Signatr Artifact

We also provide pdf and html versions of this README. If reading locally and not on github, we advise to use the pdf or html version.

The artifact contains the `signatr` tool, and the pipelines to create an R value database and to fuzz R functions with the database to find type signatures. The pipeline to create a valeu database is in `pipeline-dbgen`. The fuzzing pipeline will generate the inputs for the `sle.Rmd` R markdown notebook. That notebook can then be rendered to get all the results (tables, figures) we use in the paper.

To use the artifact to reproduce the paper results, follow the steps:

1. Install the docker image (see Install the docker image). Installing locally is possible but involved. Following the steps described in the `docker-image/Dockerfile` should help if this is the hard path you are choosing!
2. Generate a database (see Generate the database) or use an already-uploaded one (See Use an uploaded database).
3. Fuzz (see Fuzzing)
4. Render the notebook with the paper results (see Rendering the paper results)

You can also the artifact to build a custom database and fuzz the signatures you want to in [Experimenting the tool])#experimenting-with-the-tool).

Tool

The tool is packaged as an R library. It is hosted at https://github.com/PRL-PRG/signatr and uses the following building blocks:

- sxpdb: R value database
- generatr: fuzzing utilities
- contractr:type signature parsing and checking for R
- argtracer: trace R values using a patched R interpreter and store them in the R value database.

The tool and its dependencies are pre-installed in a convenient Docker image.

Install the docker image

You can:

- pull the docker image with `docker pull prlprg/sle22-signatr`, or
- build the docker image (it takes time!):

```
cd docker-image
make
```

After installing the docker image, *make sure* to run all the following commands in a shell inside the docker image (for Linux, macOS) from the artifact directory:

```
docker run --rm -ti -v $(pwd):/work -e USER_ID=$(id -u) -e GROUP_ID=$(id -g) -w /work prlprg
```

Experimenting with the tool

Run the R interpreter *inside the docker image*. It will start the patched R interpreter. The tool *does not run* in the standard R interpreter.

In the following listings, `$` indicates the shell and `>` denotes the R REPL.

```
R version 4.0.2 (2020-06-22) -- "Taking Off again"
...

> library(signatr)
```

Database

To generate a database of values, we need some code to run. One way is to extract it from an existing R package, for example `stringr`, which provides regexes:

```
> extract_package_code("stringr", output_dir = "demo")
...
7 examples/str_detect.Rd.R examples
...
```

This will extract all the runnable snippets from the package documentation and tests into the given directory. For example:

```
$ cat demo/examples/str_detect.Rd.R
...
fruit <- c("apple", "banana", "pear", "pinapple")
str_detect(fruit, "a")
str_detect(fruit, "^a")
...
```

Next, we trace the file by running it (in the patched R interpreter) and recording all the calls, using the `trace_file`function:

```
trace_file("demo/examples/str_detect.Rd.R", db_path = "demo.sxpdb")

        status time                          file    db_path db_size error
elapsed      0 0.04 demo/examples/str_detect.Rd.R demo.sxpdb      20    NA
```

The database generation is also automated in the `pipeline-dbgen` directory in the artifact, and handles there tracing on multiple files and merging the results. See Generate the database for more details.

Fuzzing

Once the database is ready, we can start fuzzing the `str_detect` function of the `stringr` package:

```
R <- quick_fuzz("stringr", "str_detect", "demo.sxpdb", budget = 100, action = "infer")
```

```
    started a new runner:PROCESS 'R', running, pid 4157
    fuzzing stringr:::str_detect [======] 100/100 (100%) 39s
    stopped runner:PROCESS 'R', running, pid 4157
```

The `infer` action will infer types for each call argument and return value using the type annotation language supported by `contractr`. It returns an R data frame with the inferred call signature in the `result` column:

```
> print(R)
# A tibble: 100 x 6
args_idx      error                 status result          time
<list>        <chr>                 <int>  <chr>           <drtn>
1 <int [3]> "Error in UseMeth...   1       NA              0.0363
2 <int [3]>  NA                    0     (character[],... 0.0351
```

If you are repeating these steps, it is possible that your results will be different since fuzzing is non-deterministic.

The listing shows two calls: a failed one (non-zero status) with an error message, and a successful one with an inferred signature. The `args_idx` column contains the indices of the values of the arguments in the database: the actual argument values can be obtained by looking up the `args_idx` in the database:

```
> library(sxpdb)
> db <- open_db("demo.sxpdb")
> get_val_idx(db, 0) # value at index 0
[1] "a"
> close(db)
```

One advantage of using R is that we can use R's many data analysis functions. For example, we can look at the resulting signatures:

```
> count(R, result)
# A tibble: 4 x 2
   result                                                         n
   <chr>                                                       <int>
1 (character[], ^character[], double) => ^logical[]             1
2 (character[], character, integer) => logical[]                1
3 (list<integer>, character[], list<integer>) => logical[]      1
4 NA                                                            97
```

This shows that in 3 cases, the fuzzer managed to generate a call that was successful, and so the signatures of those calls.

Use an uploaded database

Databases are huge, several hundreds of GB for 400 packages, so we provide a link to download it.

Generate the database

The database generation uses targets to orchestrate the pipeline.

The database for the SLE paper is obtained by tracing 400 packages in `data/packages.txt`.

To start tracing, after opening an R session and specifying an adequate number of parallel workers:

```
cd pipeline-dbgen
targets::tar_make_future(workers = 64)
```

The extracted code of the packages will be in `data/extracted-code`. The resulting database will be generated as `data/sxpdb/cran_db`. It will also output a call id companion file in `data/callids.csv`. Depending on your machine, the generation of the database for the 400 packages can take from a few hours to a few days.

You can change `packages.txt` to include less packages. For instance, `packages-4.txt` includes 2 huge and common R packages, `dplyr` and `ggplot2`. We provide pre-extracted code for a few packages already, including `stringr`, `dplyr`, and `ggplot2`.

Fuzzing

```
./run-fuzz
```

Rendering the paper results

We just have to render the RMkardown file. It will output an `experiment-uf.tex` file with macros for all the experimental values in the paper, and a pdf file (`uf-call-signatures.pdf`) for Figure 4 in the paper.

```
R -e 'rmarkdown::render("sle.Rmd")'
```