学習意図に合わせたコーディング校正ツールの提案

2010SE222　高桑　浩行　2010SE225　高島　豪人　2010SE229　竹腰　達徳
指導教員：蜂巣吉成

1　はじめに

学習者が演習などでプログラムを記述する際には，学習意図を満たし，プログラミング作法に適したコードを書くことを意識する必要がある．現在，大学等でプログラミングを学習する際には，与えられた課題に対してその課題を満たすプログラムを作成していく形式となっていることが一般的である．実行例と結果が同じであれば良いと考えて，課題を提出している学習者も存在しており，学習意図やプログラミング作法に適したコードを書こうという意識を持っている学生ばかりだとは言えない．

プログラミング作法に適したコードを記述しているかを調べるにはコーディングチェッカを用いる方法がある．コーディングチェッカにより一般的なコーディング規約に反した記述をしていないかを学習者に通知することができる．一方で，プログラミング言語の文法などを学ぶ課題に対して，課題の学習意図に合った記述をしているかは汎用的なコーディングチェッカで調べることは難しい．例えば，繰り返しを使った 2 次元配列の走査という学習意図に対し，行列を 2 次元配列を使って表現した課題を考える．行列の入出力や和の計算の課題では，2 重の繰り返しで配列を処理しているかを判定すればよいが，行列の積の課題では 3 重の繰り返しで 2 次元配列を処理しているかを調べなければならない．つまり，課題によって学習意図に適しているかの判定基準が異なる場合があり，汎用的なツールでチェックするのは難しい．ルールのカスタマイズ可能なコーディングチェッカを用いて，課題毎にルールを記述することも考えられるが，手間がかかり問題作成者の負担が大きい．

本研究では，プログラミング演習の課題では一般に問題作成者が模範解答プログラムを作成することに着目し，模範解答プログラムから学習意図にあったルール (以下，校正ルールと呼ぶ) を作成して, 学習者のプログラムをチェックして指摘する方法を提案する．学習意図とは，問題作成者が学習して欲しいと思う内容であり，文法や典型的な記述を学ぶために必要なものである．本研究ではコーディング校正とはプログラミング演習において学習意図に合わない記述を改善することと定義する．模範解答プログラムから校正ルールを作成するためのルールを学習意図ルールとし，学習意図ルールを模範解答プログラムに適用することで，校正ルールを生成する．過去のプログラミング演習課題を調べ，特徴的な学習意図を抽出し，それらを学習意図ルールとした．学習意図ルールと模範解答プログラムから校正ルールを生成するツールを試作し，学習者が記述した解答プログラムのチェックが行えることを確認した．問題作成者が新しい学習意図ルールをパターンとして記述することで新しい学習意図に対応することもできる．

なお，本研究で提案するツールは，プログラミング演習で学習者がプログラムを作成して，テストを行った後で，プログラムが学習意図に合っているかを確認するために使用されることを想定している，したがって，コンパイルに成功し，実行結果が正しいプログラムのみを対象とし，既存のコーディングチェッカと本研究で提案する校正ルールを併用することを前提とする．

2　関連研究

コーディングチェッカや学習者向きのフィードバックを行うツールはいくつか提案されている．CX-Checker[1] は，C 言語プログラムを解析してコーディング規約を満たしているかをチェックするコーディングチェッカである．MISRA-C[2] のコーディング規約をチェックでき，XPath などを用いてルールのカスタマイズもできる．

C-Helper[3] は，C 言語初学者向けの静的解析ツールである．インデントがされてない，関数定義にセミコロンが付いているといった初学者が陥りやすいミスを発見して指摘し，可能なら解決策を提案する．

proGrep[4] では学習履歴を収集し，その収集された情報から，学習者の特徴的な行動を抽出して，学習者の出来を判断したり，セミコロンがない，関数の返り値がないなどのアドバイスを与えたりできる．

C-Helper，proGrep は，学習者向けのコーディング規約などを用い指摘を行うツールなので汎用性が高いが，問題毎の学習意図を考慮した指摘などを行うことを目的としておらず，学習意図に対応したフィードバックを行うことはできない．

CX-Checker はルールのカスタマイズによって学習意図のチェックもできるが，問題ごとにルールを作ることが必要であり，手間がかかる．

3　学習意図に合わせたコーディング校正ルール

3.1　学習意図に合わせたルールの必要性

コーディングチェッカとはあらかじめ定められたコーディング規約や誤った記述のパターンに基づいてプログラムをチェックし, コーディング規約に違反したところや誤った記述のパターンに該当した箇所を指摘するツールである．コーディング規約や誤った記述のパターンはどのようなプログラムにも適用できる汎用的なものが多く，明らかに間違っている箇所を指摘したり，推奨しない書き方を直すためのものであることが多い．この仕組みやツールは初学習者が学習で用いる際にも一定の効果があり，明らかに間違った点やふさわしくない記述を指摘することができる．

プログラミング演習における演習問題にはそれぞれ学習意図があり，学習意図に適した記述をすることによって，文法や典型的な記述例を学ぶためのものである．一般的なコーディング規約には違反していない記述でも学習意図を満たしていない記述は演習問題の解答としては望ましくない場合がある．これらに対しても不適切な記述であると指摘することで，演習における学習効果を上げることができる．

例えば，2 整数の大きい値を返す関数 max2 を用いて，3 整数の最大値を関数 max3 を作成する演習問題では，例 1 のように 1 回だけ呼び出しても実行結果としては問題ないが，この問題には「関数を作成して適切に呼び出す」という学習意図があるので，例 2 や例 3 のように 2 回呼び出すことが望ましい．

(例 1)

```
1  int max2( int x, int y ){
2          if( x > y ){
3                  return x;
4          }else{
5                  return y;
6          }
7  }
8  int max3( int x , int y , int z ){
9          int max;
10         max = max2(x,y);
11         if( max < z ){
12                 max = z;
13         }
14         return max;
15 }
```

(例 2)

```
1  int max3( int x , int y , int z ){
2          return max2( max2( x , y ) , z );
3  }
```

(例 3)

```
1  int max3(int x, int y, int z){
2          int m2, m3;
3          m2 = max2(x, y);
4          m3 = max2(m2, z);
5          return m3;
6  }
```

汎用的なコーディングチェッカでは max3 で max2 を何回呼び出すべきなのかの判断ができないので，例 1 の記述を不適切な記述としてチェックすることは難しい．ある問題では学習意図を満たした記述でも別のある問題では学習意図を満たしていない記述になることもある．例えば，max2 を利用して 4 整数の最大値を求める関数 max4 では，max2 を 3 回呼び出すことが適切な記述になり，max3 と max4 では max2 の適切な呼び出し回数が異なる．学習意図を考慮したチェックを行うには，問題毎に学習意図を反映したルールを作成する必要があるが，問題に合わせて出題者がルールを作成するのは負担が大きい．

3.2 校正ルールと学習意図ルール

本研究では，プログラミング演習の課題では一般に出題者が模範解答プログラムを作成することに着目し，模範解答プログラムから学習意図を反映したルールを作成して，学習者のプログラムをチェックして指摘する方法を提案する．このルールを校正ルールと呼ぶ．一般に，模範解答プログラムは課題の難易度や学習に適しているかを確認したり，実行例の表示などのために作成され，出題者が想定した学習意図を反映したプログラムである．模範解答を用いることで指導者の負担を軽減できると考えた．

模範解答の制御構造などをそのまま抽出して校正ルールとすると，模範解答とほとんど同じプログラム以外は不適切としてチェックされる．例えば，関数 max3 の問題において模範解答が例 2 の場合，max2 の合成関数として校正ルールを作成すると，例 3 は不適切となり，ルールとしては厳しい．例 3 でも，関数を適切に呼び出すという意図には合うので，これを考慮して校正ルールを作成する必要がある．この場合は，関数 max3 内では max2 を 2 回呼び出すという校正ルールが考えられる．

本研究では，模範解答プログラムから校正ルールを作成するためのルールを用いることとする．このルールを学習意図ルールと呼ぶ．学習意図ルールは，学習者に学習して欲しいプログラムの特徴的な記述などを表現したものである．学習意図ルールを模範解答プログラムに適用して，その問題に合った校正ルールを作成する．

3.3 学習意図ルールの特徴

学習意図ルールを表現するために，過去のプログラミング演習の課題を調べ，その特徴を次のように整理した．

(a) 制御構造を用いた典型的な処理の理解
(b) 演算子の理解
(c) 関数の適切な利用方法の理解

(a) は，繰り返しによる典型的な配列走査などが挙げられる．一般に，1 次元配列は 1 つのループ，2 次元配列は 2 重ループで配列の各要素を参照することができる．しかし，2 次元配列による行列のかけ算の場合には，2 次元配列を 3 重ループで処理をすることが望ましい．

学習意図ルールとしては，制御構造と配列参照を記述したパターンを表現して，模範解答プログラムがこのパターンに適合したら，学習者のプログラムにも同じ構造があるかをチェックする校正ルールを作成するルールとして表現する．

学習意図は制御構造に反映されることが多いと考えるので学習意図ルールには制御構造の骨格を記述する．記述することができるのは制御文の if 文，while 文，for 文，do-while 文，switch 文である．対応した理由は過去の演習問題の調査により，これらの制御文が学習意図になっていることが多いからである．条件式に関わらず校正する校正ルールを生成するので，学習意図ルールを記述する際には，条件式の中には何も書かない．

学習意図ルールでは{ }を記述することによって入れ子構造を表現し，入れ子構造が異なるプログラムを区別することができる．以下が入れ子構造が異なるプログラムの例

```
1 for(){          1 for(){
2   for(){        2 }
3     for(){      3 for(){
4     }           4 }
5   }             5 for(){
6 }               6 }
```

である．配列を次元に応じて [ ]，[ ][ ]，[ ][ ][ ] という形式で記述することもできる．学習意図ルールの記述方法を拡張 BNF を用いて示す．

```
rule :: = stmt+
stmt ::= if-stmt|for-stmt|while-stmt|
  do-while-stmt|case-stmt|array-stmt
if-stmt ::= 'if'  '()'  '{'stmt*'}'
  ('else'  'if'  '()'  '{' stmt* '}')*
  ('else'  '{' stmt* '}')?
for-stmt ::='for'  '()'  '{' stmt* '}'
while-stmt ::= 'while'  '()'  '{' stmt* '}'
do-while-stmt ::='do'  '{' stmt* '}'  'while'  '()'
switch-stmt::='switch'  '()'  '{' stmt* '}'
case-stmt::='case' stmt*|'case' stmt* 'break'|
  'default' stmt* 'break'
array-stmt::='[]'+
```

以下が学習意図ルールの記述の例である．

下記は 3 重ループ文で 2 次元配列の処理を行っている例である．

```
1 for ( ){
2     for( ){
3         for ( ){
4             [ ][ ]
5         }
6     }
7 }
```

下記は 1 次元配列の最大値を求める処理の例である．

```
1 = [ ]
2 for ( ){
3     if ([ ]){
4         [ ]
5     }
6 }
```

(b) は剰余演算子 % を用いた処理などが挙げられる．例えば，文字を n 文字ずらすシーザー暗号の計算などに剰余演算を利用できる．

```
1 n = n % 26
```

しかし，学習者によっては % を用いずに余りを求める処理を記述する場合がある．

```
1 while (n>= 26){
2         n −= 26;
3 }
```

この記述の実行結果は正しいが，簡潔な処理を学んで欲しい．学習意図ルールとしては，特定の字句をパターンとして表現して，模範解答プログラムがこのパターンに適合した場合に，学習者のプログラムにも同じ構造があるかをチェックする校正ルールを作成するルールとして表現する．

(c) は 3.1 で挙げた関数を作成して適切に呼び出す処理である．プログラムによって呼び出す関数名や回数が異なるので，(a)(b) のようなパターンとして記述することは難しい．

学習意図ルールとしては，模範解答のプログラムで作成されている関数 (main 関数を除く) とその中で呼び出されている関数名と回数を抽出し，学習者のプログラムでも同様に呼出されているかをチェックする校正ルールを作成するルールとして表現する．関数呼び出しが適切かをより厳密にチェックするは関数の返り値や実引数などのデータフロー関係を調べることも考えられる．しかし，本研究では実行結果が正しい学習者のプログラムを対象としているので，データフロー関係は考慮せずに，呼出し回数でチェック行う．なお，呼び出されている関数には scanf 関数や printf 関数などの入出力関数は含めないこととする．これは，同じ結果を出力する printf 関数の呼出しは複数通り考えられ，必ずしも模範解答通りにする必要はないからである．

4 校正ツールの設計と実現

4.1 概要

本研究では従来のツールでは対応できない各問題の学習意図に応じた細かい指摘をすることができるコーディング校正ツールを提案し，既存のコーディングチェッカと本研究で提案する校正ツールを併用することでより学習に適した校正支援を行う．ツールの全体像を図 1 に示す．

校正ルール生成ツールに模範解答となるコードと学習意図ルールを入力し，コードを解析する．字句解析されたコードが学習意図ルールのパターンに適合した場合，校正ルールとして生成する．コーディング校正ツールに学習者のコードを入力し，解析する．そして校正ルールを満たしているか確認し，指摘箇所を表示する．

本研究では，実行結果が正しく出力されるプログラムに対して校正ルールを生成，確認することが前提なので，データフロー解析を行わなくても，制御文や構造等を確認することにより，校正ルールを生成，確認ができると考えたので，本研究では，字句解析が可能である TEBA[5] を利用した．

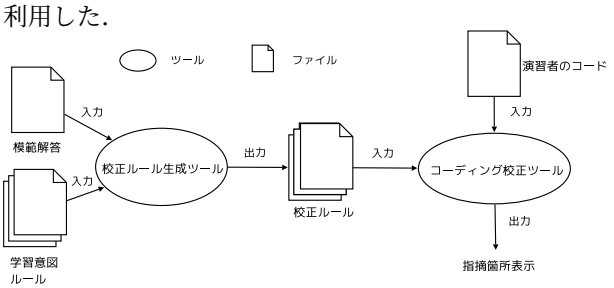

図 1 ツールの全体像

4.2 模範解答からの校正ルールの生成方法

模範解答のコードに 3.3 の (a)(b) の学習意図ルールから校正ルールを生成する方法は，模範解答のコードに TEBA

を利用し，字句解析を行い，3.3 の (a)(b) の学習意図ルールと同じ構造の場合，校正ルールとして生成する．3.3 の (c) の学習意図ルールから校正ルールを生成する方法は，模範解答のコードを TEBA で字句解析を行い，関数呼び出しのパターンの数を調べ，校正ルールとして生成する．

4.3 校正ルールによる学習者のコードの確認方法

学習者のコードが 3.3 の (a)(b) の校正ルールの確認方法は，学習者のコードを TEBA を利用し，字句解析を行い，3.3 の (a)(b) の生成した校正ルールと同じ構造の場合，校正ルールを満たしたと確認する．3.3 の (c) の校正ルールの確認方法は， 学習者のコードを TEBA を利用し，字句解析を行い，関数呼び出しのパターンの数を調べ，生成した校正ルールと関数呼び出しのパターンの数が同数の場合，校正ルールを満たしたと確認する．

4.4 実現と検証

校正ルール生成ツールとコーディング校正ツールを実装し，校正ルール生成ツールによって 3.3 で挙げた学習意図ルールと模範解答を用いて校正ルールが生成されることを確認した．その校正ルールを用いてコーディング校正ツールによって演習者のコードがチェックされることを確認した．校正ルール生成ツールは C 言語で約 120 行，コーディング校正ツールは C 言語で約 120 行である．

5 考察

提案した校正ツールは，模範解答プログラムの学習意図の記述に合わない記述を校正箇所として提示するが，意図に合わないプログラム (アンチプログラムと呼ぶ) を出題者が作成して，その記述に合った学習者の記述を校正箇所として提示する方法もある．学習意図に合わない記述をアンチ学習意図ルールとして表し，アンチプログラムに適用して，校正ルールを生成する．出題者にはアンチプログラムを作成する手間が増えるが，典型的な好ましくない記述があらかじめわかっている場合は，その記述を直接指摘することができる．(例 4) は成績評価を行うアンチプログラムの例である．else if の条件式が (score >= 70 && score < 80) となっているが，(score >= 70) とでも同じ実行結果が出るので，アンチプログラムは不必要な条件を書いている．この記述を学習意図ルールにすることにより，演習者が同様な記述をしたときに，指摘することができる．

(例 4)　成績評価を行うアンチプログラムの例

```
1 if( score >= 80 && score <= 100)
2           printf("A\n");
3 else if( score >= 70 && score < 80)
4           printf("B\n");
5 else if( score >= 60 && score < 70)
6           printf("C\n");
7 else if( score >= 0 && score < 60)
8           printf("F\n");
```

配列の繰り返しの学習意図について，今回試作したツールは繰り返しの中で配列を参照しているかのみをチェックしているが，カウンタ変数の初期値や終了条件もチェックすることで，典型的な配列走査を記述しているかを判定することができると考える．例えば下記 1，2，3 はどれも同じ実行結果となるが，1 の記述が望ましい．

1. for (i=0; i<size; i++)
   a[i]
2. for (i=0; i<=size-1; i++)
   a[i]
3. for (i=1; i<=size; i++)
   a[i-1]

また，3.3 の (c) は関数呼び出しのパターンの数から校正ルールを生成しており，呼び出す関数の順番は考慮していない．これは前提として実行結果が正しく出力されるプログラムが対象であり，関数呼び出しの数が模範解答と同数ならば，正しくプログラムが書けていると判断したからである．呼び出す関数の順番を考慮して校正ルールを生成するには，データフローを含む校正ルールにする必要がある．

本研究では，出題者に学習意図ルールを記述で作成してもらうことにより，学習意図ルールの数を増やしていくことができ，学習意図ルールを繰り返し利用できる．

6 おわりに

本研究では，学習者に問題毎の学習意図を満たしてもらうことを目的に，模範解答から校正ルールを自動生成し，学習者のコードが校正ルールを満たしているかを確認し，フィードバックを行うツールを提案し，試作した．今後の課題としては，学習意図ルールの追加をしていくことが挙げられる．

参考文献

[1] 大須賀俊憲，小林隆志，et al.：“CX-Checker：柔軟なカスタマイズが可能な C 言語コーディングルールチェッカー”, 情報処理学会論文誌， Vol. 53， No. 2， pp. 590-600， 2012.

[2] Motor Industry Software Reliability Association：“MISRA C3: Guidelines for the Use of the C Language in Critical Systems 2012”, Nuneaton, UK: MIRA, 2012. ISBN 978-1-906400-10-1.

[3] 内田公太，権藤克彦：“C-Helper:C 言語初学習者向け静的解析ツールの提案”, ソフトウェア工学の基礎 XIX 日本ソフトウェア科学会 FOSE2012，近代科学社， pp. 231-232， 2012.

[4] 長 慎也, 筧 捷彦：“proGrep-プログラミング学習履歴検索システム”, 情報処理学会研究報告．コンピュータと教育研究会報告，vol. 2005，NO. 15，pp. 29-36，2005.

[5] 吉田敦，蜂巣吉成，et al.：“属性付き字句系列に基づくソースコード書き換え支援環境”, 情報処理学会論文誌， vol. 53， No. 7， pp. 1832-1849， 2012.