Submission for SLE 2018 Artifact Evaluation

Declarative Specification of Indentation Rules: A Tooling Perspective on Parsing and Pretty-Printing Layout-Sensitive Languages

Installation

Download the artefact (Oracle VirtualBox OVA file) from Google Drive. The file is compacted as a `tar.gz` file. Sometimes when extracting the file, the extractor creates an `.vmdk` and an `.ovf` file. In this case, import the `.ovf` file into Virtual Box using the instructions below.

Size: 3.13 GB

The artefact is distributed as a VirtualBox image. To setup the image:

- Download and install VirtualBox (These instructions were tested using VirtualBox 5.2.16): https://www.virtualbox.org
- Open VirtualBox
- Click *File* → *Import Appliance* and add the OVA file
- The VM image will appear. Go to *Settings* → *System* → *Motherboard* and set at least 10GB of RAM.
- Start the VM

The VM is running Ubuntu Server 18.04 LTS.

The user account is *artefact* and the password is *artefact*.

SSH connection

Optionally, it is possible to run the experiment using SSH to access the server in the VM. All configuration for making an SSH connection should already be in place when the VM image is imported from the OVA format.

There are several options for network settings in Virtual Box. One of the options that allows for SSH connection together with the internet connection available in the VM is using NAT + Port forwarding.

In the VirtualBox click Settings → Network → Choose Adapter 1 → switch to NAT → Expand Advanced → Port Forwarding and fill in the table such that host port *3022* will be forwarded to guest port *22*, naming the entry *ssh* and leaving the rest blank.

To SSH into the guest VM, run:

```
ssh -p 3022 artefact@127.0.0.1
```

Browsing the Experiment Contents

It is also possible to find the contents of the experiment on GitHub. Even though we provide further details for running the experiment locally (see below), we hardly advise to use the VM, as the following instructions are specific to it. The repository can be used as an alternative for file browsing and visualization.

Another option consists of using SSHFS, which mounts the VM's file system locally. See https://osxfuse.github.io/ and https://www.raspberrypi.org/documentation/remote-access/ssh/sshfs.md for more instructions.

Experiment

Below, we describe how the experiment is organized, considering the content of the paper. We also provide instructions for using the approaches described in the paper inside Spoofax (outside the VM). All directories presented in this section are relative to the directory `/home/artefact/layout-decl` in the VM.

Organization

The source code of the experiment is organized as follows:

- The folder `layout-sensitive-decl` contains the classes used to run the experiment itself: `LayoutSensitivePrettyPrint` and `LayoutSensitiveParsingPerformance`. Finally, the class `LayoutSensitiveParsingCorrectness` can be used to check the correctness of our implementation, which compares the abstract syntax trees produced by our parser against the trees produced by the previous implementation of the layout-sensitive JSGLR.
- The folders `org.spoofax.jsglr`, `org.spoofax.jsglr2`, and `org.metaborg.sdf2table` contain the implementations for the parser and parse table generator for Spoofax. The folder `org.spoofax.jsglr` contains the implementation done by [1], whereas `org.spoofax.jsglr2` contains our implementation. The folder `org.metaborg.characterclasses` and the folder `org.metaborg.tableinterfaces` contain projects that are required by the parser and/or parse table generator projects.
- The folder `pp-haskell` contains the pretty-printer implementation based on GHC, which pretty-prints Haskell programs using explicit layout.
- The folder `Haskell` contains a Spoofax project for Haskell, whereas the folder `LayoutSens` contains an example project for a small layout-sensitive language that can be used to try out our approaches inside Spoofax.

Running the Experiment

To run the experiment, execute the command `mvn -o verify -Dmain-class=<BenchmarkName>` on the top-level directory (`/home/artefact/layout-decl`), where `<BenchmarkName>` is either `org.metaborg.LayoutSensitivePrettyPrint`, `org.metaborg.LayoutSensitiveParsingPerformance`, or `org.metaborg.LayoutSensitiveParsingCorrectness`. The full experiment that tests the layout-sensitive pretty-printer takes about 16 hours to run on a single core on an Intel Core i7 @ 2,7 GHz processor host. The full experiment that evaluates the parser takes about 9 hours using the same configuration. The experiment that checks the correctness of the parser takes about 3 hours. The results are produced in the folder `layout-sensitive-decl/Results`.

Optionally, it is also possible to run a subset of the experiment, containing just 1000 random Haskell files. To run this version execute the script `./shortrun.h <BenchmarkName>`. The smaller version of the experiment for the pretty-printer, the performance and the correctness of the parser takes about 45, 30, and 7 minutes to run, respectively, using a machine with the specifications above.

We have also included the results of running the full experiment in the folder `ResultsBench`. This folder has the same structure of the resulting `layout-sensitive-decl/Results` directory, and includes the information about:

- `failedToLoad.txt`: files that failed to load by the GHC preprocessor.
- `verifyParsedFiles.txt`: files in which the experiment ran successfully.
- `verifyLog.txt`: files in which there was something wrong when comparing the abstracts syntax trees of both implementations of the layout-sensitive SGLR.
- `performance-benchmark.txt`: benchmark results for the performance of the parser.
- `pp/differentASTs.txt`: files in which the abstract syntax tree of the original program and the pretty-printed program are different.
- `pp/failedToLoad.txt`: files that failed to load by the GHC preprocessor.
- `pp/failedToParsePrettyPrinted.txt`: files in which the pretty-printed program using our approach contains a syntax error.
- `pp/failedToPrint.txt`: files that could not be pretty-printed using our approach.
- `prettyPrintedCorrectly.txt`: files that were successfully pretty-printed using our approach.

Using Layout Declarations inside Spoofax

To test the approaches described in the paper using the Spoofax Language Workbench, first download a local copy of Eclipse with Spoofax pre-installed. The eclipse installations already come with an embedded JRE, and should be downloaded according with the operating system:

- Windows 32-bits.
- Windows 64-bits.
- Linux 32-bits.
- Linux 64-bits.
- macOS.

Next, follow the steps in http://www.metaborg.org/en/latest/source/install.html, in case there are any issues running Spoofax. Import the language that is going to be tested (in the folders `Haskell` or `LayoutSens`) using File → Import → Maven → Existing Maven Project and point it to one of these folders. The syntax definition for each language can be found in the subfolder `syntax` and a few examples in the subfolder `examples`. For further information on how to use Spoofax, refer to the Spoofax documentation.

Technical Details

The following packages have already been installed on the virtual machine:

```
sudo apt-get update
sudo apt-get install openssh-server
sudo apt-get install openjdk-8-jdk
sudo apt install maven
sudo apt install ghc
sudo apt install cabal-install
```

Furthermore, we also installed the following items to the VM. First, we installed the Haskell package `happy` by running `cabal update` and `cabal install happy`, then we installed `pp-haskell` by navigating to the folder `pp-haskell` and running the command `cabal install`. Next, we added the path `/home/artefact/.cabal/bin` to the `~/.profile` file (could also have been `.bash_profile`, or `.bashrc`). We have also added Spoofax repositories to the local Maven settings file in the VM. To do that, we moved this file into the path `~/.m2/settings.xml` (if the file already exists, it is necessary to add the repositories to the current settings). Finally, we increased the memory for maven by setting the `MAVEN_OPTS` environment variable as `-Xmx8G -Xmx8G -Xss64m`.

Running the Experiment Locally

Optionally, it is possible to run the experiment locally, i.e., outside the VM. Please check the applications above, which should be installed.

To download the artefact and run the experiment locally, checkout the GitHub project https://github.com/MetaBorgCube/layout-decl. The steps to perform the experiment locally should be similar to the ones described above, which are specific to running it on the VM. The main advantage of running the experiment locally consists of the UI support to copy/modify/visualize files.

Note that the files from the benchmark are not included in the repository. Therefore to run the project locally, it is also necessary to download and extract the data used in the experiment. Further instructions can be found here.

Troubleshooting

Some files fail to parse due to technical issues in the original implementation of SGLR or the pretty-printer from GHC and may terminate the experiment prematurely. Therefore, they were manually skipped when running the experiment and included in the set of failing files in the `resources` folder (`files-do-not-parse.csv`, `files-do-not-parse-no-module.csv`). We use the same strategy to skip files that contain longest-match ambiguities (`files-contain-lm-amb.csv`).

Furthermore, when running the short version of the experiment, selecting a random set of files may imply on *slightly* different statistics, since programs that contain or not contain longest-match ambiguities may affect the final performance of the parser.

Bibliography

[1] Sebastian Erdweg, Tillmann Rendel, Christian Kästner, and Klaus Ostermann. Layout-Sensitive Generalized Parsing. In *Software Language Engineering, 5th International Conference, SLE 2012*.