# プログラミング演習における模範解答派生機能を備えた学習用校正ツールの提案

2011SE055 堀尾 美貴 2011SE112 金崎 真奈実 2011SE232 佐藤 成
指導教員：蜂巣吉成

## 1 はじめに

初学習者向けのプログラミング演習は，与えられた課題を満たすプログラムを作成する形式が多く，学習者は実行結果が実行例通りであれば良いととらえる場合がある．しかし，実行例通りに動作するコードが，指導者の教えたい教育意図とは異なる記述をしたコードである場合がある．

指導者が学習者のコードを確認し，教育意図と異なる記述を指摘する方法があるが，手間がかかり指導者への負担が大きい．指導者が作成した模範解答と学習者のコードを自動的に比較し，異なる部分を指摘する方法もある．しかし，学習者のコードが模範解答と少しでも異なると不適切なコードだと判断され，教育意図に合ったコードでも不適切だと判断される場合がある．

2013 年度の卒業研究 “ 学習意図に合わせたコーディング校正ツールの提案 ” [1] では，教育意図を含んだコードの判定方法を提案している．この研究で提案された方法には，3 つの問題点がある．1 つ目は，教育意図ルールで対応できる範囲が狭いことである．2 つ目は，教育意図ルールと少しでも異なると不適切な記述と判断されることである．3 つ目は，教育意図ルールでは不適切ではない記述でも，不適切な場合があることである．

本研究では，教育意図ルールの拡張と，派生パターンを用いた模範解答の派生を提案し，2013 年度研究での問題点を解決する．教育意図ルールを拡張することで，教育意図ルールの記述範囲を広くする．派生パターンを用い，1 つの模範解答から複数の模範解答を自動で派生させることで，不適切な記述を検出できる範囲が広がる．本研究で提案するツールによって，学習効率の向上と指導者の作業負担削減が期待される．

本研究で提案するツールの目的は，学習者のコードが教育意図を含んだ記述をしているかを判別するためであり，学習者のコードの正誤を判別することは目的としていない．そのため前提条件として，学習者のコードは指定された課題の実行例通りに実行できるものに限定する．

## 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．C-Helper[2] は，C 言語初学者向けの静的解析ツールで，初学習者が陥りやすいミスを発見して指摘する．可能なら解決策を提案する．proGrep[3] は，学習履歴を収集し，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， アドバイスを与える．CX-Checker[4] 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．これらのツールは初学習者向けではあるが， 一般的な記述の問題点を指摘している．そのため，プログラミング演習において，問題ごとの学習意図を考慮した指摘やアドバイスを与えることを目的としていない．

## 3 学習用校正支援

### 3.1 学習用校正支援の必要性

[1] では，学習者に学習毎の教育意図を満たしてもらうことを目的に，模範解答から教育意図に合った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どうかを確認し，フィードバックを行うツールを提案した．教育意図ルールの記述内容は，制御文 (for，if など) の構造や配列参照，関数呼び出しの回数である．例えば，2 整数の大きい値を返す関数 max2 を用いて，3 整数の最大値を返す関数 max3 を作成する演習問題では，制御の構造と関数の呼び出し回数を記述した教育意図により，例 1 のようなコードを正解と判断することができる．

(例 1)

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

しかし，例 1 をもとに教育意図ルールを作成すると，関数の呼び出し回数のみに着目するので，例 2 は同じ呼び出し回数として不適切と指摘されない．

(例 2)

```
1 int max3(int x,int y,int z) {
2     if (max2(x, y) < z) return z;
3     else return max2(x, y);
4 }
```

このような問題点を改善することにより，学習者の解答により良い指摘をすることができ，学習効果を上げることができる．

### 3.2 教育意図パターンの特徴

[1] では，以下の 3 つの教育意図に重点を置き，ルールとして表現していた．

(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，演習に合った制御構造かどうか，配列を使っているかどうかを判断するためのルールとして表現されていた．しかし，制御構造や配列の使用自体が教育意図ではな

い場合がある．例えば，例 3 の場合，for 文，配列を使って記述するという教育意図に加えて，一般的な初期化式，条件式，再初期化式の書き方を学んでほしいという教育意図もあると考える．一方，例 4 も実行結果は例 3 と同じになる．しかし，繰り返し条件が 1 から配列のサイズ以下までとなっているので，for 文内の実行文で配列の添え字に-1 をつけなければならなくなる．このような書き方は指導者の教育意図に合わないと考える．

(例 3)

```
1 for (i=0; i<size; i++)
2     printf(" %d \n ",a[i]);
```

(例 4)

```
1 for (i=1; i<=size; i++)
2     printf(" %d \n ",a[i−1]);
```

(c) は，関数呼び出しを使っているかどうかを判断するためのルールとして表現されていた．しかし，関数呼び出しを使っているが，それが適切な利用ではない場合がある．例として，3.1 節での例 1，例 2 がある．正しく判定できない例 2 のような場合も，対応できるようにする必要がある．それらをふまえて本研究では，(a) と (c) の教育意図を拡張する．(a) には繰り返し文の初期式や条件式など，適切な書き方がされているかについても追加し，(c) にはどの部分で関数が使用されているかについても教育意図として追加する．(a)，(b) は，教育意図パターンによって実現する．(c) は，派生パターンによる模範解答派生によって実現する．

教育意図パターンとは，模範解答から教育意図となる部分を抽出したソースコードであり，これを基準にして学習者の解答と比較するために必要となる．[5] を参考に，教育意図パターンの表現に次の 7 つを含める．

・制御構造 (if，for，while)
・配列の “[]”
・変数宣言の型名 (int，double)
・定数の値
・演算子 (比較演算子の “<”，“>”，“>=”，“<=” 以外)
・関数名と関数の “()”
・“ ; ”

制御構造と配列の “[]” は，演習に合った制御構造，配列の使用となっているかを調べるために必要となる．変数宣言の型名は，適切な型を使用した宣言となっているかを調べるために必要である．定数の値と演算子は，繰り返し文の初期式や条件式など，適切な書き方がされているかをチェックするために必要である．関数は関数呼び出しを行っているかを確認するために必要である．区切り記号は構文解析を行う上で必要な字句であると考えた．

例 3 では，配列操作を目的とした繰り返しが教育意図であるので，for 文の制御構造と配列の “[]” を表現に含めることで教育意図を含めることができる．また例 3 では，初期化式，条件式，再初期化式の内容が配列の添字に影響するので，配列操作の学習において，0 から始めて配列のサイズ未満まで繰り返すことが教育意図となる．定数の値と演算子を表現に含めることで，その教育意図を含めることができる．

変数は学習者によって宣言した変数名が異なるので，教育意図パターンの表現に含めない．比較演算子の “<”，“>”，“>=”，“<=” は，例えば if 文や while 文の条件式で “<” を使って書く場合，条件式の意味は同じで “>” を使った書き方も存在する．if 文や while 文ではどちらの場合でも一般的な書き方となり得るので，教育意図パターンの表現に含めない．

### 3.3 教育意図パターンを適用するルール

for 文では初期式の変数を条件式の左辺にすることが一般的であり，初期式の変数を条件式の右辺にする書き方は，for 文の書き方を学ぶ上では良くないと考える．for 文の条件式では，例外として比較演算子の “<”，“>”，“>=”，“<=” を教育意図パターンに含めるものとする．このように，教育意図パターンを適用するルールを条件によって変えることで，教育意図をより反映した教育意図パターンにすることができる．

例 3 を教育意図パターンで表現したものが例 5 である．演算子や定数を教育意図パターンに含めることで，例 4 のコードを不適切と判断することができる．

(例 5)

```
1 for(=0;<;++)
```

## 4 模範解答派生

### 4.1 模範解答派生の必要性

[1] で提案されたツールでは，模範解答と制御文の構造や配列参照，関数呼び出しの回数が同じならば教育意図ルールを満たすとされていたが，このルールと少しでも異なってしまうと不適切と判断されてしまう．また不適切であってもルールを満たしているならば適切となってしまう問題点がある．

(例 6)

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

[1] のツールでは，例 1 を教育意図ルールとして設定した場合，max2 を 2 回呼び出せば教育意図ルールを満たすとしている．[1] では例 2 と例 6 のような max2 を別変数に代入した場合でも max2 を 2 回呼び出しているので適切とツールは判断する．そこで模範解答の派生を行い，結果生成された例 6 を基準として追加することで，例 1 にも例

6 にも当てはまらない例 2 を適切でないと判定することが可能となる．

### 4.2 派生パターンの種類

派生パターンを次の 4 種類に分類する．
(a) 構文の派生
(b) 制御フローの派生
(c) データフローの派生
(d) 典型的な処理の派生
(a) は変数定義や初期化等の基本的な書式で使われる記述，つまりプログラムの構文の派生を指す．(b) は while 文や for 文といった制御フローの派生を指す．(c) は関数呼び出しなどのデータフローを考慮した派生を指す．(d) は配列走査やポインタ操作などのプログラム学習で頻出する処理の派生を指す．

## 5 設計と実装

### 5.1 設計

本研究では模範解答を元に模範解答の派生と教育意図パターンの生成を行うことでプログラミング学習を支援するツールを設計する．ツールの全体像を図 1 に示す．

本ツールに模範解答を入力すると，ツール内部で派生を行い，派生された模範解答それぞれの教育意図パターンを生成し保存する．その後，学習者が実装したプログラムと教育意図パターンのファイル名を入力すると学習者のコードを教育意図パターンと同じ書式に変換し，教育意図パターンと比較することで学習者のコードを評価する．

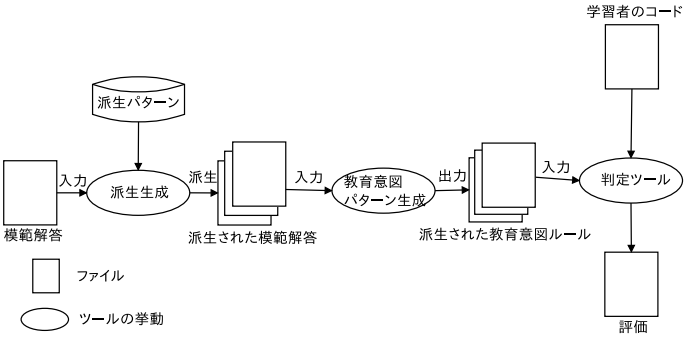


図 1 ツールの全体像

指導者が行う手順
1. 本ツールに模範解答のファイル名を入力する．
2. 1 度模範解答から生成された教育意図パターンを確認し問題が無ければ処理を続行させる．
学習者が行う手順
3. 本ツールに学習者が記述したコードのファイル名を入力する．
4. フィードバックを受け取る．

### 5.2 模範解答からの派生方法

本ツールに模範解答に当たる拡張子.c のファイル名を入力する．ツールが模範解答の構文解析を行い，模範解答から不要な要素を取り除いた教育意図パターンを一度ツール使用者に提示する．教育意図パターンが使用者が想定した内容であったら教育意図ルールとして保存する．その後，模範解答を派生させた結果それぞれの教育意図パターンを生成し保存する．

### 5.3 生成した教育意図パターンの確認方法

学習者のコードを元に教育意図パターンと同じ書式にしたファイルを構文解析し，結果に教育意図パターンの要素全てが含まれているかを確認する．この構文解析には TEBA[6] を利用して行う．要素の確認方法については教育意図パターンの要素について以下の情報が含まれているかを確認する．
・TEBA で定義された種別
・実際に記載された内容

### 5.4 派生パターンの適用方法

派生パターンの適用方法は TEBA を利用して行われる．構文解析を行った後，パターンで定義される書き換えるべきコード部分が存在していたら，変換を行い，派生された C 言語のソースコードを出力する．

### 5.5 派生パターンの例

典型的な処理の派生の一例として，受け取った配列型文字列をポインタに変換した場合を次に示す．
派生前：

```
1 int count(char a[]){
2   int cnt=0;
3   for(i=0;a[i]!='\0';i++){
4     printf("%c\0",a[i]);
5     cnt++;
6   }
7   return cnt;
8 }
```

派生後：

```
1 int count(char *a){
2    int cnt=0;
3    for(;*a!='\0';a++){
4     printf("%c\0",*a);
5     cnt++;
6    }
7    return cnt;
8 }
```

この派生は，for 文条件式の派生と printf 関数内部の派生を同時に行っている．例 7 が for 文条件式の派生パターンであり，for 文中の文字列を配列の添字で参照している箇所は TEBA の出力結果である属性付き字句系列をポインタの間接参照に書き換えるフィルタとして実現した．
(例 7)

```
%before
for(${i:ID_VF}=0;${cond:ID_VF}[${i}]!='\0';
${i}++){
  $[stmt:${:STMT}$]+
}
%after
for(;*${cond:ID_VF}!='\0';${cond}++){
  $[stmt]$;
}
%end
```

## 6 考察

同じ for 文の派生でも，演習問題によって派生して良い場合と悪い場合がある．累乗を求める例 8 では，計算の繰り返しによって望んだ値を求めることを目的としているので，例 9 のような for 文の記述でも適切となる．一方で例 3 のような配列操作を目的とした繰り返しでは，初期化式や条件式が配列の添字の記述に影響するので，例 4 は不適切となる．本研究では派生が適切かどうかの判断はツール上では行わず，派生後に教員が派生した解答を見て，適切なものを判断する．これを，ツール上で自動で判断する方法について考察する．

(例 8)

```
1 p = 1 ;
2 for (i=0; i<n; i++)
3     p = p*x;
```

(例 9)

```
1 p = 1 ;
2 for (i=1; i<=n; i++)
3     p = p*x;
```

for 文の派生の場合，カウンタ変数に着目する方法がある．カウンタ変数が for 文の本体に出現しなければ，カウンタ変数の値には依存せずに，回数だけが影響すると考えられるので派生が可能と判断する．例えば例 8 では，カウンタ変数の i が for 文の本体に出現しないので，例 9 への派生は適切な派生であると判断することができる．

教育意図パターンにおける変数宣言の抽出について，考察を行う．複数の変数宣言を行うにあたって，変数を一つの型にまとめて宣言，もしくは複数回に分けて宣言するかは個人によって異なる．しかし本研究で提案した教育意図パターンにおいて，模範解答の記述どおりに宣言をしなければ，適切な解答であっても不適切と判断をしてしまう．変数の数に応じて派生パターンを適用したとしても，問題によっては大量のパターンが生成されるという問題点がある．解決策として，変数宣言の抽出を指導者の判断によって行うことで，各学習者の解答への対応が可能となり，不必要な派生を防ぐことができると考えられる．

変数宣言の抽出による問題点はこの他にもある．オブジェクト形式マクロを利用した場合，変数宣言や for 文の記述など異なってしまう．出力部分での計算を行う場合と，変数を別に用意して出力部分外で計算を行っている場合も，教育意図パターンを適用するだけでは解答が適切かどうか判断できない．

## 7 おわりに

本研究では指導者の教えたい教育意図を反映し，模範解答派生により様々な学習者の解答に対応することを目的とした，学習用校正支援ツールを提案した．ツールの利用により学習効率の向上と指導者の作業負担削減が期待される．手法としては，指導者が用意した模範解答から複数個の解答を派生，それら解答から教育意図となる記述を抽出し，学習者の解答と照らし合わせフィードバックを行う．

今後の課題として，ツールの更なる検証，教育意図パターンの改善や派生パターンを追加することが挙げられる．

## 参考文献


[1] 高桑浩行，高島豪人，竹腰達徳，” 学習意図に合わせたコーディング校正ツールの提案 ”，南山大学情報理工学部 2013 年度卒業論文，2013.

[2] 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.

[3] 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.

[4] 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.

[5] 蜂巣吉成，吉田敦，阿草清滋：” プログラミング演習におけるコーディング状況把握方法の考察 ”，情報処理学会研究報告．コンピュータと教育研究会報告 2014-CE-125(3)，pp. 1-8，May，2014.

[6] 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.