プログラミング課題採点支援のための プログラム差分抽出法に関する研究

2008MI041 平林 奈都季　　2008MI146 森 由貴　　2008MI254 塚本 恭子
指導教員　吉田 敦

1　はじめに

大学で行われているプログラミング実習の課題の採点では，プログラムの仕様の確認作業が行われている．プログラムの確認を何百人といる学生の数だけ手作業で行うことは，作業効率が悪く間違いの見落としが起こる可能性がある．その効率を上げるための手段として，模範解答と学生が提出したプログラムに対して差分抽出ツールを用いることがある．この 2 つのプログラムの比較に差分を用いると，模範解答通りに処理が記述されていない箇所を抽出できる．

既存の差分抽出法は，プログラム開発での 2 つの版間の比較に用いられ，構造がほぼ同一のプログラム間の比較に使用されることが多い．学生が提出したプログラムは，作成者が異なるので，プログラミングスタイルや同一の役割の識別子に異なる名前を付けるという特徴があり，構造が異なる場合がある．そこで，本研究ではプログラムの記述方法を揃え，異なる作成者のプログラム間での対応付けを行う．この対応付けの方法として，記述方法の統一がある．これにより，記述方法が異なることで抽出された差分が減り，より本質的な差分の抽出が望める．本質的な差分の抽出とは，比較したプログラム間で，処理が異なる箇所のみを抽出することである．

プログラムの記述方法を揃える手段として正規化がある．正規化により，記述方法が異なっても処理が同一となる箇所のみの差分が抽出でき，差分の量が減る．しかし，2 つの問題が存在する．1 つ目は，正規化を行うと，差分が正規化後のプログラムから抽出され，正規化前のプログラムのどこに差分結果が該当するのか採点者が理解しづらい．その原因として，差分抽出結果からは正規化前後のプログラムの対応関係がわからないことが挙げられる．2 つ目は，正規化を行っても，入れ子構造などにおいて差分が構造にまたがって抽出されることがある．構造にまたがった差分の抽出とは，例えば，対応しない左右の括弧が組となって差分に含まれることである．また，作成者が異なるプログラムは，関数名や変数名が同じでも，異なる処理で用いられることがある．

本研究は，プログラムの正規化と構造に対応した差分抽出を実現するために，正規化前後のプログラムの対応付けと正規化後のプログラム間の対応付けを行う差分抽出法を提案することを目的とする．差分を求めるアルゴリズムとして，最長共通部分列 (Longest Common Subsequence:LCS) を用いる．

記述方法の違いを吸収するためにプログラミングスタイルや繰り返しなどの構文を統一する正規化を行い，構造が似ている箇所や変数を対応付けて，正規化したプログラム間の差分を抽出する．正規化前後のプログラムの対応関係を表すために，正規化の際に識別の番号を字句に付け保持させる．求めた差分の中に，この番号を残すことで，採点者が正規化前のプログラムとの対応関係を知ることができる．正規化後のプログラム間の対応付けは，差分を用いることで対処する．

2　問題分析

2.1　基本技術

2 つの版間の差分の抽出には，diff コマンドが広く用いられ，変更，追加，削除が行単位で出力される．差分を求める際には，比較対象間に共通する最長の要素列 (LCS) を求め，LCS に含まれない部分が差分となる．

行単位の差分の出力は，意味に差のない部分も差分として出力され，記述の間違いの箇所以外も差分となる．よって，プログラムの本質的な違いを抽出するためには，字句単位の差分抽出が必要となる．

2.2　作成者が異なるプログラムの特徴

学生が提出したプログラムには，実習の課題の指示を満たさないものや文法的な誤りを含むもの，必要のない記述を含むプログラムがある．学生 15 人が記述した 90 個のプログラムを確認したところ，課題の指示を満たしていても次の 9 つの違いが見られた．

- スペースなどを含むプログラミングスタイル
- コメントの有無
- {} の有無
- プロンプトや出力の文字列
- 関数名，変数名
- 条件文，繰り返し文の記述方法
- 宣言数と宣言の順番
- 関数プロトタイプ宣言の有無
- 式の順序

課題ではプログラミングスタイルが強制されていない．本来，差分を用いる版間の比較では，プログラミングスタイルや識別子が一致していることが多く，プログラム間での大きな差がないことから，diff などの差分ツールを実用的に使える．しかし，作成者の異なるプログラム間で差分を抽出すると，同一の処理でも記述方法が異なれば差分として抽出する．よって，プログラムが本質的に異なる箇所以外も抽出される．

2.3　プログラム間の違いの吸収

処理が同一でも，記述方法が異なることで抽出される差分を減らすために，本研究ではプログラムを正規化する．作成者が異なることによって起こるプログラミングスタイルや識別子の違いを踏まえて，プログラムの正規化の処理としては次の 4 つを考える．

- プログラムの意味に影響がない部分の削除
- 未記述の字句の追加
- 識別子，記述方法，出力の文字列などの統一
- 文の並び替え

スペースや改行など，プログラムの意味に影響がないものは，削除を行っても比較に影響はない．また，関数名，変数名などの識別子や繰り返しなどの構文要素は作成者によって異なることが多く，異なる識別子や構文要素でも同一の意味を持つ場合がある．処理が同一であるプログラムの比較を行うために，識別子や記述方法を統一する．前述であげた 4 つの項目を行うことで，プログラムの作成者が異なっても，プログラムの構造を類似させ，差分を減らすことができる．

2.4 正規化の問題点

正規化後にプログラム間の比較を行うと本質的な差分の抽出が望める．しかし，正規化前後の対応関係が取れないこと，プログラムの構造に対応した比較が行えないことの 2 つの問題が存在する．

正規化後に差分を抽出すると，差分の結果では書換えが行われており，プログラムが書き換わった状態で差分抽出結果を得る．差分抽出結果が，正規化前のプログラムのどこに該当する差分であったのか理解しづらい．この原因は，正規化前後のプログラムの対応関係が取れておらず，正規化後の差分抽出結果が正規化前のプログラムで表せないことにある．

作成者が異なるプログラムは，図 1 のように，処理が異なるプログラムでも，似た構造を持つ場合がある．このプログラム間で，正規化後に字句系列を比較すると，構造にまたがった差分が抽出される．また，同一の関数名や変数名が使用されていても，その関数や変数が異なる処理で用いられる場合，正規化後のプログラム間でも，プログラムの構造に対応した比較が行えない．これにより，プログラムの構造が異なる箇所の差分が抽出される可能性がある．

3 作成者が異なるプログラム間の差分抽出法

3.1 対応付けの概要

作成者が異なるプログラムの比較の流れを図 2 に示す．この比較では，記述方法の違いを吸収するためのプログラムの正規化と構造に対応した差分抽出を行う．本研究で提案する対応付けは，次の 2 種類である．

- 正規化前後のプログラムの対応付け
- 正規化後のプログラム間の対応付け

これらの対応付けは，正規化によって書換えた字句の追跡と同一の処理に用いられる識別子の統一，構造にまたがった差分の抽出を避けるために行う．これにより，正規化後のプログラム間の差分抽出において，より本質的な差分を抽出できる．

3.2 プログラムの正規化

プログラムの書換えを行う環境として TEBA[3] が提案されている．TEBA は，プログラムを字句系列に分解し，その字句系列に構文上の属性などの情報を加えた書換え支援環境である．TEBA は，構文が完全でないプログラムに対しても解析でき，プログラムの書換えツールを字句系列のパターン変換として見通し良く実現できる．よって，本研究ではプログラムを TEBA によって解析し，得られた属性付き字句系列を書き換えてプログラムを正規化する．

3.3 正規化前後のプログラム間での対応関係

本研究では，プログラムの正規化後に差分を抽出する．しかし，正規化後の差分抽出結果からは，正規化前との対応関係が分からない．字句単位での正規化において，字句 1 つ 1 つの変化を理解することが難しくなる．よって，書換えと同時に対応関係を表すことが必要であり，字句に対してプログラム内のどこに存在していた字句なのか特定できるようにしなければならない．そこで，TEBA によって得られた属性付き字句系列に新たに字句番号を持たせ，それを保持して正規化する．

字句番号は，すべての字句系列に付加する．字句番号は，TEBA の字句系列が持つ識別番号とは異なり，先頭から番号付けしていく．ここで，TEBA の識別番号とは，組として使われる中括弧などの字句に対して，字句同士の対応関係を表す番号である．

TEBA を用いたプログラムの正規化は，書換えルールによって字句系列を書き換える．書換えルール上で，書換え前後で対応する字句に同一の記号を付ける．同じ記号を持つ字句は，同じ字句番号を持つように書き換える．正規化前後で同じ字句番号を持つ字句は，正規化前

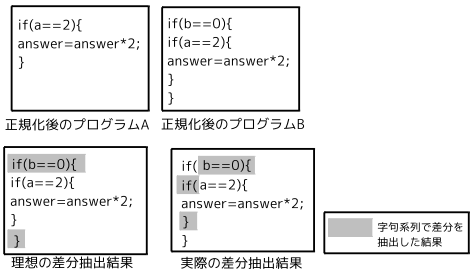

図 1　構造をまたがった差分が抽出される例

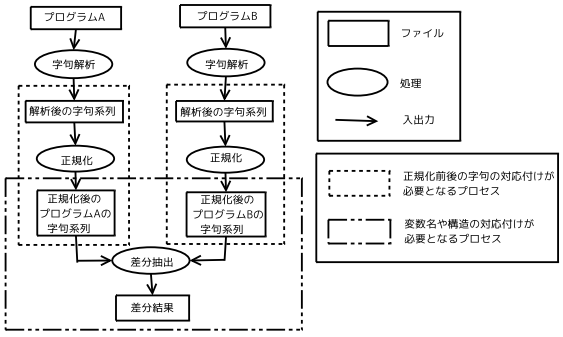

図 2　作成者が異なるプログラムの比較のプロセス

後において同一の意味を持つ．この字句番号を保持することで，正規化後の字句系列から抽出された差分が正規化前のどの字句に対応していたのかを表す．

3.4 正規化後のプログラム間の対応関係

比較するプログラム間の変数名や構造を対応付けには，2 つのプログラムの差分を抽出し，その結果からプログラム間の対応関係を見つける必要がある．差分抽出結果から対応関係があると判断する情報として，共通部分列がある．比較した 2 つのプログラムの字句同士が同一である場合，共通部分列として抽出される．共通部分列は，比較したプログラム間からそれぞれ共通となる字句を表しているので，各字句を組み合わせとして抽出でき，プログラム間の対応関係を表しやすくなる．よって，この共通部分列を利用し，変数名やプログラムの構造の対応付けを行う．

差分を抽出する前に，字句系列に対して，変数名や識別番号の除去を行う．変数名と識別番号の除去を行った後で差分を抽出することで，変数名や識別番号の違いを吸収できる．2 章で挙げたように，対応付けを行わずに差分を抽出すると，同一の処理でも名前が違うだけで変数が差分として抽出されたり，構造にまたがって差分が抽出されることがある．また，識別番号を含めて差分を抽出すると，同じ構造のプログラムが必ずしも同じ識別番号であるとは限らず，識別番号が異なる場合，差分が増える．変数名や識別番号の除去は，図 3 のように行う．差分を抽出するときの等価性の判定に用いる部分は図 3 の四角で囲まれた部分である．この部分において，比較した字句が同じだった場合，その字句同士は等価であるとする．等価性の判定に用いる部分から変数名を除去することで，種別だけで差分を抽出でき，比較した字句が共通部分列となる．

共通部分列から除去していた変数名をペアとして生成する．このペアは，共通部分列から作るので，その変数名同士に対応関係がある可能性があり，1 番多く生成されたペアの変数名が最も対応する可能性が高いと考えられる．対応する可能性が高いペアから対応付けを行い，変数同士が対応していることを表すために，変数名は統一する．変数名の統一が完了した字句は，図 3 で表した等価性の判定に用いる部分の中に変数名を戻し，再度ペアとして抽出されることを防ぐ．ペアの抽出がなくなるまで差分抽出を繰り返し，変数名の統一を行う．これにより，変数名が異なることによる差分が減少する．

変数名や構造の対応付けは 1 つの字句系列に対して対応付けているが，関数名の対応付けは，連続する複数の字句を対応付けなければならない．これに関して，関数ごとに差分を抽出し，その差分が少なかった関数の組み合わせは，対応関係があると判断する．

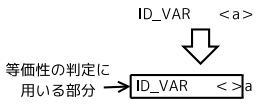

図 3　変数名の除去

4 評価と考察

4.1 評価基準

本研究では，次の 3 つを評価基準とする．

- 正規化前後のプログラムの対応がとれているか
- プログラムの正規化や構造の対応付けによって本質的な差分抽出結果が出力されているか
- 解答の種類がいくつに集約できるか

プログラムに対し正規化を行い，正規化前後の字句番号の対応関係を調査する．また，正規化後のプログラム間で差分を抽出し，本質的な差分が抽出できているか，構造にまたがった差分が抽出されていないかを確認する．さらに，正規化後のプログラムをグループに分ける．分けられたグループの数は，課題の採点に用いた際の，採点者が模範解答として用意する必要があると考えられるプログラムの数であり，集約されたプログラムの数が少なければ，正規化の効果があるといえる．

4.2 評価方法

評価に使用するプログラムは，15 人の学生が課題の解答として提出したプログラムであり，学生 1 人につき 6 個の課題のプログラムを用いる．各プログラムに対し，字句解析を行い，正規化後に差分を抽出する．学生が提出したプログラムに多く含まれていた違いを吸収するために次の 6 つの正規化を行う．

- 必要のないスペース，改行，コメントの削除
- {} の有無
- 複数宣言の分離
- 繰り返し，演算子の書き方の統一
- 変数名，関数名の統一
- 宣言文，関数の並び替え

これらの正規化後の差分抽出結果から正規化前のプログラムのどこが差分となるか確認する．その際に，正規化前のプログラム内に差分の箇所を色付けし，正規化後のプログラムとの対応関係を確認する．入力は属性付き字句系列とし，字句単位の差分を抽出する．

4.3 評価結果

4.3.1 正規化前後のプログラムの対応付け

正規化前後の字句の対応関係を確認するために，字句番号の数を確認した結果，90 個のプログラムすべての字句に対して字句番号が保持できた．

正規化の際に追加された字句に関しては，正規化前に存在していない字句であり，保持の対象とはなっていない．また，正規化では，スペースなどの字句以外が消えることはなく，正規化前の字句番号を保持できた．

4.3.2 プログラムの正規化と構造の対応付けによる差分抽出結果

正規化後のプログラム間で差分を抽出し，グループに分け，その結果を確認した．抽出された差分では，記述方法の違いを吸収でき，正規化の効果がみられた．しかし，条件分岐の違いや計算式の冗長な括弧など，処理が同じでも差分として抽出された．そこで，正規化の項目

を増やすことで，差分を減らすことができ，より本質的な差分の抽出が望める．評価結果から，正規化により解答の種類を集約でき，確認するプログラムの数を減らせた．集約ができなかった理由としては，記述の間違い以外にも，関数や条件の分け方の違いによって構造が大きく異なるプログラムがいくつか存在したことがある．

変数名の対応関係に関しては，約 9 割の変数名に対して対応関係を示すことができた．対応関係を示すことができなかった原因として，比較したプログラム間に学生の考え方や記述の誤りによる TEBA の解析結果の違いなどがあり，これらの影響を受けている変数名の対応関係を示すことは難しくなる．また，構造の対応付けに関しては，対応付けを行った括弧や仮想字句以外で，構文にまたがった差分が抽出されているプログラムがあった．しかし，このような差分抽出結果となったプログラムは 1 割と少なく，多くのプログラムで構造の対応付けによる効果を確認できた．

4.4 考察

構造の対応付けを行った字句以外に関して図 4 のように文にまたがって差分が抽出された．その原因として，差分抽出において LCS を用いていることがある．字句単位で差分を抽出した場合，差分となる字句の数が少なくなるように差分の判定を行っている．LCS により，構造の対応付けを行っても差分が少なくなれば，文にまたがって抽出される．この問題の対処法として，LCS アルゴリズムの改善や差分の判定における新たな判定基準の導入が考えられる．

本研究で提案した 2 つの対応付け方法は，等価である可能性が高い要素間を対応付けている．正規化前後の対応関係は，字句番号の付加により等価性を示している．また，正規化後の対応関係は，変数名や識別番号を等価性を判定に用いる部分から除去することで対応する字句を見つけている．対応する可能性が高い変数名同士や識別番号同士は，統一を行うことで対応関係を示している．現在は，各対応付けをそれぞれ行っているが，共通のフレームワークを用いることで，保守性を高めることができると考えられる．

5 関連研究

プログラムの差分抽出方法として，構文木を比較する方法 [1] があり，構文を考慮してプログラムの比較を行える．しかし，この構文木の比較では，あるノードが差分として判別された場合，その部分木がすべて差分となる．よって，間違った箇所のみを抽出できない．

既存の差分抽出ツールとして，Semantic Diff[4] がある．Semantic Diff は，プログラムの版管理で差分を抽出する際に用いられ，異なる版間の同一名の関数単位にステートメントリストを比較して差分を抽出する．正規化は行っていないが，差分を抽出する際に構造的等価性を用いて対応するステートメントの組を判定する．同じプログラムの変更箇所を比較しており，前提条件として識別子に違いがないものとしている．プログラム採点に用いた場合，文や宣言単位で変更箇所を見つけるので，同一の処理で記述方法が異なった場合でも差分となる．

プログラムの正規化に基づいた差分抽出法 [2] は，Semantic Diff の構造的等価性では吸収しきれない構造の違いをなくすために，プログラムの正規化を行っている．構文要素単位で差分を抽出しているが，変数名や関数名の統一，正規化の逆変換を行っていないので，プログラム採点に適さない．

本研究では，作成者の異なるプログラム間で見られる記述方法の違いを吸収することを重視して，正規化を行っている．その際に，正規化前後のプログラムの対応付けを行っており，正規化前後の対応関係が取れる．これにより，採点者に内容が理解しやすい差分を提示でき，差分抽出段階で構造の対応付けを行うことで，本質的な差分を抽出できる．

6 おわりに

本研究では，プログラムの正規化と構造に対応した差分抽出を実現するために，正規化前後のプログラムの対応付けと正規化後のプログラム間の対応付けを行う差分抽出法を提案した．作成者が異なるプログラムを比較して差分を抽出することを重視したが，構造の違いに対処仕切れないことがあった．今後の課題は，LCS アルゴリズムの改善や新たな差分の判定基準の導入方法を検討し，差分抽出の精度を上げること，構造の違いに対する更なる対処が挙げられる．

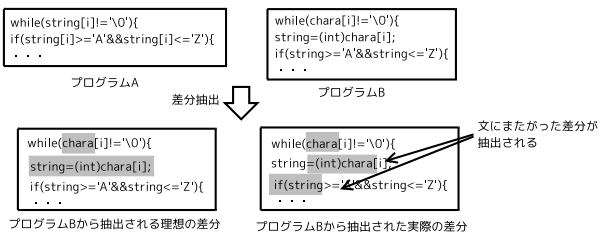

図 4　文にまたがって抽出される差分

参考文献

[1] Yang W., “Identifying Syntactic Differences Between Two Programs,” Software-Practice and Experience, vol21, no.7, pp.739-755, 1991.

[2] 尾崎憲幸，吉田敦，山本晋一郎，阿草清滋，“プログラムの正規化に基づいた差分抽出法の提案，” 情報処理学会研究報告．ソフトウェア工学研究会報告 99(37)，pp.25-32，May. 1999.

[3] 吉田敦, 蜂巣吉成, 沢田篤史, 張漢明, 野呂昌満, “属性付き字句系列に基づくプログラム書換え支援環境の試作，” ソフトウェアエンジニアリング最前線 (ソフトウェア・エンジニアリング・シンポジウム 2010 予稿集)，pp.119-126，Aug. 2010.

[4] 吉田敦, 山本晋一郎, 阿草清滋, “意味を考慮した差分抽出ツール，” 情報処理学会論文誌 38(6)，pp.1163-1171，Jun. 1997.