A-005

# Java プログラム中に現れる数式表現の揺らぎを検出するシステムの評価と改善

秀山 祐司　　樋口 昌宏

近畿大学大学院総合理工学研究科

## 1. はじめに

ソフトウェアの保守性を向上させるためにはソースコードの可読性を高めることが重要である．可読性を妨げる要因として同じ意味の数式を異なる表現で記述している場合がある．本研究では，この様な場合を数式表現の揺らぎと呼び，Java プログラム中からこれを検出しプログラマに提示することで可読性を向上させる事を目的とする．検出手法では，プログラムから抽出した全ての数式に変数名やメソッド呼び出しの引数の違いを吸収する抽象化を行った上で，それぞれの式が等価であるかを判定することにより揺らぎを検出する．しかし実際のプログラムに適用したところいくつかの問題点が見られた．本発表ではその問題点に対する改善点について示す．

## 2. 数式表現の揺らぎ

まず，数式表現の揺らぎに関して記述する．本研究の数式表現の揺らぎ(以下，揺らぎ)とは，プログラム中に出現する 2 つの数式が論理的に等価であるにも関わらず異なる表現になっていることを指す．例えば，英語・数学に基づいた記述では係数を先に記述する(3*apple+4*orange)場合や，日本語では主語述語の順に記述する(apple*3+orange*4)場合等がある．この様に，プログラマ毎に思考言語，思考方法が異なるため揺らぎが現れると考えられる．

次に検出したい揺らぎに関して記述する．数式は各種算術演算子とそれ以外の構成要素であるフィールド変数，ローカル変数，メソッド呼び出しの戻り値，定数で構成されている．表 1，2 に揺らぎとしたい数式の例を示す．height，width，depth はフィールド変数， input，tmp，kbdIn はローカル変数とする．表 1 の 1)は式 1 と式 2 でフィールド変数とローカル変数の順序が入れ替わっているため揺らぎとする．2)についてもローカル変数とメソッド呼び出しの戻り値が現れる順序が入れ替わっているため揺らぎとする．3)は括弧で囲まれた部分式の位置が異なるため揺らぎである．4)は分配則によって等価であるが表現が異なるので揺らぎとする．

表 2 の 1)は変数と定数の順序が入れ替わっている，2)は 2 つの変数の比較の仕方が異なっている，3)は用いる不等号が異なっている，にもかかわらず全て論理的に等価であるため揺らぎであるとして検出対象とする．

表 1 算術式の揺らぎ

| | 式 1 | 式 2 |
|---|---|---|
| 1) | height+input | tmp+width |
| 2) | kbdIn<br>+width.size(input) | depth.get(tmp)<br>+kbdIn |
| 3) | height<br>*(input+tmp+kbdIn) | (input+tmp)*width |
| 4) | input<br>*(height+depth) | input*height+<br>input*depth |

表 2 不等式の揺らぎ

| | 式 1 | 式 2 |
|---|---|---|
| 1) | kbdIn > 0 | 0 < kbdIn |
| 2) | height > input | height − input > 0 |
| 3) | tmp > 0 | tmp >= 1 |

## 3. これまでの検出システム

これまでの検出手法について記述する．まず，対象のプログラムから数式を全て抽出しテキストファイルに書き出す．そのファイル中の数式に対し構文解析を行い，変数名やメソッド呼び出しの引数の違いを吸収するために後述する抽象化を行う．それらの抽象化された数式(抽象式)に総当たり的に等価性評価を行う．2 つの数式の等価性評価には SMT ソルバ[1]を用いる．これにより等価であると判定された場合揺らぎとみなし，元の数式とプログラムの何行目に存在しているのかを表示する． 2 節で述べた変数名の違いにかかわらない揺らぎを検出するため，対象プログラムから抽出された全ての数式に抽象化を行った．抽象化とは，数式に現れる変数等を種類毎に別の文字列に置き換えることを言う．例えば，対象プログラム中に height+input と tmp+width という数式があったとしてもそのまま SMT ソルバにかけても 2 つの式は等価ではなく揺らぎとして検出されない．そこで，フィールド変数，ローカル変数，メソッド呼び出し，算術

式のそれぞれの頭文字 f，l，m，e を取り，左から順に出現した変数名を，例えばフィールド変数であれば，f1，f2，f3，…に変換する．メソッド呼び出しに関しては，与えられた引数に違いが存在しても同じメソッド呼び出しとして判定することができる．これにより，上記の抽出された数式はそれぞれ f1+l1 と l1+f1 という形に抽象化され，SMT ソルバにより等価と判定され揺らぎとして検出できる．

## 4. 検出システムの問題点と解決策

上記の検出システムをいくつかのプログラムに適用したところ，いくつかの問題点が見られた．それらの問題点と解決策を以下に記述する．以下，height，width をフィールド変数，input をローカル変数とする．

同じ種類の変数が異なる順序で現れた場合，例えば height+width と width+height では抽象化を行った際にどちらも f1+f2 となり揺らぎとして検出しなかった．しかし適用したプログラムのいくつかにその様な場合が頻出し，可読性を低下させていた．これは数式に対して抽象化を行い過ぎた為に生じた問題である．抽象化を行わずに元の数式同士でも等価性判定を行う必要があると考えられる．

また，for 文の繰り返し条件式に現れる不等式を他の不等式と同列に扱い比較し揺らぎとして検出していた．しかし，これらはプログラムの中で記述の特性が異なるので揺らぎとすべきではなく，プログラマに提示するべきではない．カウンタ変数は他のローカル変数と別の種類の変数として扱うべきであると考えられる．

さらに，構文解析と抽象化を行う際にテキストファイルに書き出された数式に対して 1 行ずつ読み進め，一括で同時に行っていたため，数式 1 つに対して 1 種類の抽象式のみ作成していた．例えば input*(height+depth)は部分式を優先的に判定し l1*e1 のみ作成され，括弧を部分式として見なさない パターンの l1*(f1+f2)は作成されなかった．これにより表 1 の 4)の input*height+input*depth，即ち l1*f1+l1*f2 の様な数式との揺らぎを判定することができなかった．また二重括弧の場合も外側括弧を部分式とした抽象式のみを作成するようにしていた．しかし括弧を部分式と見なさない場合や，内側括弧のみ部分式とした場合等 1 つの数式に対していくつかの抽象化パターンが存在しうる．これには 1 つの数式から複数の抽象式の作成を行う．これらは全て別の抽象式として作成される事で，多くの数式パターンの評価を行い，より多くの揺らぎを検出する事ができる．

加えて，抽象式をそれぞれ総当たりで等価性判定を行うため，抽出した数式の数が増えると等価性評価に掛かる時間が増加する．解決策として抽象式群の中から複数存在する同じ抽象式を取り除くことを考えている．対象プログラムから揺らぎを検出するためにはその性質上総当たりで等価性評価を行う必要がある．しかし，抽象式群の中に全く同じ抽象式が現れた場合は1つに絞り込むことで等価性評価の対象の数を減らしつつ総当たりを満たすことができる．

## 5. 新しい検出システム

今までの検出システムに加え，4 節の解決策をシステムに追加する．対象プログラムから抽出した数式に対して構文解析を行い，抽象化を一切施さない式を含む様々なパターンの抽象式を作成する．対象プログラムに for 文の条件式にカウンタ変数が現れた場合は，これまでの構成要素の抽象化に加えてカウンタ変数の抽象化を行う．現在抽象化するために分類した変数の種類は，フィールド変数，ローカル変数，メソッド呼び出しの戻り値，括弧で括られた式を部分式とした 4 つである．これらに加えて for 文等に使われるカウンタ変数の分類を追加(抽象化後:c1，c2，c3，…)する．今まではフィールド変数以外の宣言された変数全てをローカル変数として見なしていた．カウンタ変数を追加する事で条件式にカウンタ変数が含まれていれば他の数式と差別化を行う事ができ，条件式同士で揺らぎの判定を行う事が可能となる．等価性判定を行う前に，抽象式と元の数式の中に全く同じ式が複数存在すれば 1 つに絞り込む．残った抽象式群，元の数式群それぞれに対して総当たり的に等価性判定を行う様にする．

## 6. 今後の予定

4 節で挙げた問題点に対する解決策を含めた 5 節のシステムについて現在誠意作成中である．今後は，作成したシステムをオープンソース等様々なプログラムに適用させ，解決策の評価と問題点の考察を行う予定である．

## 参考文献

[1] 梅村 晃広 SAT ソルバ・SMT ソルバ の技術の応用，コンピュータソフトウェア,Vol.27,No.3(2010)
[2] GitHub [https://githun.com/Z3Prover/z3]
2020/6/18 閲覧