OOPSLA 2021 Artifact

Promises Are Made To Be Broken

This artifact is composed of two parts. The results in Section 6 of the paper are reproduced by the steps in the Robustness section and the results in Section 5 are reproduced by the Performance section.

Robustness

Getting Started

The artifact is a Docker image. It generates & validates strictness signatures by dynamic analysis of packages written in R. Generated HTML reports contain the graphs & data from our paper.
The report is served by the Docker container on `localhost:8000`.

The Docker image has been tested on Linux using Docker version 20.10.5.

1. Download

Setup Docker on Windows: https://docs.docker.com/docker-for-windows/ and Mac: https://docs.docker.com/docker-for-mac/. On Linux, use the package manager.

After installation:

- Start Docker daemon (`sudo systemctl start docker`)
- Add user to `docker` group to run Docker commands (`sudo addgroup $USER docker`)

It is recommended to *restart the machine.*

Pull the artifact from https://hub.docker.com/r/aviralgoel/oopsla-2021-r-promisebreaker with:

```
docker pull aviralgoel/oopsla-2021-r-promisebreaker:latest
```

The artifact is a 2.75 GB compressed image. Docker should show the following sha256 digest:
`d02f7a246d5ad6a8b5836a8f7a4666c94c26076c8691218d09cd69c97489fb2b`

You can build the image locally by navigating to `promisebreaker-artifact/robustness` and running `make build`.

2. Start

To start the image:

```
docker run --rm -p 8000:8080 -ti aviralgoel/oopsla-2021-r-promisebreaker:latest
```

This command creates a minimal Linux distro customized to run the artifact, and opens bash.

- The `-p` option is used to describe port mapping.
  Details are here: https://docs.docker.com/config/containers/container-networking/
  The container starts an `nginx` web server on port `8080` serving files from `/var/www/`
  The container port `8080` is mapped to the system port `8000`. You can navigate to `localhost:8000` in your browser to see an index of the container's `/var/www/` directory. It will contain the artifact's actual and expected output.
- The `-ti` options allocates a `tty` for the container.
  Details are here: https://docs.docker.com/engine/reference/run/

The container runs Debian Buster. The user and pwd are `aviral`. For convenience, user `aviral` has passwordless `sudo` privileges.

`Emacs` and `Vim` are included for viewing code. `zsh` and `fish` are installed as `bash` alternatives.

3. Run

To run the artifact, type in the shell:

```
cd ~/promisebreaker-experiment
make synthesize-signatures
make install-signatures
make validate-signatures
make report-results
```

These make commands run the pipeline for synthesizing and validating signatures. Each step is composed of multiple substeps that print copious amounts of progress information. The commands take 38 mins (11m +33s + 26m + 44s) on 4 CPUs on a Linux machine.

The first make command extracts executable programs from packages listed in:

`experiment/profile/trace/index/corpus-small`

it performs dynamic analysis to generate execution traces, and analyzes the data to generate strictness signatures for multiple configurations in the directory:

`experiment/profile/signatures`

The second make command copies these signatures in the `strictr` package directory:

`dependency/strictr/inst/signatures/`

and installs the `strictr` package.

The third make command runs programs extracted from client packages listed in:

`experiment/validate/trace/index/client-small`

by applying the strictness signatures installed by the previous command.

The fourth make command generates HTML reports from RMarkdown notebooks which analyze the data to produce graphs/numbers found in the paper. You can view the reports in your browser by navigating to:

`localhost:8000/`

The main report of interest is validation.html which should be compared against small/validation.html. This file contains the table of final results presented in Section 6 -- “Robustness of Inferred Signatures”. The last table has the result of running programs in strict mode. The `total` column contains the number (and percentage) of programs that failed.. This table should match across the two files with minor differences owing to non-determinism and timeouts. The lower the number, the better. For most packages, we expect this number to be 0 because overall we have very few failures.

	order	signature	total
1	0	+U+E+R	0&0\%
2	1	+U+E-R	0&0\%
3	2	+U-E+R	0&0\%
4	3	+U-E-R	0&0\%
5	4	-U+E+R	0&0\%
6	5	-U+E-R	0&0\%
7	6	-U-E+R	0&0\%
8	7	-U-E-R	0&0\%

To minimize the time taken in this step, we have chosen a single corpus and client packages. The single corpus package provides 17 executable programs for synthesis and the single client package provides 118 client programs for validation. We next repeat these steps for a bigger corpus.

Step by Step Instructions

Our artifact is CPU and IO-intensive. The results in the paper were obtained from synthesizing signatures for 500 R packages and validating them on 2000 client packages. The complete experiment generated 1.5 TB of data and took 5 days on a 256 GB RAM, 72 core server. A complete Docker image containing all R packages and their dependencies occupies over 150 GB of disk space and takes a day to build on this server. To keep the build size and time manageable the Docker image provided has 279 R packages installed. The default list of corpus and client packages has been carefully selected to aid evaluation in a reasonable time.

To run the larger version of the artifact (on 4 corpus and client packages), start a fresh Docker container (using the command provided earlier) and run the following commands:

```
cd ~/promisebreaker-experiment
make synthesize-signatures CORPUS_FILENAME=corpus-large
make install-signatures
make validate-signatures CLIENT_FILENAME=client-large
make report-results
```

The file `corpus-large` present at `experiment/profile/trace/index/` contains 4 packages which provide 204 synthesis programs.
The file `client-large` present at `experiment/validate/trace/index/` contains 4 packages which provide 474 client programs.

These commands can take a long time to execute. On a 72 core 256 GB RAM Linux server, they take 30 minutes.

You can view the reports in your browser by navigating to:

`localhost:8000/`

This time, compare the top-level HTML reports against the corresponding ones in the `large` directory. The key results are in `validation.html`. The remaining reports contain data points and tables for the quantitative analysis presented in the paper in the earlier sections. The numbers in these reports will not match the ones in the paper because the data presented in the paper is from 500 packages.

1. Organization

In this section, we will describe the structure of the artifact and the outputs generated by the intermediate steps.

The artifact relies on multiple projects checked out in the `~/promisebreaker-experiment/dependency` directory.

The repositories are described below.

`R-dyntrace`	Modified R VM with probes
`R-vanilla`	Unmodified R VM
`instrumentr`	Dynamic analysis framework for R
`lazr`	Dynamic analyzer for laziness
`strictr`	Enforce strict evaluation semantics
`experimentr`	Package for creating analysis pipelines

The `~/promisebreaker-experiment/experiment` directory consists of three subdirectories.

1. Corpus

The `corpus` subdirectory consists of executable programs extracted from the preinstalled R packages. It has the following structure

```
experiment/corpus/extract/programs/
├── <type>
│   ├── <package>
│   │   ├── <filename>.R
...
```

Here `<type>` is one of `example`, `test`, `testthat`, or `vignette`, `<package>` is the package name, and `<filename>` is the extracted program.

2. Profile

The `profile` directory contains the results of the signature synthesis stage.

The `profile/trace` subdirectory contains the execution traces generated by running lazr on the corpus package programs. It has the following layout.

```
experiment/profile/trace/programs/
├── <type>
│   ├── <package>
│   │   ├── <filename>
│   │   │   ├── output
│   │   │   │   ├── arg_ref.fst
│   │   │   │   ├── arguments.fst
│   │   │   │   ├── call_ref.fst
│   │   │   │   ├── calls.fst
│   │   │   │   ├── effects.fst
│   │   │   │   ├── environments.fst
│   │   │   │   ├── functions.fst
│   │   │   │   └── metaprogramming.fst
│   │   │   ├── program.R
│   │   │   ├── seq
│   │   │   ├── statistics
│   │   │   │   ├── allocation.fst
│   │   │   │   └── execution.fst
│   │   │   ├── stderr
│   │   │   └── stdout
...
```

The execution traces for `<filename>` of `<type>` from `<package>` is present in the form of compressed binary tabular files in the `<type>/<package>/<filename>/output` directory.

The `<type>/<package>/<filename>/statistics` directory contains general program statistics such as number of allocations and time taken for execution.

The `<type>/<package>/<filename>/program.R` is the program whose run generated this data. It is wrapped in a call to the tracer to do the data collection.

The `<type>/<package>/<filename>/{stderr, stdout}` files contain the stdout and stderr output of the program.

The `<type>/<package>/<filename>/seq` file contains the sequence assigned to this program run by GNU parallel used for parallelizing the data collection run.

The execution traces are analyzed by a map-reduce style analysis pipeline with three stages, `reduce`, `combine` and `summarize`.

The reduced version of
`profile/trace/programs/<type>/<package>/<filename>/output`
is present in the directory
`Profile/trace/programs/<type>/<package>/<filename>/reduce`

The combined version of reduced data is present in the directory
`profile/combine`

The summarized version of combined data is present in the directory
`profile/summarize`

This summarized version is used to generate the strictness signatures present in the directory
`profile/signature`

This directory has the layout:

```
profile/signature/
├── <config>
│   ├── <package>
...
```

Here, <config> is the signature configuration and <package> file contains its functions' strictness signatures.

3. Validate

The validate/trace directory contains the output from running programs after applying the signatures of different configurations. It has the following layout.

```
validate/trace/programs
    ├── <type>
    │   ├── <package>
    │   │   ├── <filename>
    │   │   │   ├──<config>
    │   │   │   │   ├── application.fst
    │   │   │   │   ├── seq
    │   │   │   │   ├── stderr
    │   │   │   │   ├── stdout
    │   │   │   │   └── strictr-log
```

The key files of interest are
`validate/trace/programs/<type>/<package>/<filename>/<config>/{stderr, stdout}`

These files contain the output generated by running the programs in strict mode.

The validate/summary directory contains summaries of the results obtained from the validation run.

Lastly, the output of experiment/`profile/summarize and experiment`/validate/summary is copied to the paper/data repository and reports are generated from paper/*.Rmd files

2. Customization

To synthesize and validate signatures for packages of your own choice, you can add them to the `corpus-large` and `client-large` files and run the previous commands again. Make sure to perform the run in a fresh container. As expected, the result of validation will be found at `localhost:8000/validation.html`

It is important to ensure that the custom packages specified in the corpus and client files are installed. The list of currently installed packages can be found in the directory:

```
~/promisebreaker-experiment/dependency/library/install
```

We have preinstalled 279 packages. For the actual experiment we installed over 19,000 R packages from CRAN and BIOCONDUCTOR and chose the top packages from those.

When you specify custom packages, make sure to avoid using R.oo, R.utils, dplyr, vctrs, ggplot2, tidyverse, and future packages. The system supports these packages but they either take a very long time and generate too much data or, they are responsible for all the validation failures. This means, to get the same validation rate as the paper, you will have to use all the 500 corpus packages and 2000 client packages which can take days.

Make sure to initiate the custom run in a fresh Docker container.

Errors and Warnings

The following errors and warnings can be safely ignored.

```
make[1]: *** [Makefile2:337: experiment-profile-reduce] Error 9
```

```
Warning: `recursive` is deprecated, please use `recurse` instead
Warning: `recursive` is deprecated, please use `recurse` instead
```

```
Warning messages:
1: In bind_rows_(x, .id) :
 Vectorizing 'fs_path' elements may not preserve their attributes
2: In bind_rows_(x, .id) :
 Vectorizing 'fs_path' elements may not preserve their attributes
3: In bind_rows_(x, .id) :
 Vectorizing 'fs_path' elements may not preserve their attributes
4: In bind_rows_(x, .id) :
 Vectorizing 'fs_path' elements may not preserve their attributes
5: In bind_rows_(x, .id) :
 Vectorizing 'fs_path' elements may not preserve their attributes
6: In bind_rows_(x, .id) :
 Vectorizing 'fs_path' elements may not preserve their attributes
7: In bind_rows_(x, .id) :
 Vectorizing 'fs_path' elements may not preserve their attributes
```

```
System has not been booted with systemd as init system (PID 1). Can't operate.
Failed to create bus connection: Host is down
```

```
Warning message:
In system("timedatectl", intern = TRUE) :
  running command 'timedatectl' had status 1
```

Performance

This section describes the artifact that reproduces the experiments for Section 5 (figures 3, 4, and 5) in the paper. The artifact is a Docker image. It has been tested on Linux using Docker version 20.10.5.

1. Download

Download and install Docker as described in previous section

Pull the artifact from https://hub.docker.com/r/sebastiankrynski/rsh-artifact with:

```
docker pull sebastiankrynski/rsh-artifact:latest
```

The artifact is a 3.61 GB compressed image. Docker should show the following sha256 digest:

```
eabecffea2d26de164b1b467f30409688158c9c7ac2e8ee0f7affb061693ab51
```

You can build the image locally by navigating to:

```
promisebreaker-artifact/Rsh-container
```

and running `make build`.

2. Run

To start the image:

```
docker run sebastiankrynski/rsh-artifact:latest
```

After the Docker image launches, it automatically checks out the appropriate versions of the code for each experiment, compiles the source code, performs the experiments, and generates graphs from the experiments' output into a pdf file. These steps are orchestrated by a shell script, `run.sh`, which can be found at the top-level in the Docker container. This script can take many hours to finish. On our 256 GB RAM, 72 core Linux server, it takes 10 hours.

After the script finishes executing, the graphs can be found in `/paper/main.pdf`. This file can be compared against the results presented in the paper in Section 5. Since experiments 1 and 2 measure performance, they were run in a dedicated benchmarking machine for the results presented in the paper. Hence, the numbers obtained from the artifact will differ from the ones presented in the paper.

The following message can be safely ignored.

```
GNU VM dir does not exist: beware that if you want to run benchmarks for GNU-R it
will fail
FASTR dir does not exist: beware that if you want to run benchmarks for FAST-R it
will fail
Minimizing noise with rebench-denoise failed
    possibly causing benchmark results to vary more.
```

3. Details

In this section, we describe the organization of the artifact and how the experiments are performed. The artifact consists of three repositories:

1. **Ř**: Main Ř repository in /opt/rir/
2. **gnur**: GNU R implementation that is slightly modified to work with Ř.

   It is located at /opt/rir/external/custom-r.

3. **rbenchmarks**: Performance benchmark suite for the compiler.

   It is located at /opt/rbenchmarking

Benchmarks are run using the benchmarking framework, Rebench.

You can study the run.sh script to see how it executes the experiments.
When it finishes execution, it creates the following data files:

```
/paper/data/file<XX>.data
/paper/data/1174869738.data
/paper/data/1174881563.data
/paper/data/1172663147.data
/paper/data/1172677834.data
/paper/data/gnur-promises.csv
/paper/data/rsh-promises.csv
/paper/data/rsh-strict-promises.csv
```

These files are then analyzed to produce the `/paper/main.pdf` file with the graphs.