Artifact Guide

FuzzDP is a Haskell package that provides an embedded programming language with language primitives for differential privacy, and an automatic testing framework that checks differential privacy properties of languages written in the embedded language.

Evaluating FuzzDP with Docker

We have packaged FuzzDP and its external dependency `z3` into the provided docker image. We also provide both `vim` and `emacs` text editors in the docker image for modifying and experimenting with programs in the docker image.

1. Install Docker following the official guide
2. Download the image (size: ~11G) here
3. Start the docker daemon. Docker will ask for your host system credential on first time startup, and it may also show a login UI for dockerhub. However, you do *not* need to login for the following steps
4. Run `docker image load -i fuzz-dp-artifact.tar`, this may take a while to complete (around 20 minutes on a 4.0GHz quad-core CPU)
5. Run `docker images`, and verify it shows an image with `REPOSITORY fuzz-dp`
6. Run `docker run --rm -it fuzz-dp` to start a docker shell

Step-by-step guide

FuzzDP's build system is managed by the Haskell build tool `stack`. We use `stack` to build, and also run the entire benchmark algorithm test suite.

To start, run `stack test` in the docker shell. This will run the entire test suite for all benchmark algorithms listed in the paper. The entire process takes around 70 minutes to complete on a 4.0GHz quad-core CPU (using only 1 core).

We also provide a detailed guide on writing your own differentiall private program in FuzzDP and testing it at the detailed guide below.

Claims supported by this artifact

Here is a table that matches the benchmark test definitions with the FuzzDP benchmark results table in the evaluation section.

Test name	File:Line Number	Evaluation	Correct	Buggy
prop_simpleCountIsDifferentiallyPrivate	`test/Spec.hs:522`	nc	x	
prop_simpleCountEpsTooSmallIsNotDifferentiallyPrivate	`test/Spec.hs:526`	nc		x
prop_simpleMeanIsDifferentiallyPrivate	`test/Spec.hs:530`	nm	x	
prop_unboundedMeanIsNotDifferentiallyPrivate	`test/Spec.hs:534`	nm		x
simpleSumBuggyNotPrivateTest	`test/Spec.hs:108`	ns		x
prop_prefixSumIsDifferentiallyPrivate	`test/Spec.hs:458`	ps	x	
prefixSumBuggyNotPrivateTest	`test/Spec.hs:96`	ps		x
prop_privTreeIsDifferentiallyPrivate	`test/Spec.hs:502`	pt	x	
prop_privTreeBuggyIsNotDifferentiallyPrivate	`test/Spec.hs:506`	pt		x
prop_rnmIsDifferentiallyPrivate	`test/Spec.hs:435`	rnm	x	
prop_rnmBuggyIsNotDifferentiallyPrivate	`test/Spec.hs:447`	rnm		x
prop_smartSumIsDifferentiallyPrivate	`test/Spec.hs:454`	ss	x	
prop_smartSumBuggyIsNotDifferentiallyPrivate	`test/Spec.hs:462`	ss		x
prop_sparseVectorIsDifferentiallyPrivate	`test/Spec.hs:466`	sv	x	

Test name	File:Line Number	Evaluation	Correct	Buggy
prop_sparseVectorBuggyIsNotDifferentiallyPrivate	`test/Spec.hs:482`	sv		x
prop_sparseVectorBuggy4IsNotDifferentiallyPrivate	`test/Spec.hs:490`	sv		x
prop_sparseVectorBuggy5IsNotDifferentiallyPrivate	`test/Spec.hs:494`	sv		x
prop_sparseVectorBuggy6IsNotDifferentiallyPrivate	`test/Spec.hs:498`	sv		x
prop_sparseVectorGapIsDifferentiallyPrivate	`test/Spec.hs:514`	svGap	x	
sparseVectorGapBuggyNotPrivateTest	`test/Spec.hs:306`	svGap		x

Claims not supported by this artifact

The statistical tests we perform on the two versions of DAS are not released in this artifact archive: because 1) these tests require a cluster, lots of disk space, and the 1940s Census Data to run, and 2) we are not allowed to release the 1940s Census Data along with this software package.

Detailed guide on writing your own program and testing it

We will walk through an example of implementing a simple noisy count algorithm in FuzzDP. The algorithm can be motivated by the following scenario:

A teacher is grading an exam for a group of students, and also releasing statistics on the number of students that passed/failed the exam. Suppose there were 50 students in total, and 49 of them took the exam as scheduled, among which 39 passed, and 10 failed, and the teacher published the following statistic:

Pass	Fail
39	10

The one student who did not take the exam as scheduled took the exam at a later time. Now, suppose the teacher updates the statistic like the following:

Pass	Fail
39	11

Then, just by comparing the published statistics, we can deduce the student had failed the exam, even though the teacher did not publish any identifiable information about any student.

A better alternative is to publish these statistics by adding a certain amount of noise, so that this process is differentially private.

The easiest way to reproduce the following steps is to first launch a ghci Haskell interpreter session by:

1. multiplex the docker shell with `tmux` or `screen` (both already installed in the docker image)
2. in one of the sessions, launch a Haskell interpreter `ghci` instance with `stack ghci` under the fuzz-dp project directory
3. in another session, edit the file `src/Data/Fuzzi/Examples.hs`
4. to load the edited content into the interpreter, type the command `:r` in the `ghci` session

Note that the `src/Data/Fuzzi/Examples.hs` already contains all the definitions introduced below.

We model this problem using FuzzDP. First, we define the criteria of passing the exam. Let's say the passing grade is 60/100.

```
-- In src/Data/Fuzzi/Examples.hs
passOrFail :: forall real bool.
  ( FuzziType real
  , FracNumeric real
  , CmpResult real ~ bool
  ) => Fuzzi real -> Fuzzi bool
passOrFail score = score %>= 60.0
```

Next, we define a function that counts how many passing scores there are in an input list of scores, and adds randomly sampled noise to the count.

We need to decide how much noise to add. This step requires some analysis of how adding/removing a single score from the input may influence the exact count of number of passing scores. In this case, adding/removing a single score can change the exact count by up to 1. This factor is known as the "sensitivity" of a private value.

If a private value has sensitivity `s`, and we add noise sampled from the laplace distribution with width `w` to it, then the resulting algorithm is `s/w`-differentially private. In other words, the privacy parameter epsilon for such a procedure is `s/w`. In this example, we choose the width to be `1.0`, so that `countPassedDP` is a 1-differential private program. We will use FuzzDP's testing combinators to check this property.

```
-- In src/Data/Fuzzi/Examples.hs
countPassedDP :: forall m real.
    FuzziLang m real => [Fuzzi real] -> Mon m (Fuzzi real)
countPassedDP []     = lap 0 1.0
countPassedDP (x:xs) = do
  ifM (passOrFail x)
      (do
          tailCount <- countPassedDP xs
          return (1.0 + tailCount))
      (countPassedDP xs)
```

These functions have elaborate type signatures and typeclass constraints. These constraints enable us generalize the same piece of code for the two kinds of interpretation that FuzzDP uses: concrete interpretation, and symbolic interpretation. This flexibility comes at the cost of moderately complex typeclass based abstractions. More details about these types and typeclasses can be found in FuzzDP's documentation, which we also provide along with the docker image. Please see the section at the bottom on how to access the documentation.

We can concretely evaluate this function (on an artificial input that contains 30 perfect scores, and 20 scores at 50) in the `ghci` session by typing the command

```
> :r
> sampleConcrete $ eval (reify $ countPassedDP $ take 30 (repeat 100) ++ take 20 (repeat 50))
30.89295347209235
> sampleConcrete $ eval (reify $ countPassedDP $ take 30 (repeat 100) ++ take 20 (repeat 50))
31.346492003757792
> sampleConcrete $ eval (reify $ countPassedDP $ take 30 (repeat 100) ++ take 20 (repeat 50))
29.420243650890267
```

Here, we first pass the artificial input to `countPassedDP`, and then we construct a concrete program abstract syntax tree by reifying the shallowly embedded program with the function `reify`. We then pass the resulting AST to the concrete interpreter `eval`. Since the result is a distribution object, we sample from this distribution object with `sampleConcrete`.

Next, let's run some tests. First, we need to write down the expected differential privacy property as a Haskell function. FuzzDP uses the property testing framework QuickCheck to express such properties. In this case, our property is parameterized by a pair of similar inputs, which we represent with the type `BagList Double`.

```
-- In src/Data/Fuzzi/Examples.hs
countPassedDPPrivacyTest :: BagList Double -> Property
countPassedDPPrivacyTest xs =
  monadicIO $
    expectDP -- replace with `expectDPVerbose` for logging
      1.0    -- 1.0-differentially private
      500    -- run test with 500 sampled traces
      ( reify . countPassedDP . map realToFrac $ left xs
      , reify . countPassedDP . map realToFrac $ right xs
      )
```

Here, the type `BagList` is exported from `Data.Fuzzi.NeighborGen`, a module in FuzzDP that implements several generators of test data that satisfies common similarity relations. `BagList` means the pair of similar inputs generated satisfy a kind of relation called "bag distance". In particular, we assume the inputs have bag distance = 1 here, because at most one score needs to be removed/added to the input list to make the two input lists contain the same set of data (up to reordering).

We use the test combinator `expectDP` exported from `Data.Fuzzi.Test` to assert that the program is expected to be `1.0` differentially private. The parameter `500` asks the testing framework to use 500 sampled concrete execution traces to generate SMT formulas that will be checked by Z3 as evidence of differential privacy. The `realToFrac` function is a Haskell's standard library that performs numeric type casting. In this example, the two calls to `realToFrac` convert the input `Double` values into values suitable for concrete and symbolic execution, respectively.

Finally, the last argument tuple passed to `expectDP` are `countPassedDP` applied to the `left` and `right` projections of the pair of similar bag list inputs.

We can run this test in the `ghci` session by running the command. The `bagListSmall` function comes from `Data.Fuzzi.NeighborGen`, and is a utility function that defines a generator for small lists. Here, we ask the generator to produce small lists whose values are between `40` and `80`, and lists are considered similar if at most `1` element needs to be removed/added to make the two identical.

```
> :r
> quickCheckWith stdArgs{maxSuccess = 20} $ forAll (bagListSmall (40, 80) 1) countPassedDPPrivacyTest
[Ok 1.6653345369377348e-16]
[Ok 1.8041124150158794e-16]
[Ok 0.9999999999999998]
[Ok 0.9999999999999999]
[Ok 3.469446951953614e-18]
[Ok 1.91224152101474e-18]
[Ok 5.002217949828752e-7]
[Ok 1.0]
[Ok 0.9999994999999998]
[Ok 2.500000002220446e-7]
[Ok 5.722198694046998e-17]
[Ok 0.9999999999999999]
[Ok 2.483632295657845e-16]
[Ok 7.667369328650443e-17]
[Ok 1.1027877500069394e-17]
[Ok 1.0]
[Ok 1.2263329207909403e-16]
[Ok 2.50000000194289e-7]
[Ok 6.552397513459596e-18]
[Ok 5.008958e-7]
+++ OK, passed 20 tests.
```

This command kicks off the testing process, asking the testing framework to generate 20 random pairs of similar inputs, and checks that the differential privacy test succeeds on all generated inputs. The printed numbers (`Ok 1.0`, etc.) are the empirical privacy cost observed by solving the SMT fomulas passed to Z3, we observe that they are all at most `1.0` ---the expected epsilon privacy parameter of the program under test.

We can then modify the `countPassedDP` program to be faulty, for example, by making it use less noise than currently designed. Let's change the source code so that we now sample from a laplace distribution with width `0.1` instead of `1.0`.

```
-- In src/Data/Fuzzi/Examples.hs
countPassedDP :: forall m real.
    FuzziLang m real => [Fuzzi real] -> Mon m (Fuzzi real)
countPassedDP []     = lap 0 0.1 -- width parameter changed here
countPassedDP (x:xs) = do
  ifM (passOrFail x)
      (do
          tailCount <- countPassedDP xs
          return (1.0 + tailCount))
      (countPassedDP xs)
```

Now, if we run the same tests again, we observe the testing framework successfully reports a privacy violation:

```
> :r
> quickCheckWith stdArgs{maxSuccess = 20} $ forAll (bagListSmall (40, 80) 1) countPassedDPPrivacyTest
[FailedUnSat [
  "|eps >= abs(0 % 1 + shift - 0 % 1) / 3602879701896397 % 36028797018963968!1|",
  "|abs(5122616779373027 % 72057594037927936 + shift - run_499_lap) <= 1 % 1000000!1497|",
  "|abs(6915481215411151 % 2251799813685248
        - (1 % 1 + (1 % 1 + (1 % 1 + (1 % 1 + run_499_lap)))))
```

```
    <= 1 % 1000000!1499|",
  "|eps <= 1 % 1!1500|"]]
*** Failed! Assertion failed (after 1 test):
BagList {
  _blDataLeft = [68.91823750212234,78.25974633049898,62.42855681062086],
  _blDataRight = [78.90182712823368,68.91823750212234,78.25974633049898,62.42855681062086]}
```

FuzzDP prints the unsat core reported from Z3, and QuickCheck prints the particular pair of similar inputs that the test failed on. In this case, the two lists are `[68.91823750212234,78.25974633049898,62.42855681062086]` and `[78.90182712823368,68.91823750212234,78.25974633049898,62.42855681062086]` (`78.90182...` is the differing element among these two lists).

To observe more internal logging from FuzzDP, users can change `expectDP` to a drop-in substitute `expectDPVerbose` to turn on verbose logging. But logging comes at a performance penalty.

Building and reading FuzzDP documentation with Docker

First launch a docker shell with the provided `fuzz-dp` image

```
$ docker run -it --rm fuzz-dp
```

Then, run the command

```
stack haddock
```

in the docker shell, which builds documentation for FuzzDP and all of its dependencies. Once this step completes, `stack` will report the locations of the built documentation. The output will look something like this:

```
Updating Haddock index for local packages in
/tmp/fuzzi-model/.stack-work/install/x86_64-linux/d3091bfee<abbreviated>/8.6.5/doc/index.html
Updating Haddock index for local packages and dependencies in
/tmp/fuzzi-model/.stack-work/install/x86_64-linux/d3091bfee<abbreviated>/8.6.5/doc/all/index.html
Updating Haddock index for snapshot packages in
/root/.stack/snapshots/x86_64-linux/d3091bfee<abbreviated>/8.6.5/doc/index.html
```

Next, run the following sequence of commands to copy the documentation to your host machine, and open them with a web browser of your choice.

```
# determine the running fuzz-dp container id
$ docker container ps -q
dfddd1e36017 # this is just an example, your output may be different
# run docker cp
$ docker cp dfddd1e36017:/tmp/fuzzi-model/.stack-work/install/x86_64-linux/d3091bfee<abbreviated>/8.6.5/doc ./doc
```

You may open doc/index.html for fuzz-dp documentation only, or open doc/all/index.html for fuzz-dp and all dependencies documentation