chromeDriverLocation = "path/to/chromedriver";
   System.setProperty("webdriver.chrome.driver",chromeDriverLocation);
   WebDriver driver = new ChromeDriver();
   ```

   If you use the Chrome/Chromium browser and you have the `chromedriver` and installed in the `webdriver/` folder in the learning project, then the default configuration in the project will work without needing any changes.

2.3 Learn Chocolate Bar Machine SUL

Learn a first model of the website by running the `main` method of `SeleniumChocolateBarMachineLearner.java` by just clicking the Run button in Eclipse while the file is the current opened file in Eclipse. If all works fine, you should be able to type a counterexample, i.e. a space separated sequence of inputs, and continue with learning. Keep the Chrome browser window next to the IDE window so that you can see what is happening. If you are unable to type the inputs, ensure that in the `SeleniumChocolateBarMachineLearner` class you give the enum value `BasicLearner.TestingMethod.UserQueries` as a parameter when launching the `BasicLearner.runControlledExperiment` learner.

Important: do not interact with the website in the Chrome browser opened by Selenium otherwise the learner may detect none-determinism during learning. If you want to browse the website during learning, then open it in another browser!

If all went well, then you are ready to make the learning assignment!

Appendix: Import project into IntelliJ IDEA

The learning project has project files for Eclipse. However this java project *should* work with any IDE. Here I explain how to import this project into IntelliJ IDEA:

- in IDEA choose "File" → "Open"...
- in the opened file browser browse to the learning project and open the `pom.xml` file. The `pom.xml` file describes the Maven project.
- Say yes to open this as a project
- Maven uses a different layout of projects then Eclipse does. Eclipse uses the `src` folder for your java sources, but Maven doesn't. The none standard use of the `src` folder for your java sources is therefore explicitly specified in the `pom.xml` maven configuratioin file. So when importing a maven project in an IDE using this `pom.xml` the source folder should be set immediately right in the IDE. However if this would fail in IDEA, then to make this folder known as the source folder select with the right mouse button the `src` folder, and in the context menu select "Mark Directory as" → "Sources Root".

Appendix: Import project into Visual Studio Code

The learning project has project files for Eclipse. However this java project *should* work with any IDE. Here I explain how to import this project into Visual Studio Code:

- in Visual Studio Code install the extensions 'Extension Pack for Java' and 'Graphviz Interactive Preview'
- then just the "File" → "Open..." menu to open the learning project folder. Visual Studio Code should automatically recognize it is Java Maven project and opens it correctly for you. Now everything should work.
- Maven uses a different layout of projects then Eclipse does. Eclipse uses the `src` folder for your java sources, but Maven doesn't. The none standard use of the `src` folder for your java sources is therefore explicitly specified in the `pom.xml` maven configuratioin file. So when importing a maven project in an IDE using this `pom.xml` the source folder should be set immediately right in the IDE. However if this would fail in Visual Studio code, then fix this in your project by looking online for instructions.