“ Automated Memory Error Repair Based on Hybrid Program Analysis ”の修士発表会の質問に対する回答書

QIAN ZECHANG

February 17, 2022

初めに、先生たちは私の修論発表と修士論文に対する指摘を行っていただけたことに誠にありがとうございます。指摘の点をもとに、修士論文を修正して再提出させていただきます。質問に対する回答と追加した内容を以下に記載させていただきます。

1　修論発表会の質問に対する回答

1.1　林先生の質問

林先生の質問に対する回答と論文中の修正箇所です。

質問: **RQ2 の評価主張の根拠は？**

回答: RQ2 とは、(1) 重要なテストの収集 (2) 複数リターン patch の生成 (3) 複数エラーの修復、この三つの問題を解決したかどうかです。当然全部の状況が修復できると言えませんが、今回の実験で作ったテストコードの修復結果により、希望な機能に達すると思います。テスト 8 は修復されない状況がありますが、それは HAMER の問題ではありません。HAMER の修復性能は利用する fuzzer の性能に依存するため、fuzzer がエラーを検知しないと HAMER 側が仕方がありません。だから、将来の仕事としては、性能が良い動的検知ツールを探し、あるいは自分で HAMER の修復アルゴリズムに適合する動的検知ツールを作ります。

修正箇所:

Section 5.3.2 (page)
Although we can address a part of the issue shown in test8 (Figure 5.4) by improving HAMER's repair algorithm (discussed in Chapter 7.3), the basic issue is that the fuzzer may fail to detect errors, which gets more common as the type of input becomes more complex. If the error is not detected by the fuzzer, HAMER will be unable to fix it, since the performance of HAMER also depends on the detection tool it uses. Experiments have shown that HAMER can handle these three problems well when the fuzzer detects the error, thus we can say that HAMER can handle these three problems well. In the future, we will find or develop a fuzzer with better performance and more compatibility with HAMER.

Section 5.3.2 (page)
HAMER can handle multiple returns, multiple errors, and collect critical tests when LibFuzzer detects the errors successfully.

質問: **実験データは平等か？**

回答: HAMER の現時点実装されたバージョンの scalability は良くないため、今回の実験は複雑なエラーに対する修復性能を評価します。だから、scalability の視点は平等ではありません。合成検体を作るときに、複雑なエラーパターンを再現することを方針とし、リアル検体のエラーを参考し、そして、自分で別の複雑なエラーパターンも想定し、論文の表 5.1 の六方面からテストコードを作成しました。全部のエラー

パターンを考えたと言えませんが、できるだけよく出現する複雑なエラーパターンを再現しました。だから、scalability を考慮しない場合に、この六方面のエラーに対する修復能力は今回の実験により平等で評価できると考えます。

修正箇所:

Section 5.2 (page)
Since the goal of making synthetic codes is to reproduce complex error patterns, we refer to memory leaks in real-world programs and also assume some more complicated cases. Of course, we can not guarantee that all cases are taken into account, so the goal of this experiment is to evaluate the repairability of the repair tool in the six directions of the error patterns shown in Table 5.1, and do not evaluate its scalability.

質問: **Patch の正しさの基準は？**

回答: Patch の正しさの基準はメモリリークを修復したかどうかです。そして、patch は新しいエラーを起こしません。人手で確認します。

修正箇所:

Section 5.2 (page)
We manually checked the patch to see if it fixed the leak without introducing the new error.

1.2 春日さんの質問

春日さんの質問に対する回答と論文中の修正箇所です。

質問: **記号実行を使わない理由は？**

回答: HAMER は既存の検知ツールを用いてエラー検知を行います。だから、エラー検知は本研究の注目点ではありません。既存の検知ツールと結合しやすく高い修復能力を持っているメモリリーク修復テクニックの提案は本研究の目的です。今回は実装に便利の視点から LibFuzzer を利用します。記号実行を使うと検体コードの計装と情報収集とエラー判定など多くの実装が必要です。そして、動的記号実行を用いてのメモリリーク検知の研究があります。将来の仕事としては HAMER に適合する動的検知ツールを開発することが非常に重要ですが、今回の研究の目的は有効なメモリリーク修復アルゴリズムを提案することです。

修正箇所:

Section 4.2 (page)
Since HAMER relies on the existing fuzzer to detect errors, its performance is also affected by fuzzer. The purpose of this research is to propose a memory leak repair technique that can be easily integrated with a fuzzer and has high repairability. We use LibFuzzer, which has high scalability and can detect memory leaks, to make HAMER implementation easier. In the future, we will find or develop a fuzzer that is more compatible and suitable for HAMER.

1.3 渡部先生の質問

渡部先生の質問に対する回答と論文中の修正箇所です。

質問: **修復途中の誤 patch とは？**

回答: Fuzzer はエラーを trigger するテストが出力できますが、正しい条件式の生成にテストが足りない状況がよくあります。例えば、`if(a<5)p=malloc(1);` に対し、 f uzzer は `a=3` を入力してメモリリークを trigger することができます。しかし、このテストを用いて `if(a<=3)free(p);` を生成する可能性があります。これは修復途中の誤 patch です。HAMER は fuzzer を用いて patch したコードを再検査し、`a=4` のような重要なテストを収集することができます。

修正箇所: ありません。Chapter 3.1(page) と Chapter 4.5.2(page) に説明があります。

1.4 田村先生の質問

田村先生の質問に対する回答と論文中の修正箇所です。

質問: **自分に都合の良い検体を作るか？**

回答: 合成検体を作るときに、複雑なエラーパターンを再現することを方針とし、リアル検体のエラーを参考し、そして、自分で別の複雑なエラーパターンも想定し、論文の表 5.1 の六方面からテストコードを作成しました。できるだけ実験の平等性を保証します。

質問: **実験は十分か？**

回答: scalability を考慮しなく、その六方面のエラーパターンに対する修復能力の評価に対して実験が十分だと思います。合成検体を増やすことができますが、今の合成検体との重複性が高いため、特に実験の意味がないと考えます。

修正箇所: この二つの質問に対しての論文中の修正箇所は林先生の二番目の質問と同じです。

2 論文に追加した内容

2.1 補足 1

HAMER の高い修復能力と fuzzer と結合しやすいことを書きます。

修正箇所:

Abstract (page)
HAMER has high repairability and can be easily integrated with a fuzzer.
Introduction (page)
HAMER has high repairability and can be easily integrated with a fuzzer, which can detect memory error and verify the patches generated by HAMER.
Introduction (page)
HAMER has high repairability and can be easily integrated with a fuzzer.

2.2 補足 2

Component-based program synthesis の説明を補足します。

修正箇所:

Section 2.3 (page)
We can use the above components to synthesize an expression that satisfies the test suite shown in Table 2.1. We assign the test suite to the synthesized expressions and use SMT-solver to check whether the logical formula is satisfiable. If it is the case, we output the result; otherwise, we continue to try other expressions. The test suite is satisfied when c of expression x+c is 1, hence we can obtain the result expression x+1.
Obviously, the synthesized result of CBPS depends on the quality of the test suite. If the test suite does not contain the important test, CBPS may synthesize the overfitting expression, which means that although this expression satisfies the test suite, it is not the result we want. Furthermore, if the given components do not contain the necessary operators and variables, CBPS is unable to synthesize the correct expression.