Artifact of 'FLACK: Counterexample-Guided Fault Localization for Alloy Models'

Guolong Zheng*, ThanhVu Nguyen*, Simón Gutiérrez Brida†, Germán Regis†,
Marcelo F. Frias‡, Nazareno Aguirre†, Hamid Bagheri*
*Univeristy of Nebraska-Lincoln
{gzheng, tnguyen}@cse.unl.edu, bagheri@unl.edu
†University of Rio Cuarto and CONICET
{sgutierrez, gregis, naguirre}@dc.exa.unrc.edu.ar
‡Dept. of Software Engineering Instituto Tecnológico de Buenos Aires
mfrias@itba.edu.ar

***Abstract*—This document provides instructions to setup and execute FLACK. FLACK is an automatic fault localization tool for Alloy. Given an Alloy model with violated assertions, FLACK automatically outputs a list of expressions ranking based on their suspiciousness to the error. The link to the replication package is https://github.com/guolong-zheng/flack-ae. The replication package contains the source code of FLACK and benchmarks to reproduce all the evaluation results in the ICSE 2021 submission.**

***Index Terms*—Alloy, fault localization**

I. Introduction

This document provides instructions to setup and execute FLACK. FLACK is an automatic fault localization tool for Alloy. Given an Alloy model with violated assertions, FLACK automatically outputs a list of expressions ranking based on their suspiciousness to the error.

II. Envrionments

FLACK is implemented in Java 8 and uses Alloy 4.2. It has been tested on Ubuntu 16.04 and macOS. We also provide a docker image for ease of use.

III. Execution Instructions

A. Obtain the Artifact

The artifact can be downloaded at DOI. The artifact is downloaded as a tar.gz file named **flack.tar.gz**. The user can use the command `tar -xvf flack.tar.gz` to unzip.

B. Inventory of the Artifact

- flack\
  - benchmark\
    - alloyfl\: Alloy models from AlloyFL
    - large\: large real-world Alloy models
  - src\main
    - alloy\: extended Alloy with Pardinus
    - finder\: source code of FLACK
  - libs: sat solver jars
  - solvers: sat solver dynamic link libraries
  - AlloyFL: replication package of AlloyFL

IV. Execution Instructions

In this section, we provide two ways to run FLACK: run in docker and build from source.

A. Run in Docker

1) Dependencies: Docker(https://www.docker.com/)

2) Instructions:

i In directory `flack/`, use `docker build -t flack .` to build the docker image

ii Use `docker run -it flack` to start the docker image

iii Generate TABLE III use `flack icse21`, the whole process takes aoubt one or two minutes, and result will be written to result.cvs (please note that the ranking information can not be generated automatically and has to be checked manually. Due to the randomness of Alloy, the numbers may be slightly different)

iv Generate TABLE IV use `flack loc -f bnechmark/large/ -m 5`, this process takes about 30 minutes

v Run FLACK on a single model use `flack loc -f path/to/model -m #/of/instances`. The assertion to be checked in the model need to be rename to repair_assert_#, where # can be an arbitrary number.

B. Build from Source

1) Dependencies:

- Java 8
- Maven(https://maven.apache.org/)
- bash

2) Instructions:

i In directory `flack\`, build the project use `mvn clean package`, this will produce jar files in the `target\` directory

ii Generate TABLE III use:

```
java -Djava.library.path=solvers
    -cp ./libs/*:./target/flack
    -1.0-jar-with-dependencies.
    jar loc icse21
```

this will finish in one or two minutes

iii Generate TABLE IV use:

```
java -Djava.library.path=solvers
    -cp ./libs/*:./target/flack
    -1.0-jar-with-dependencies.
    jar loc -f benchmark/large/ -
    m 5
```

this may take about 30 minutes

iv In directory flack/, run FLACK on a single model use

```
java -Djava.library.path=solvers
    -cp ./libs/*:./target/flack
    -1.0-jar-with-dependencies.
    jar loc -f /path/to/model –m
    #/of/instances
```

C. TABLE V: Compare with AlloyFL

For more details, please check AlloyFL paper and its git repo(https://github.com/kaiyuanw/AlloyFLCore). We also include a distribution in our package, to run AlloyFL on the benchmark:

i Go to flack\AlloyFL\

ii Build AlloyFL use `mvn clean package`

iii Run AlloyFL on all models use

```
java -Djava.library.path=sat-
    solvers -cp lib/*:target/
    aparser-1.0.jar alloyfl.hybrid.
    HybridAverageFaultLocator
```

the total runtime for AlloyFL takes about 80 minutes

D. Illustrative Example

Use `flack loc -f benchmark/alloyfl/addr.als -m 5` to run FLACK on `addr.als` with 5 pairs of instances, if successfully installed, the output will be:

```
/flack/benchmark/alloyfl/addr.als:
example generation time:0.525
RANK LIST:
0: lone ((n . (b . listed))) 1.31
1: n in lookup[b,n] 1.30
2: (n . ^((b . listed))) 1.30
3: l in lookup[b,n] 1.30
4: !(n in lookup[b,n]) 1.30
5: l in (b . entry) 1.17
6: l in lookup[b,n] => l in (b . entry) 1.00
-----------------------
analyze time(sec): 0.84
# rel: 1
# val: 3
# Slice Out: 10
# Total AST: 74
LOC: 21
evals: 368 | node: 6
====================
```

E. Explaination of Output

Output	Explaination
RANK LIST	The ranking list of suspicious expressions
analyze time	Total runtime
# rel	Number of different relations
# val	Number of different values
# Slice out	Number of AST nodes sliced out
# Total AST	Number of total AST nodes in the model
LOC	Line of Code of the model
evals	Number of instantiated expressions