バグ事例集 Defects4J を対象とした形式的検証ツール KeY によるバグ検出

芦田　聖奈　【 ソフトウェア検証・解析学研究室 】

1　はじめに

現代社会において，ソフトウェアの信頼性は極めて重要である．形式的検証はプログラムの正当性を厳密に保証できる手法である．本研究では，Java のバグデータセットである Defects4J[1] を対象とし，形式的検証ツール KeY を用いて実際のバグを検出することを試みた．本研究の目的は，形式的検証が実際のバグの特定と修正にどのように寄与できるかを明らかにすることである．

2　形式的検証ツール KeY

KeY[2] とは，Java 言語を対象としたプログラム検証ツールである．ソースコードに対し，仕様記述言語 JML (Java Modeling Language) を用いて事前条件や事後条件を付加する．検証時には動的論理 [3] を用いてプログラムの挙動を定式化し，その式が仕様を充足することを数学的に証明する．これにより，特定の条件下でのみ発生するバグを網羅的に検出することが可能となる．

3　研究手法

本研究では，Defects4J に含まれるバグ事例のうち Apache Commons Lang プロジェクトから抽出された 10 件を対象とし，以下の手順で実験を行った．

1. バグの混入版と修正版をチェックアウトし，テスト実行とソースコードの分析からバグの原因箇所を特定する．

2. KeY を用いた検証を可能にするため，対象メソッドを抽出した上でパッケージ依存等を排除した検証用 Java ファイルを作成する．

3. 外部メソッドのスタブ化によるモデル化を行う．計算資源の枯渇を防ぐため，検証対象外のメソッドを特定の戻り値等を返す簡易的なプログラム (スタブ) に置き換え，解析のリソースをバグの原因となっている主要な箇所に集中させる．

4. JML による仕様定義を行い，KeY での検証を試行する．Open Goal の発生により，バグが発生する具体的な条件が特定されることを確認する．

5. 特定した原因に基づきコードを修正し，欠陥が解消されることを確認する．さらに開発者による修正版と比較し，自作の修正案との差を分析する．

4　実験結果および考察

以下,「16 進数文字列の変換処理におけるバグ」を例に実験結果と考察を述べた後，全体の傾向を述べる．

まず，バグ混入版の検証においては，KeY を用いることでテストコードでは到達困難な境界条件における欠陥を特定できた．16 進数 16 文字かつ先頭が 8 以上の場合に，数値型の上限を超えてオーバーフローが発生するという具体的な矛盾が数式上で特定され，文字数のみに頼る型判定の欠陥が客観的に示された．

次に，特定した原因に基づき，以下の自作案および開発者案の双方に対して検証を試みた．自作の修正案では，9 文字以上の入力を一律に制限する事前条件を追加した．検証の結果，バグの発生条件は遮断されていたが，スタブ化の不備等により Proof Closed には至らず，Open Goal が残った．開発者の修正案では，有効数字を判定するロジックが導入された．検証はループ処理の複雑さから Open Goal となったが，解析結果の数式を調査したところ，原因であったオーバーフローが条件分岐によって回避されていることを確認できた．

10 件のバグ事例を通じた全体の傾向として，自作案のような単純な制約による修正は Open Goal の数が少なく正当性の確認が容易であったのに対し，開発者案のように可用性を損なわない精緻な設計ほど，検証ツール側の計算負荷が増大し，解析が困難になる傾向が確認された．このように，実用的なコードの検証には困難が伴うものの，形式的検証は完全な自動証明に至らない場合であっても，バグの発生条件の精密な特定に有効であることがわかった．さらに，Open Goal として残された数式を詳細に解析することで，修正案による欠陥解消を客観的に評価できることが示された．

5　まとめ

本研究では，Defects4J の実際のバグ事例に対し，形式的検証ツール KeY を用いてバグ検出および修正案の分析を行った．実験を通じて，Proof Closed には至らなかったものの，形式的検証によってバグの発生条件がどのように変化するかを追跡することができた．今後は Open Goal に対してより適切な定義を行い，実用的な修正案に対しても Proof Closed を実現できる環境を構築することが課題である．

参考文献

[1] R. Just, D. Jalali, and M. D. Ernst, “Defects4J: A Database of Existing Faults to Enable Controlled Studies for Java Programs,” in ISSTA 2014, pp. 437–440.

[2] B. Beckert, et al., “The Java Verification Tool KeY: A Tutorial,” in FM 2024, pp. 597–623.

[3] D. Harel, J. Tiuryn, and D. Kozen, Dynamic Logic. MIT Press, 2000.