ProveNFix: Temporal Property-Guided Program Repair (Artifacts Evaluation)

YAHUI SONG, National University of Singapore, Singapore
XIANG GAO, Beihang University, China
WENHUA LI, National University of Singapore, Singapore
WEI-NGAN CHIN, National University of Singapore, Singapore
ABHIK ROYCHOUDHURY, National University of Singapore, Singapore

This document is submitted for artifact evaluation for paper #142 at FSE24. Please get in touch with Yahui Song (yahui_s@nus.edu.sg) for any necessary clarifications. Five checkpoints are highlighted using **Check Point #n**, claimed from the paper and the appendix.

1 STATUS

We are applying for the Available, Functional, and Reusable badges. Reasons are:

(1) The Benchmarks[1] are available from Zenodo. The Source Code[2] of the tool (ProveNFix) is available from GitHub. The final version of the Paper[3] and the Appendix[4] are publicly accessible.
(2) The functionality of the artifact can be validated by following the instructions in Sec. 2, which reproduce the experimental results claimed in the paper (Tables 2-5) and Appendix (C, D, E).
(3) Our artifact is reusable, given the detailed desecration and documentation provided in Sec. 3.

We appreciate your effort in trying out ProveNFix! We would love to hear your feedback!

2 README AND REQUIREMENTS

2.1 Build

For artifact evaluation, we have built a docker image, which (is recommended and) can be accessed by the following commands:

```
$ docker pull yahuuuuui/fse24-prove_n_fix:ubuntu
$ docker run -i -t yahuuuuui/fse24-prove_n_fix:ubuntu /bin/bash
# the source code repository is placed in "/home/", called "infer_TempFix"
```

Alternatively, one could build ProveNFix from scratch using a Linux system (tested on Ubuntu 22.04.4 LTS), with the following dependencies:

```
$ apt install opam menhir cmake z3 sqlite3
$ opam init
$ opam switch create 4.14.0
$ git clone https://github.com/songyahui/infer_TempFix.git # pull the repository
```

To test if the environment is sufficiently built (no matter it is in the docker or a local machine):

[1] https://zenodo.org/records/10695186
[2] https://github.com/songyahui/infer_TempFix
[3] https://www.comp.nus.edu.sg/~yahuis/ProveNFix.pdf
[4] https://www.comp.nus.edu.sg/~yahuis/ProveNFix_tr.pdf

Authors' addresses: Yahui Song, National University of Singapore, Singapore, Singapore, yahui_s@nus.edu.sg; Xiang Gao, Beihang University, Beijing, China, xiang_gao@buaa.edu.cn; Wenhua Li, National University of Singapore, Singapore, Singapore, liwenhua@comp.nus.edu.sg; Wei-Ngan Chin, National University of Singapore, Singapore, Singapore, chinwn@comp.nus.edu.sg; Abhik Roychoudhury, National University of Singapore, Singapore, Singapore, abhik@comp.nus.edu.sg.

```
$ cd infer_TempFix # direct into the source code folder
$ ./compile # This takes 3 mins from docker and up to 2 hours from scratch
```

If the compilation succeeds, we are ready for some testing! The docker image summarizes all the benchmarks, and the instructions below assume the given docker image environment.

2.2 Artifact Evaluation for RQ 1 and 2: find and fix bugs

(1) Checkout the `main` branch and build:

```
$ cd /home/infer_TempFix && git reset --hard && git checkout main # Use the main branch here
$ ./compile
```

(2) **Check Point #1** and **Check Point #2**: Use pre-defined specifications to analysis each project, reproducing the columns marked with "ProveNFix", in Table 2 and 3. We use the "Swoole" project for demonstration: (The experiments cannot be done in parallel, as they write to the same outcome file.)

```
# Reproduce Table 2 and 3 - project "swoole"
$ cd /home/infer_TempFix && cp spec_Swoole.c spec.c # Copy the specifications for "swoole"
$ cd /home/benchmarks-RQ1N2/swoole-src && make clean
$ /home/infer_TempFix/infer/bin/tempFix # Run ProveNFix!
# You will see the following text when finished:
==================================
[Lines of Spec] 52
[Predefined Spec] 26
[Generated Spec] 0
[Failed Assert] 98 # Check Point #1: this is the total bug found, shown in Table 2
[Repaired Bugs] 98 # Check Point #2: this is the total bug repaired, shown in Table 3
[Analysis (s)] 17.60901689528574
[Repair (s)] 1.9051456451379445

$ vim /home/infer_TempFix/TempFix-out/detail.txt
# Cf. Appendix C.1 for the bug report, manually classified by bug types: NPD, ML, and RL.
```

In Table 2, for the row of "Swoole", let the total bug number be N1=(30+23)+(12+16)+(13+6)=100. In Table 3, for the row of "Swoole", let the total repaired bug number be N2=53+28+19=100. Due to the deviations caused by (i) manual documentation; (ii) platform difference, and (iii) tool version regression, we accept N1'=N1±3 and N2'=N2±3 as the result ranges. The actual execution results for "Swoole" here, are N1'=98, N2'=98. The detailed classification of these bugs can be found in Appendix C.1.

► To reproduce the row of "lxc" in Table 2 and 3:

```
# Reproduce Table 2 and 3 - project "lxc"
$ cd /home/infer_TempFix && cp spec_Lxc.c spec.c # Copy the specifications for "lxc"
$ cd /home/benchmarks-RQ1N2/lxc && make clean
$ /home/infer_TempFix/infer/bin/tempFix # You will see the following snippet:
[Failed Assert] 56
[Repaired Bugs] 56

$ vim /home/infer_TempFix/TempFix-out/detail.txt
```

Here, N1=(5+19)+(10+12)+(5+5)=56, N2=24+22+10=56, N1'=56, N2'=56 The detailed classification of these bugs can be found in Appendix C.2.

▶ To reproduce the row of "WavPack" in Table 2 and 3:

```
# Reproduce Table 2 and 3 - project "WavPack"
$ cd /home/infer_TempFix && cp spec_WavPack.c spec.c # Copy the specifications for "WavPack"
$ cd /home/benchmarks-RQ1N2/WavPack && make clean
$ /home/infer_TempFix/infer/bin/tempFix # You will see the following snippet:
[Failed Assert] 52
[Repaired Bugs] 52

$ vim /home/infer_TempFix/TempFix-out/detail.txt
```

Here, N1=(20+21)+(3+9)+(0)=53, N2=41+12+0=53, N1'=52, N2'=52 The detailed classification of these bugs can be found in Appendix C.3.

▶ To reproduce the row of "flex" in Table 2 and 3:

```
# Reproduce Table 2 and 3 - project "flex"
$ cd /home/infer_TempFix && cp spec_flex.c spec.c # Copy the specifications for "flex"
$ cd /home/benchmarks-RQ1N2/flex && make clean
$ /home/infer_TempFix/infer/bin/tempFix # You will see the following snippet:
[Failed Assert] 24
[Repaired Bugs] 24

$ vim /home/infer_TempFix/TempFix-out/detail.txt
```

Here, N1=(14+4)+(3+1)+(0+1)=23, N2=18+4+1=23, N1'=24, N2'=24 The detailed classification of these bugs can be found in Appendix C.4.

▶ To reproduce the row of "p11-kit" in Table 2 and 3:

```
# Reproduce Table 2 and 3 - project "p11-kit"
$ cd /home/infer_TempFix && cp spec_p11.c spec.c # Copy the specifications for "p11-kit"
$ cd /home/benchmarks-RQ1N2/p11-kit && make clean
$ /home/infer_TempFix/infer/bin/tempFix # You will see the following snippet:
[Failed Assert] 37
[Repaired Bugs] 37

$ vim /home/infer_TempFix/TempFix-out/detail.txt
```

Here, N1=(2+2)+(12+15)+(5+1)=37, N2=4+27+6=37, and N1'=37, N2'=37 The detailed classification of these bugs can be found in Appendix C.5.

▶ To reproduce the row of "x264" in Table 2 and 3:

```
# Reproduce Table 2 and 3 - project "x264"
$ cd /home/infer_TempFix && cp spec-x264.c spec.c # Copy the specifications for "x264"
$ cd /home/benchmarks-RQ1N2/x264 && make clean
$ /home/infer_TempFix/infer/bin/tempFix # You will see the following snippet:
[Failed Assert] 21
[Repaired Bugs] 21

$ vim /home/infer_TempFix/TempFix-out/detail.txt
```

Here, N1=(0)+(11+5)+(2+3)=21, N2=0+14+5=19, and N1'=21, N2'=21 The detailed classification of these bugs can be found in Appendix C.6.

▶ To reproduce the row of "recutils-1.8" in Table 2 and 3:

```
# Reproduce Table 2 and 3 - project "recutils-1.8"
$ cd /home/infer_TempFix && cp spec-recutils.c spec.c # Copy the specs for "recutils-1.8"
$ cd /home/benchmarks-RQ1N2/recutils-1.8 && make clean
$ /home/infer_TempFix/infer/bin/tempFix # You will see the following snippet:
[Failed Assert] 75
[Repaired Bugs] 72

$ vim /home/infer_TempFix/TempFix-out/detail.txt
```

Here, N1=(22+8)+(11+29)+(1+7)=78, N2=30+36+8=76, and N1'=75, N2'=72 The detailed classification of these bugs can be found in Appendix C.7.

▶ To reproduce the row of "inetutils-1.9.4" in Table 2 and 3:

```
# Reproduce Table 2 and 3 - project "inetutils-1.9.4"
$ cd /home/infer_TempFix && cp spec-inetutils.c spec.c # Copy the specs for ""
$ cd /home/benchmarks-RQ1N2/inetutils-1.9.4 && make clean
$ /home/infer_TempFix/infer/bin/tempFix # You will see the following snippet:
[Failed Assert] 37
[Repaired Bugs] 37

$ vim /home/infer_TempFix/TempFix-out/detail.txt
```

Here, N1=(5+8)+(7+10)+(1+5)=36, N2=13+17+6=36, and N1'=37, N2'=37 The detailed classification of these bugs can be found in Appendix C.8.

▶ To reproduce the row of "snort-2.9.13" in Table 2 and 3:

```
# Reproduce Table 2 and 3 - project "snort-2.9.13"
$ cd /home/infer_TempFix && cp spec_snort-2.9.13.c spec.c # Copy the specs for "snort-2.9.13"
$ cd /home/benchmarks-RQ1N2/snort-2.9.13 && make clean
$ /home/infer_TempFix/infer/bin/tempFix # You will see the following snippet:
[Failed Assert] 98
[Repaired Bugs] 85

$ vim /home/infer_TempFix/TempFix-out/detail.txt
```

Here, N1=(33+34)+(15+16)+(1+1)=100, N2=67+13+2=82, and N1'=98, N2'=85 The detailed classification of these bugs can be found in Appendix C.9.

▶ To reproduce the row of "grub" in Table 2 and 3:

```
# Reproduce Table 2 and 3 - project "grub"
$ cd /home/infer_TempFix && cp spec_Grub.c spec.c # Copy the specifications for "grub"
$ cd /home/benchmarks-RQ1N2/grub && make clean
$ /home/infer_TempFix/infer/bin/tempFix # You will see the following snippet:
[Failed Assert] 12
[Repaired Bugs] 12

$ vim /home/infer_TempFix/TempFix-out/detail.txt
```

Here, N1=(6+5)+(1)+(0)=12, N2=11+1+0=12, and N1'=12, N2'=12 The detailed classification of these bugs can be found in Appendix C.10.

2.3 Artifact Evaluation for RQ 3: automatically detect double free errors

(1) Checkout the `doubleFreeClose` branch and re-build:

```
$ cd /home/infer_TempFix && git reset --hard && git checkout doubleFreeClose # This branch is
    specialised for reporting double-free errors
$ ./compile
```

(2) **Check Point #3**: Use pre-defined specifications to analysis the double free bugs, reproducing Table 4: (The experiments cannot be done in parallel, as they write to the same outcome file.)

```
# Reproduce Table 4 - project "lxc"
$ cd /home/infer_TempFix && cp spec_Temp_lxc.c spec.c # Copy the specifications for "lxc"
$ cd /home/benchmarks-RQ3/lxc && make clean
$ /home/infer_TempFix/infer/bin/tempFix # Run ProveNFix! You will see the following snippet:
[Failed Assert] 0
[Repaired Bugs] 0
$ vim /home/infer_TempFix/TempFix-out/detail.txt # It is supposed to be empty

# Reproduce Table 4 - project "p11-kit"
$ cd /home/infer_TempFix && cp spec_Temp_p11.c spec.c # Copy the specifications for "p11-kit"
$ cd /home/benchmarks-RQ3/p11-kit && make clean
$ /home/infer_TempFix/infer/bin/tempFix # Run ProveNFix! You will see the following snippet:
[Failed Assert] 4
[Repaired Bugs] 4
$ vim /home/infer_TempFix/TempFix-out/detail.txt # Checkout the detailed repairs

# Reproduce Table 4 - project "grub"
$ cd /home/infer_TempFix && cp spec_Temp_grub.c spec.c # Copy the specifications for "grub"
$ cd /home/benchmarks-RQ3/grub && make clean
$ /home/infer_TempFix/infer/bin/tempFix # Run ProveNFix! You will see the following snippet:
[Failed Assert] 5
[Repaired Bugs] 5
$ vim /home/infer_TempFix/TempFix-out/detail.txt # Checkout the detailed repairs
```

As shown in Table 4, there are no double-free bugs in "lxc", and projects "p11-kit" and "grub" each have 3 true double-free bugs, and ProveNFix generates 1 false positive for them each. So ProveNFix is supposed to output "0", "4", "4" bugs found for "lxc", "p11-kit" and "grub", respectively. The integrated bug report and patches (containing 8 records) are summarised in Appendix D, and each of them has correspondences in the output "`detail.txt`" file. Notably, the result for "grub" shows ProveNFix found 5 bugs because of one duplicated bug record, which can be easily spotted from the "`detail.txt`" file.

2.4 Artifact Evaluation for RQ 4: generating specifications for OpenSSL and analysis its applications

(1) Checkout the `Infer_OpenSSL` branch and re-build:

```
$ cd /home/infer_TempFix && git reset --hard && git checkout Infer_OpenSSL # This branch is to
      generate the most concise specs for openssl
$ ./compile
```

(2) Checkout the initial two specifications in the specification file:

```
$ vim /home/infer_TempFix/spec.c # you will see the follows text, containing two initial specs

#define SW_CHANNEL_MIN_MEM (1024*64) # This header is not important
/*@ return(arg): Post (TRUE, return(arg)) @*/
/*@ ERR_new() : Post (TRUE, ERR_new()) @*/
```

(3) Get the source code of OpenSSL-3.1.2, and generate the specifications for the OpenSSL APIs:

```
$ cd /home/benchmarks-RQ4/openssl-openssl-3.1.2 && make clean # Pre-configured openssl code
$ /home/infer_TempFix/infer/bin/tempFix # It takes ~20 mins, because OpenSSL has 556.3 kLoC
# You will see the following snippet:
[Predefined Spec] 2
[Generated Spec] 128
```

(4) **Check Point #4**: Check out the generated 128 specifications, extended to the same specification file, claimed in Paper Sec 7.5 and detailed in Appendix E.

```
$ vim /home/infer_TempFix/spec.c # you will see 2+128=130 specs in total now
```

(5) **Check Point #5**: Use the 130 specifications in total, to analysis OpenSSL applications and repair the bugs, reproducing Table 5 in the paper:

```
$ cd /home/infer_TempFix && cp spec.c ../opensslspec.c # Backup the generated specifications
$ git reset --hard && git checkout main # Use the main branch to do the analysis
$ ./compile
$ cp ../opensslspec.c spec.c # Copy back the specs for OpenSSL APIs

# Reproduce Table 5
$ cd /home/benchmarks-RQ4/opensslApp/keepalived # Pre-configured openssl app benchmarks
 *** Repalce the project here, "keepalived", with
     "thc-ipv6" "freeradius-server" "trafficserver" "sslsplit" "proxytunnel",
      and run the following 3 commands (CMDs) for each round (6 projects in total) ***
$ make clean # CMD 1: Clean the project
$ /home/infer_TempFix/infer/bin/tempFix # CMD 2: Run ProveNFix! You will see the following:
====================================
[Lines of Spec] 260
[Predefined Spec] 130
[Generated Spec] ?
[Failed Assert] ?
[Repaired Bugs] ?
[Analysis (s)] t1
[Repair (s)] t2

$ vim /home/infer_TempFix/TempFix-out/detail.txt # CMD 3: Checkout the detailed repairs
```

In Table 5, the "**Time**" column is computed by (`t1+t2`). The "**Target API**" column shows the bugs detected by misusing particular APIs, which can find correspondence from the "`detail.txt`"

file. It's important to note that ProveNFix detects more bugs than those shown in Table 5. This is because not all the API misuse was recorded from GitHub and has a "**Issue ID**", which further emphasizes the advantages of ProveNFix, in finding a more complete set of bugs.

3 REUSABILITY DOCUMENTATION

ProveNFix is built on top of Meta Infer. Specifically, ProveNFix relies on the infrastructure of Infer, which can input big C projects and does modular analysis based on the compiled AST. Therefore, any extension needs related to the infrastructure can be found in the Infer documentation. We here outline the components deployed by ProveNFix (https://github.com/songyahui/infer_TempFix).

3.1 Project Structure and Key Files

(1) The front-end forward verifier: In the file of "`infer_TempFix/infer/src/clang/cFrontend.ml`". The entry of the analysis can be found in the "`do_source_file`" function, which subsequently calls the "`syh_compute_stmt_postcondition`" function for the main Hoare-style reasoning, corresponding to the forward rules defined in paper Figure 11.

(2) The back-end entailment checker: In the file of "`infer_TempFix/infer/src/clang/ast_utility.ml`". The inclusion checking of plain regular expressions is done in the function "`inclusion'`", and the entailment of linear arithmetic constraints is done in "`entailConstrains`". Built on top of these two components, the entailment checker for IntRE (the specification language) is defined in "`effectwithfootprintInclusion`", corresponding to the rewriting rules defined in paper Figure 12.

(3) The program repair module: In "`infer_TempFix/infer/src/clang/cFrontend.ml`", the program repair is discharged by the "`program_repair`" function, which takes the target specification, and synthesis a program which satisfies the specification, corresponding to the paper Algorithm 1.

(4) The utility functions: All the data structures, pretty printing functions, and utility functions are defined in "`infer_TempFix/infer/src/clang/ast_utility.ml`".

(5) A specification parsing module: The parser for inputting the specifications is defined in two files: "`infer_TempFix/infer/src/clang/lexer.mll`", and "`infer_TempFix/infer/src/clang/parser.mly`".

(6) The specification input file: The input specification file is taken as default from "`infer_TempFix/spec.c`". To achieve a more flexible input for specifications, one could modify the "`retriveSpecifications`" function in "`infer_TempFix/infer/src/clang/cFrontend.ml`".

(7) The bug report file: The execution results, meticulously generated for the analysis, are output to the folder "`infer_TempFix/TempFix-out/`". This folder contains two files: "`detail.txt`" and "`report.csv`". The former records detailed information for bugs, while the latter summarises the bug numbers from each analyzed file, ensuring a comprehensive overview.

(8) The top-level file wrapping the execution commands: The executable of "`infer_TempFix/TempFix.ml`" is to create the output folder, and initialize the output files. The python file "`infer_TempFix/TempFixDataAnalysis.py`" generates an integrated summary from the output files and shows the summary on the terminal.