Artifact Evaluation for

“Using application conditions to rank graph transformations for graph repair” by Lars Fritsche, Alexander Lauer, Andy Schürr and Gabriele Taentzer

In the following, you will find all the steps to reproduce our results and run the evaluation on your own system. For this purpose, we created a virtual machine to ease the installation.

VM Setup

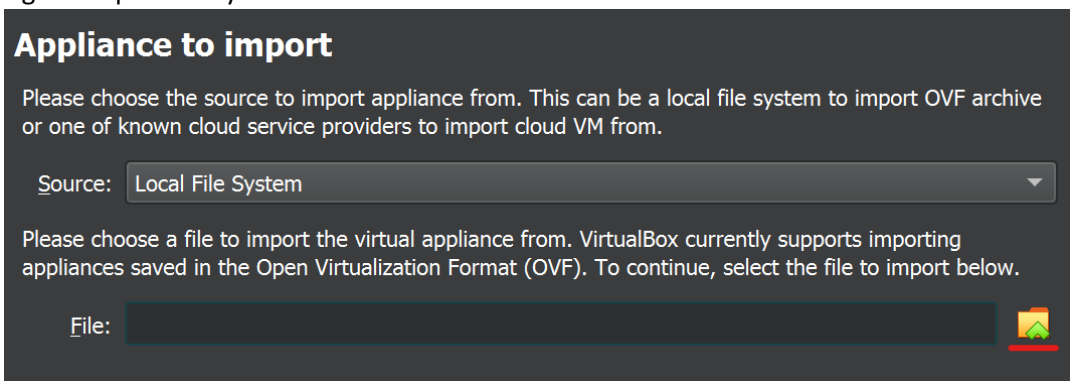
---

1. Please download the VM from Zenodo (10.5281/zenodo.10727438)

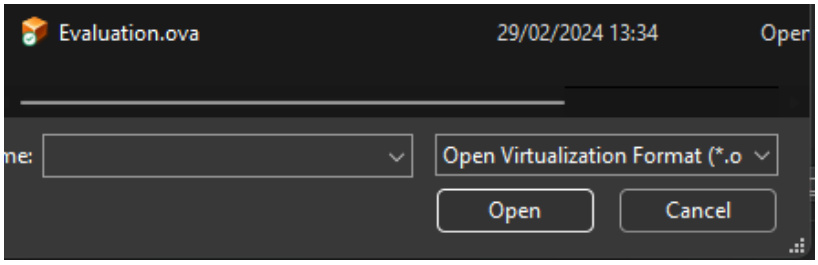
2. Install VirtualBox from here

3. Start VirtualBox and press **Import**

4. A new window opens. There, choose “Source: Local File System” and press the yellow file to the right to open the system’s file chooser

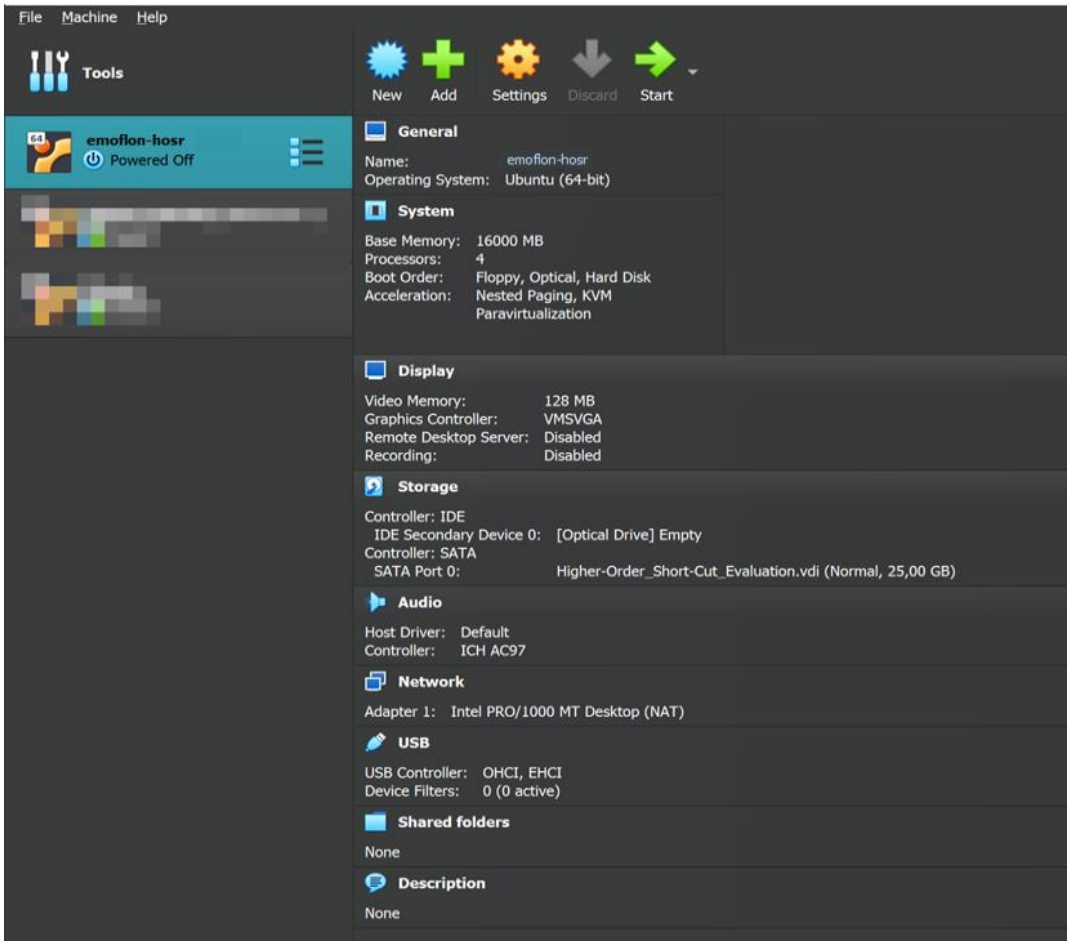


5. Navigate to the downloaded OVA-file and press **Open**

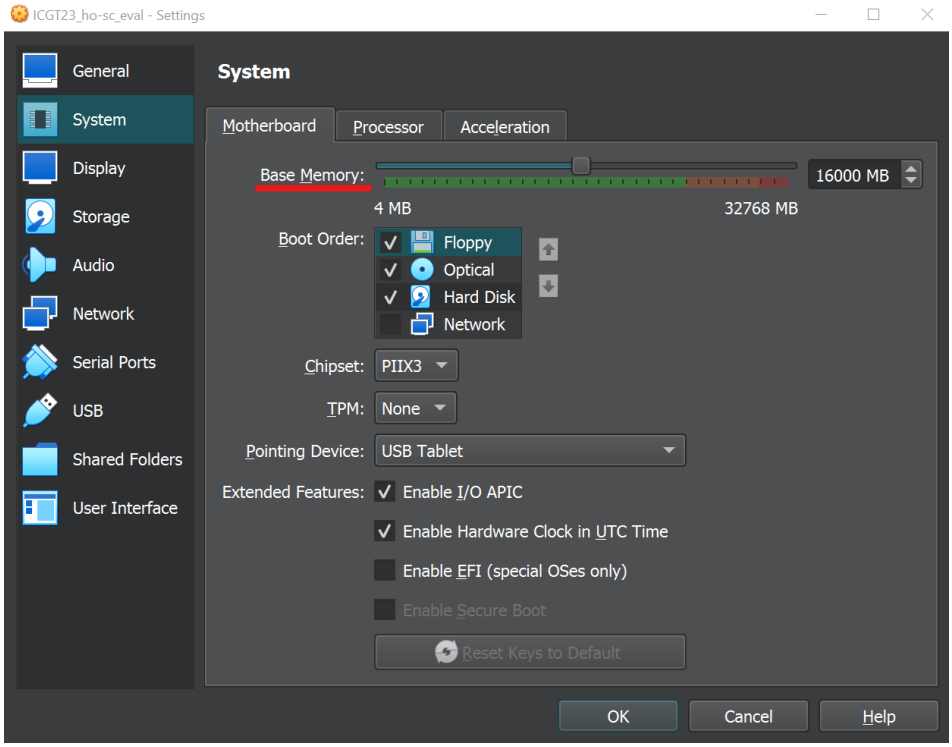


6. Press **Next**. You should now see an overview of the virtual machine's parameters. Here, press **Finish**.

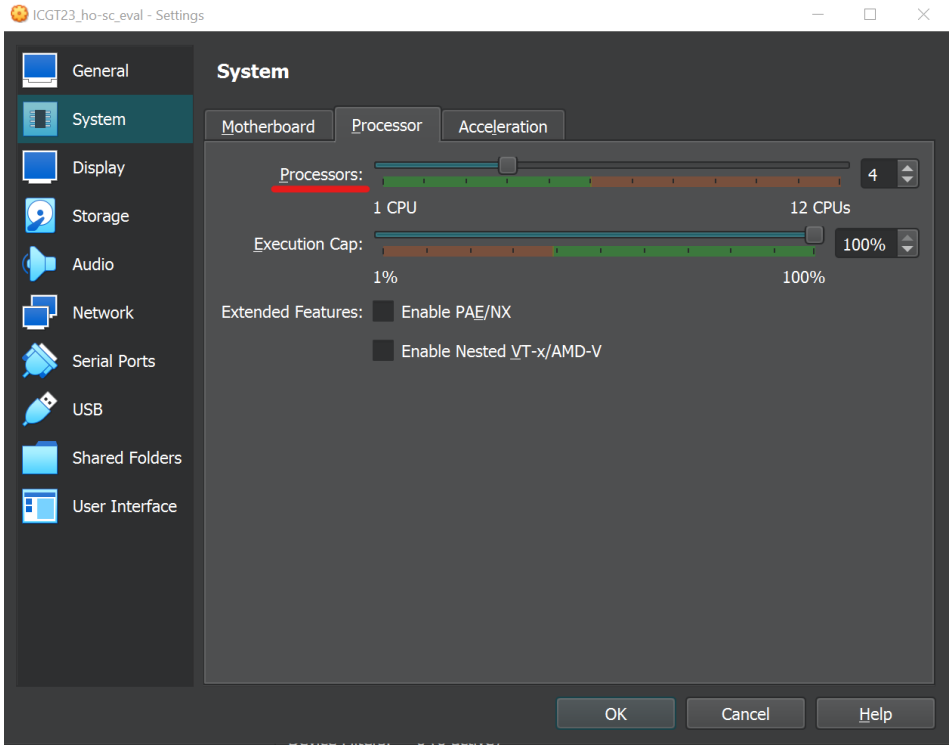
7. You should now see the VM in the upper left corner and a description of the configured system:



8. Press **Settings** and go to **System** and then **Motherboard**. Here you can configure the base memory. For optimal performance, you should choose this value close to the red interval.



Then go to **Processor** and choose as many CPU cores as you can spare. eMoflon relies on a pattern matching engine that highly benefits from an increased number of cores.



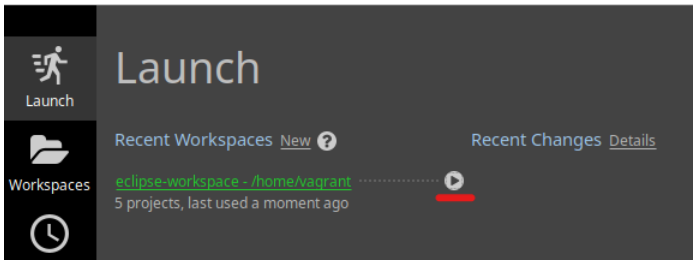
9. Select the VM and press Start. When asked for a username and password, type in **vagrant**

Eclipse Setup

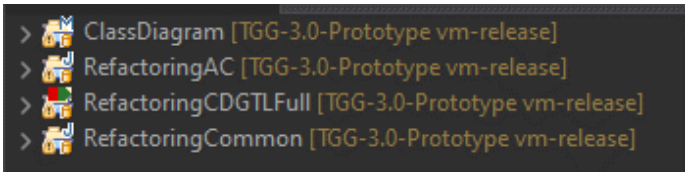
On your desktop you will find a folder, a PDF file, and a link:

- The folder **paper-eval-results** contains the original results used in our paper.
- The **ICGT24-Setup.pdf** file is a copy of this document, but without the first chapter “VM Setup”
- The link **eMoflon::IBeX Eclipse** points to a pre-configured Eclipse version with eMoflon installed.

1. Double-click on the **eMoflon::IBeX Eclipse** link to run eMoflon
2. A quick workspace configuration follows, which you can click through
3. When prompted for the workspace location, make sure that it matches the following one and click the **launch-button**



Now, the Eclipse workspace should be up and running. On the left you should be able to see four projects:



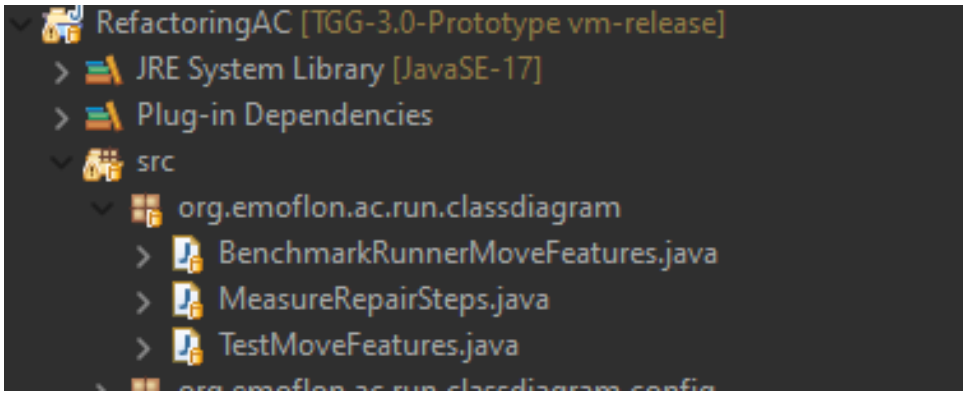
ClassDiagram contains the ecore specifications of our papers running example.
RefactoringCDGTLFull is an eMoflon project that will find our rule and pattern matches.
RefactoringAC contains the control flow and executables of our evaluation.
RefactoringCommon contains a few helper classes such as a container for ranked rule matches.

Benchmarking

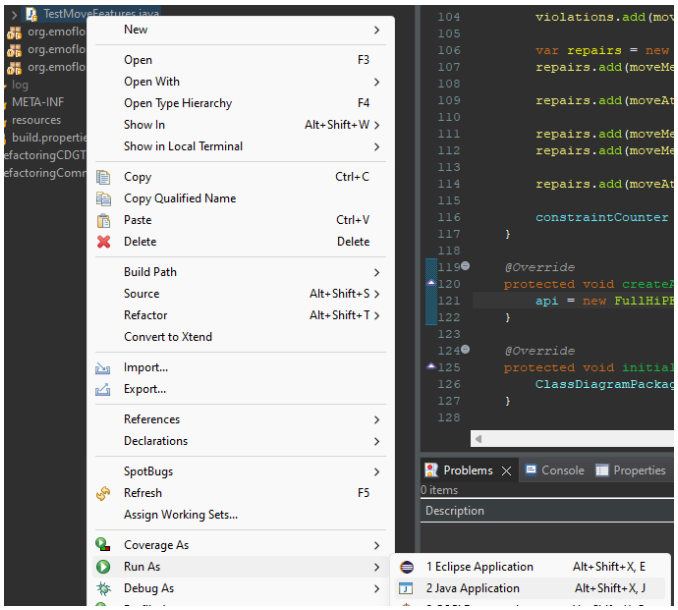
There are three runnables:

- **BenchmarkRunnerMoveFeatures**: Spawns processes to perform our performance evaluation with increasing model sizes.
- **MeasureRepairSteps**: Performs incremental repairs steps for a generated model until it is no longer possible to repair more than would be newly violated.
- **TestMoveFeatures**: Implements our papers running example and prints all ranked rule matches.

You can execute each of these by running them as a Java Application, e.g., via **Run as.**



To run a Main-File, right-click on it and choose **Run As -> Java Application**.



Executing the performance benchmark will yield results like this:

```
initialisationTime;performingStepsTime;totalExecutionTime;modelSize;iterations;ruleMatchCount;patternMatchCount
1.57739;0.4337082;2.0110986;276;10;6000;32121
1.7345305;0.4924629;2.2269939;276;10;6000;32243
1.5902209;0.500289;2.0905101;276;10;6000;32121
1.6189702;0.4199;2.0388704;276;10;6000;32352
```

initialisationTime: This is the time it takes to initially find all matches for the given model
performingStepsTime: The time it takes to execute the given number of iterations
totalExecutionTime: The sum of initialisationTime and performingStepsTime
modelSize: The model size is measured in nodes, e.g., objects (excluding edges)
iterations: The number of performed repair steps
ruleMatchCount: The number of available rule matches
patternMatchCount: The number of found of found pattern matches used to rank rule matches based on their potential for repairing or causing new violations

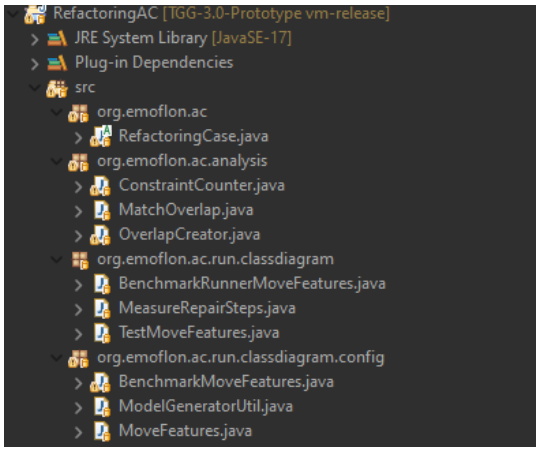
Executing the repair step evaluation will yield results like this:

```
iteration;gain;repairs;violations
10;42;80;38
20;84;152;68
30;128;226;98
40;170;302;132
50;209;375;166
```

iteration: The current iteration.
gain: The accumulated gain, which is the difference of repairs and violations.
repairs: The accumulated repairs until this iteration.
violations: The accumulated violations until this iteration.

Implementation

The main benchmark implementation can be found in the RefactoringAC project.



***.ac**: Contains one class that is responsible for communicating with eMoflon, initializing the model, collecting matches and managing components to track repairs and violations.

***.ac.analysis**: Contains logic to calculate overlaps between rule and application condition matches. Also is responsible for calculating ranked matches based on how many repairs or violations a rule would introduce.

***.ac.classdiagram**: Contains Main-files to execute our evaluation as described above.

***.ac.run.classdiagram.config**: Contains the example specific configuration of our evaluation, e.g., where to overlap rules with application condition patterns and which matches indicate a repair or a violation. You will also find our model generator here as well as the Main-class that is used for our performance evaluation by BenchmarkRunnerMoveFeatures.java (you should use this class instead).

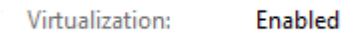
Troubleshooting

Note that our evaluation was performed on a workstation with 64GB RAM and an AMD Ryzen 9 3900x, although we also tested it for the largest model size using a laptop with 12GB and an AMD Ryzen 7 5800H. eMoflon relies on incremental pattern matching, which may consume a lot of memory, depending on the size of the model and benefits from multi-core setups.

If your system has less memory and less cores, consider choosing lower values for **modelSize**.

Problems with the VM:
If there are problems when starting the VM, please check if there are any warnings from VirtualBox. We recommend reading them thoroughly and solving them as recommended by the tool.

If you get an error stating that virtualization has been deactivated for your system, you may have to activate this feature in your BIOS.
If you are using Windows, you can find out if virtualization is enabled via accessing the **Task Manager -> Performance**.



Errors during Execution:

If the log states that there were too many errors during measurement, please select all projects in your workspace and refresh them. This should make Eclipse find all missing files again. It may also help to Clean build all projects.

Errors in Workspace:

Occasionally, Eclipse generates a lot of errors that do not make sense. If a clean build (or use the black eMoflon Button in the menu bar above on your projects) does not resolve these issues, you may have to go to the **problems** view in Eclipse, select all errors and delete them via right-click->Delete. If everything worked as intended the RefactoringCDGTLFull project should have generated code in src-gen and no compile errors should be present in your workspace.