変数に関する不吉な匂い除去のための開発支援方法の提案
―データの群れの配列化―

M2021SE007 松井亮介
指導教員：蜂巣吉成

1　はじめに

ソフトウェア開発において変数は，開発者のコード理解やコードの保守性へ影響を及ぼす要素であり，どのように変数を用いるかは重要である．変数は使い方によってコードの不吉な匂いを生み出すことがある．これはファウラーによるリファクタリングカタログ [1] において体系化されている．その 1 つにデータの群れ [1] がある．これは数個のデータがグループとなってクラスのフィールドやメソッドのシグニチャなどさまざまな箇所に現れる状況を指す．引数の場合は多すぎる引数 [1] と呼ばれることもある．データの群れにより理解の妨げになることやメソッドを呼び出す際に記述する実引数が増え，変更が容易でなくなるという影響がある．

データの群れに関する修正作業を行うための問題点として，膨大な数の変数に対しそれぞれの振る舞いを理解するための作業コストがかかるという点が挙げられる．変数を修正するためには，それぞれの振る舞いを十分に理解した上で，修正すべきものを適切に選択し修正しなければならない．コードの不吉な匂いに対し，支援方法が提案されているが [2][3][4]，データの群れは抽象的であるために，パターン化が難しいという技術的な課題がある．

本研究の目的は変数の群れを排除するために，変数の型やデータ構造の修正を支援する方法を提案することである．変数の使い方を評価し共通する特徴を持つ変数を配列など別のデータ構造に修正する方法を考察する．ここでは C 言語を対象に，変数として仮引数と局所変数の 2 つを扱う．一般に大域変数は避けるべきとされるので対象から除外した．データの群れの排除を支援することで，修正作業の効率化およびコードの品質向上に貢献する．

2　関連研究

変数の群れに関連するカタログとして，クラスの抽出や一時変数の削除，引数の削除 [1] などが示されており，これらを支援する研究も提案されている [2][3][4].

例えば小田ら [2] の提案では，C プログラムのカプセル化を支援する方法が提案されている．型とグローバル変数，関数というオブジェクト要素を解析し，それらをふるい分けることによってカプセルを導出する．Bavota ら [3] はクラス抽出を支援するために，関連するメソッドの抽出を図るアプローチを提案している．村松ら [4] はリファクタリング箇所を特定するために，クラス間の関係とメソッドの内部構造の表現を同時に扱うためのパターン記述言語を提案しているが，多すぎる引数をはじめ，表現が難しいものもある．

本研究では先行研究が課題としていたり，扱っていないデータの群れに焦点を当て，修正箇所の発見を支援する．

3　開発支援方法の提案

本研究では変数の使われ方を評価することで，変数の型やデータ構造の修正を支援する方法を提案する．共通の使われ方をしている変数がデータ構造を修正可能な変数群であると考え，図 1 のように前処理，2 変数間での文の評価，3 変数以上への評価結果の拡張という 3 つのプロセスによって変数群を発見する．

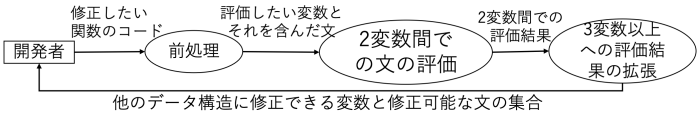

図 1　修正支援方法のプロセス

修正したいコードに対し前処理にて必要な情報を抽出し，2 変数間での文の評価によって修正後の型やデータ構造に応じた観点や基準での評価を行う．このときデータの群れを探すために式の粒度に着目する．配列化ならば繰り返し文という観点で，文から添字になりうる部分や定数，変数などを捨象したものが一致するか評価する．構造体化ならば変数が組で利用されるという観点で，変数が同じ関数呼び出しの実引数に現れるといった点などを評価する．複雑な評価に対応するために 3 変数以上へ評価結果を拡張し，他のデータ構造に修正できる変数と修正可能な文を集合として抽出する．

文献 [4] のパターン記述言語は式の粒度でパターンを表現しておらず，データの群れを排除するための観点や基準を表せない．そもそも評価の観点や基準はどのようなものが表現できれば良いかわからないので，パターンで表現することが難しい．本研究では修正後の型やデータ構造に応じて式の粒度の評価方法を提案する．

C 言語のサポートするすべての型やデータ構造が修正後の候補となる．本稿では先行研究で着手されていないものの，事例調査の結果からナンバリングされた変数があった点を考慮し配列化支援方法を提案する．

4　配列化支援について

4.1　配列化支援概略

配列化支援では配列にできる変数と繰り返しに修正可能な文の発見を行い，発見した変数と文の書き換え方法を修正ガイドラインに示す．これは 4.2 節に述べるように，配列になる変数に対しては同じ計算をしている，すなわち繰り返しで記述されることが多いと仮定し，繰り返しに書き換えられる文が配列化の判断材料となるという着想に基づいている．

変数を配列に直すことで複数のデータを一括で操作でき，データの群れにあたる変数を削減できる．群れをなす

データを 1 つの識別子に集約して可読性を高めたり，関数呼び出しなどで複数のデータを 1 つの識別子で簡潔に受け渡し可能となり保守性向上の効果が生まれる．さらに配列に集約した変数は添字で操作できるので繰り返し文と相性が良く，繰り返し文を用いて記述してコードを簡潔にしたり，条件式で処理を容易に修正可能となる．

提案にあたってナンバリングされた変数を例として扱う．これは調査にてナンバリングされた変数は 3 つの OSS[1][2][3] から 163 種類見つかっており，どれも同じ型であったこと，開発者から好まれないことから採用した．

4.2 配列になり得る変数

配列にできる変数について議論する．配列とは「同一の型のデータを一列に有限個並べたもので，添字を用いて操作するもの」である．配列の特性を次に示す．

- 配列の各要素は同じ型である
- 配列の各要素は同じ種類を表すデータである

例えば身長や体重は共に実数のデータとなるが，一般にこれらが混在した配列は用いず，身長の配列と体重の配列のように区別する．このとき各配列のデータは同じ種類となる．これら 2 点の特性を持つ変数は配列になる可能性を持つと考えられる．

同じ型であることは，同じ宣言子で定義された変数を探せばよい．同じ種類を指すデータを宣言子や格納している値の類似性から判別することは難しい．そこで変数を用いた計算過程に着目し，同じ計算がされている変数同士を発見する．同じ種類のデータを格納した配列の場合，各要素で類似する計算をする可能性が高いと考えた．コードでは計算は文単位で表現されることから，文に着目し同じ計算となる文同士を判定する必要がある．同じ計算をする文は構造も同じと考えられ，それらは繰り返しで記述可能な文同士となる．同じ計算がされている変数同士とは，配列化によって繰り返し文に書き換えられる文を持った変数同士とも言える．

具体的には同じ計算を次の 3 つの基準によって判定する．

観点 A すべて 完全一致する文同士ならば，同じ計算をしていると考える．

観点 B 添字 繰り返し文で記述する文は添字部分のみが規則的に変化する．添字部分以外が一致する文同士は同じ計算をしている可能性が高いと考える．

観点 C 演算子 文中で演算子は計算と直接関係する要素と捉え，同じ演算子を用いている文同士は同じ計算をしている可能性があると考える．

これらの基準を取り入れた比較方法は 4.3.2 項で説明する．

4.3 配列化支援のアプローチ

関数に対し，配列にできる変数と繰り返しに修正すべき文の候補の集合を提示することで支援を行う．4.2 節で述べたように，繰り返し文に書き換えられるような同じ計算をする文は配列化の判断材料になると考え，これを抽出して支援を図る．そのために変数を含んだ文をいくつかの基準から比較し，繰り返し可能な文を特定する．

4.3.1 前処理

コードから変数と，それらを含んだ文を抽出する．関数の仮引数と局所変数のうち，ポインタを除いた基本型またはユーザ定義型の変数を扱う．for 文や if 文などの制御文は条件式を 1 文として抽出する．

4.3.2 2 変数間における文同士の評価

繰り返し可能な文を発見するための評価方法を次に示す．手順 1，2 をすべての変数の組み合わせに対して行う．

手順 1 取り上げる変数を 2 つ選ぶ

手順 2 取り上げた変数を含むそれぞれの文を 3 つの基準から比較し，同じ計算をしている文同士を探す

手順 2 では取り上げた 2 変数に対して文同士を比較する．取り上げた変数それぞれを含む文を総当たりで比較する．その際，同じ計算とは基準によって評価が異なる点を考慮し，3 つの比較で文が一致するか判定する．各比較の基準と比較内容を次に示す．

比較 A:すべて 繰り返しで記述できることが自明な文を見つけるために行う．比較対象変数以外のすべてをそのまま比較する．

比較 B:変数 繰り返しになる可能性が高い文を見つけるために行う．添字部分のみ規則的に変化する文は繰り返しで記述できる．しかし添字がとる値をモデル化することは困難である．そこで添字部分以外が同じ文の発見を行う．この比較では添字部分にあたる，変数に付与された数字や定数のみを除き比較する．

比較 C:演算子 繰り返しで記述できる可能性を持つ文を見つけるために行う．この比較では文の変数や定数，マクロという 3 つのトークンを除いて比較する．

比較に応じて 4.3.4 項で示す修正作業の際に考えるべきことが変わる．

事例調査に用いたソースコードより，out_val1 と out_val2 という変数を含む 2 つの文を比較する場合を例として説明する．比較した結果，必ずしも文は 1 対 1 で

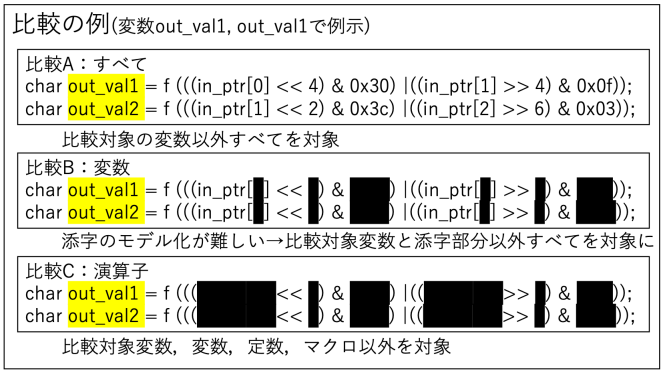

図 2 比較の例

一致するとは限らない．その場合，比較 A から文の対応関係を定め，対応づかない場合は関数内での文の位置が

[1] https://github.com/facebookresearch/darkforestGo
[2] https://github.com/hashcat/hashcat-legacy
[3] https://github.com/EricssonResearch/openwebrtc

近い方を対応するものとして扱う．一致した文同士は繰り返しになり得る文の集合として扱う．

変数 v_a, v_b に対し，v_a, v_b を含む文の集合を $S(v_a), S(v_b)$ とする．X を比較 A,B,C いずれかとすると，v_a, v_b について比較 X となる文の集合を次のように表す．

$compare(v_a, v_b, X) = \{(v_a, s_a), (v_b, s_b)\}\ (s_a \in S(v_a), s_b \in S(v_b))$

4.3.3 3 変数以上への評価結果の拡張

2 変数間の compare を推移的に適用し，3 変数以上における修正可能な変数と文の候補となる集合 R を生成する．

1 $R(\{v_a, v_b\}, X) = compare(v_a, v_b, X)$ とする．

2 $R(V, X) \cap compare(v_b, v_c, X) \neq \phi$ ならば，
$R(V \cup \{v_b, v_c\}, X) = R(V, X) \cup compare(v_b, v_c, X)$ とする．

例を示す．変数 v_1, v_2, v_3, v_4 に対し
$S(v_1) = \{s_{11}, s_{12}\}, S(v_2) = \{s_{21}, s_{22}\}$,
$S(v_3) = \{s_{31}, s_{32}\}, S(v_4) = \{s_{41}, s_{42}\}$ という文集合があり，次のような結果が得られたとする．

$compare(v_1, v_2, B) = \{(v_1, s_{11}), (v_2, s_{21})\}$
$compare(v_2, v_3, B) = \{(v_2, s_{21}), (v_3, s_{31})\}$
$compare(v_3, v_4, B) = \{(v_3, s_{31}), (v_4, s_{41})\}$
$compare(v_1, v_4, C) = \{(v_1, s_{12}), (v_4, s_{42})\}$
$compare(v_2, v_3, A) = \{(v_2, s_{22}), (v_3, s_{32})\}$

すると，集合 R として次のものが得られる．

$R(\{v_1, v_2, v_3, v_4\}, B) = \{(v_1, s_{11}), (v_2, s_{21}), (v_3, s_{31}), (v_4, s_{41})\}$
$R(\{v_1, v_4\}, C) = \{(v_1, s_{12}), (v_4, s_{42})\}$
$R(\{v_2, v_3\}, A) = \{(v_2, s_{22}), (v_3, s_{32})\}$

得られた集合 R は配列化できる変数と繰り返しに書き換え可能な文の候補を示している．例えば $R(\{v_1, v_2, v_3, v_4\}, B)$ であれば，4 つの変数 v_1, v_2, v_3, v_4 が配列化でき，それに伴って文 $s_{11}, s_{21}, s_{31}, s_{41}$ を繰り返しに書き換えられる可能性がある．開発者が集合 R を参照して修正するか判断する．

4.3.4 集合 R を用いた修正作業

集合 R 用いて修正を行うためのガイドラインを示す．次の手順を得られたすべての集合 R に対して行う．

修正ガイドライン

- 得られた集合 R のうち 1 つに着目し，どのように修正するか考える．
  繰り返しへ修正する際，比較 A，B，C に応じてカウンタ変数を用いて添字などを表す式を考える．

  比較 A 考えるものは特にない

  比較 B 添字と定数値

  比較 C 添字や定数値，および変数

  繰り返しにするための添字部分が複雑になる場合や，繰り返しにすることでコードが増えてしまう場合などは修正しない．集合 R に抽出される文は，コードの中での位置が並んでいる場合と離れている場合がある．文が並んでいる場合は並ぶものすべてを繰り返しにすれば良い．文が離れている場合，次の手順のように前後の文を繰り返しに含めるか判断し書き換えを行う．
- 着目した集合 R と同時に修正できる文を考える．
  着目した集合 R 以外の集合 R について，同時に繰り返しが可能か考える．それ以外にも着目した集合 R の前後の文を参照し，同じ繰り返し文に含めるか考えなければならない．

5 評価

ナンバリングされた変数を持つ関数に対して提案方法を適用し修正作業を行うことで評価する．評価の目的は，実際のコードに方法を適用し修正できる文が見つかるかを確認することと，修正が可能か確認することにある．

5.1 評価方法

事例調査で調査したナンバリングされた変数のうち，4 つの変数がある 14 種類のいずれかを持った関数をテストデータとして利用する．評価手順を次に示す．

- テストデータに提案方法を適用し集合 R を求める．
- ガイドラインに従って修正作業を行う．
- 基準ごとに抽出された集合 R の数とそのうち修正作業で修正できた集合 R の数，および抽出した各集合 R の持つ要素数を集計する．同時に集合 R で抽出される文についても，各比較基準で抽出された文の数，そのうちで修正できた文の数を集計する．

5.2 評価結果

テストデータに方法を適用し，各比較基準で得られた集合 R の数を集合の要素数別に集計したものと，そのうち修正できた集合 R についての集計を表 1 に示す． 表 1

表 1 抽出された集合数について

比較基準	要素数 2 の集合数	要素数 3 の集合数	要素数 4 の集合数	抽出された合計集合数
A	0/3	0/0	37/37	37/40
B	0/9	0/2	67/67	67/78
C	0/0	0/0	0/0	0/0
合計	0/12	0/2	104/104	104/118

の右下の値は 104/118 であるが，これは 118 個得られた集合 R のうち，104 個の集合 R が繰り返し文へ書き換えができたことを示す．この 104 個の集合 R はすべて要素数 4 であった．118 個の集合 R は比較 A，B のどちらかで得られており，比較 C からは得られなかった．

次に比較対象となったすべての文のうち，いずれかの集合 R に抽出された文の数についての集計を表 2 に示す．ただし比較 C で得られた集合 R はなかったので表からは省略した． 表 2 は 1 行目ならばいずれかの集合 R に抽出

表 2 比較対象となった文について

集計対象	すべての文数	比較 A における文数	比較 B における文数
抽出された文	439	154	285
修正できた文	416	148	268

された文が 439 文あり，そのうち 154 文が比較 A で抽出され，285 文が比較 B で抽出されたことを示す．評価のテストデータにおいて，ナンバリングされた変数を含む

文は合計で 452 文あった．そのうち集合 R として抽出された文は 439 文で，13 文は抽出されなかった．抽出された 439 文のうち，416 文は修正が可能であった．

修正作業を実施した例として図 3 を例示する．out_val0, out_val1, out_val2, out_val3 を対象に修正を行う．なお，ナンバリングされた変数を含む文のコメントには s1，s2 といった様に識別用の番号を振っている．

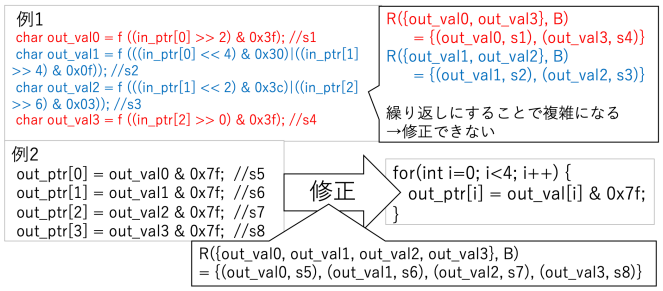

図 3　修正例

例 1 では，out_val0, out_val3 と out_val1, out_val2 の間で比較 B の集合が得られた．しかし，これらは文が順番に現れるものではなく添字の表現も難しい．繰り返しにすることで記述がわかりにくくなる可能性を持つ．よって修正できないとして扱う．例 2 では，out_val0, out_val1, out_val2, out_val3 が比較 B で 1 つの集合となっている．添字部分も 0 から 3 まで単調増加する．これは繰り返し文での書き換えることができる．

6　考察

6.1　支援について

配列化については評価の結果から，修正できる候補として 118 個の集合 R を発見し，そのうちの約 88%にあたる 104 個の集合は実際に書き換えられる可能性があることが確認できた．評価にて使用したテストデータのような関数であれば支援できる可能性がある．一方で書き換えられなかった文は，4 つある変数のうち 2 つまたは 3 つの変数の間で得られた集合 R の文であった．図 3 の例 1 は 2 つの変数における集合 R の例にあたる．$s1, s2, s3, s4$ はそれぞれ $s1, s4$ と $s2, s3$ の文が集合となるが，これらは文の位置が離れており修正が難しい．比較 A においても要素数 2 の集合 R は同様の理由で修正が難しいものであった．3 つの変数における集合 R の例として，図 4 がある．これは添字の式が直感的に表現しづらく，書き換えによる恩恵が少ないと判断し修正していない．このように 4 つある変数を網羅しない場合，別の繰り返しになる文同士が互いに干渉する位置にあるなどの問題があり，書き換えると複雑化していた．

本稿では配列を取り上げたが，他のデータ構造も支援できるように拡張することが望ましい．図 1 のプロセスを用いた構造体化の修正支援方法として，変数の特徴ベクトルを用いて構造体への修正候補となる変数を発見する方法を挙げる．2 変数間での文の評価では，変数間の特徴ベクトルを生成する．その際は同じ関数の実引数に用いられることや，同じブロックで値が複製されることと

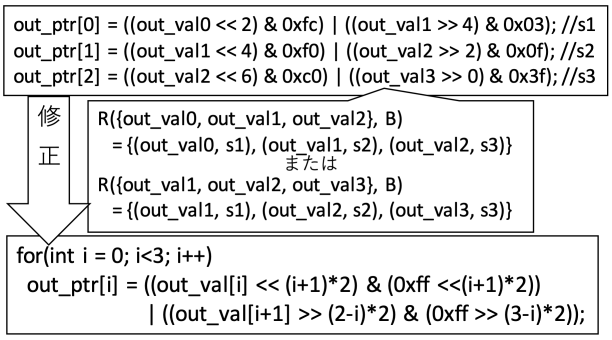

図 4　繰り返しにすると複雑になる例

いった変数を用いた操作に着目し，それらの出現回数や各操作の重要度を反映した重みづけを行うことで，構造体化可能な変数の特徴を表現できると考えられる．3 変数以上への評価結果の拡張では，生成した特徴ベクトルを距離などを元にまとめる．まとめたベクトルに該当する変数を近い特徴を持った変数同士であると考え，構造体への修正候補となる変数として抽出する．

6.2　妥当性への脅威

本研究では調査に用いた 3 つの OSS を用いているが，他の OSS においての結果も検証しなければならない．修正作業を行う際，添字部分をどのように表現するかは修正者に依存するので，繰り返しへ書き換え可能かについては修正者によって扱いが変わる．

7　おわりに

本研究では変数の不吉な匂いを排除する作業を支援するために，C 言語を対象とした変数のデータ型や構造の修正支援を提案した．変数を含む文に対し，いくつかの観点から比較することで配列化可能な変数と修正可能な文を集合として算出する．これを提示することによってコードを読む作業を支援し，修正すべきか判断を促すアプローチを取った．今後の課題は構造体をはじめ対応可能なデータの拡張と，さまざまな例における検証を行うことである．

参考文献

[1] Martin Fowler and Kent Beck: Refactoring: Improving the Design of Existing Code., Addison-Wesley Professional, 1999.

[2] 小田章夫, 鯵坂恒夫: C プログラムに対するカプセル発見手法とその支援ツールについて, 情報処理学会ソフトウェア工学研究会報告, vol.107, pp.41-48, 1996.

[3] G.Bavota, A.De Lucia, A.Marcus, and R.Oliveto: A two-step technique for extract class refactoring. , In Proc. IEEE/ACM Int. Conf. Automated software engineering(ASE '10), pp.151-154, 2010.

[4] 村松裕次, 中川晋吾, 出口博章, 水野忠則, 太田剛, 酒井三四郎: リファクタリング箇所特定支援のためのパターン記述言語, 情報処理学会論文誌, Vol.46, No.12, 2005.