判定木を用いたプログラミング学習用プルーフリーダの提案

2018SE011 原田知空　2018SE037 前川雅貴
指導教員：蜂巣吉成

1　はじめに

大学のプログラミング演習において，学習者は教育者から出題された課題に対し，実行結果が同じになるプログラムを作成することで，学習を進める．しかし，学習者は実行結果が同じになるプログラムを作成すればよいと考え，教育者の意図（以下，教育意図とする）を考慮しないプログラムを作成する場合がある．この場合，教育者が教育意図を指摘するには，学習者のプログラムを一つ一つ確認する必要があるので，教育者の負担は大きくなる．

我々の研究室では，学習者のプログラムが教育意図を満たしているか確認するために模範解答のプログラムと比較して判定するプログラミング学習用プルーフリーダを提案している [1][2]．プルーフリーダでは，教育意図を学習項目ごとに分類し，学習者の解答が教育意図を満たしているか判定するツールを提案している．教育意図が記述されている箇所を意図パターンと呼び，意図パターンを用いて，教育意図を反映したコード断片を評価コードと呼ぶ．模範解答から抽出された評価コードと学習者の解答から抽出された評価コードを行単位の差分を比較して，教育意図を満たしているか判定を行っている．この研究で提案されたプルーフリーダでは，「判定に用いる箇所が限定的であり，教育意図を反映したものではない場合がある」，「模範解答が複数存在する場合の処理が考慮されていない」といった 2 つの問題点がある．

本研究の目的は，学習者が教育意図を満たした解答を作成しているか判定すること，上記で述べた過去年度のプルーフリーダの問題点を解決することである．過去年度のツールの問題点を解決するアプローチとして，課題ごとに教育者がプログラムをどのような基準で正誤判定を行っているかを考察し，これを基に判定したい要素（以下，判定要素とする）を抽出し，判定木を用いて判定を行う．判定木とは，ノードに判定箇所，エッジのラベルに判定要素の値を持ったラベル付き n 分木である．

2　関連研究

コーディングチェッカー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[3] は C 言語初学者向けのコーディングチェッカで，初学者が陥りやすいミスに対してわかりやすいメッセージで指摘し，解決策を提案している．しかし，作成されたプログラムが教育意図を考慮した判定は行えない．

CX-Checker[4] は，コーディング規約のカスタマイズ機能を備えた C 言語プログラムのコーディングチェッカである．XPath や DOM ，ラッパを用いて単純なコーディング規約から複雑なコーディング規約まで定義することができ，独自のチェックを行うことができる．しかし，教育意図を満たしているか判定を行うには，課題毎に教育意図に沿ったコーディング規約を定義する必要があり，教育者の負担が大きい．

proGrep[5] は，プログラミング教育で学習者のプログラムがコンパイル，実行されたタイミングで学習履歴を収集する．収集したデータから学習者の特徴的な行動をパターンとして抽出し，学習者にアドバイスを行う．しかし，作成されたプログラムが教育意図を満たしているか判定は行えない．

1 節で示した問題が発生する例として累乗を求めるプログラムを示す．この課題の教育意図は「繰り返しで計算する変数の初期化を繰り返し前に行い，適切な値で初期化する」，「for 文を典型的に繰り返す（n+1 や n-1 といった記述をしない）」である．学習項目を for 文（初期化式有，条件式無）と設定する．ソースコード 1，2 は累乗を求めるソースコード，ソースコード 3，4 はそれらの評価コードである．

ソースコード 1　累乗を求めるソースコード 1

```
1 pow = 1;
2 for (i = 0; i < n; i++){
3   pow = pow * x;
4 }
```

ソースコード 2　累乗を求めるソースコード 2

```
1 pow = 1;
2 for (i = 0; i <= n-1; i++){
3   pow = pow * x;
4 }
```

ソースコード 3　ソースコード 1 の評価コード

```
1 $var=1;
2 for(;;){
3 $var=;
4 }
```

ソースコード 4　ソースコード 2 の評価コード

```
1 $var=1;
2 for(;;){
3 $var=;
4 }
```

ソースコード 1 は 0 から n 未満まで，for 文を繰り返しているので，教育意図を満たしている．それに対して，ソースコード 2 では for 文の条件式に n-1 といった記述がみられるので不適切な解答である．学習者が実行結果を見て繰り返し回数を調整すると，ソースコード 2 のような不適切な記述をすることがある．過去年度のツールは，ソースコード 1，2 から評価コードの抽出を行う（ソースコード 3，4）．学習者の解答と模範解答の評価コードを行単位

で差分を比較し，学習者のソースコードが教育意図を満たしているか判定する．しかし，ソースコード 3，4 の評価コードは一致しているので，ソースコード 2 を適切であると判定する．また，累乗を求めるプログラムの模範解答として，ソースコード 1 の他にカウンタ変数を 1 ずつ減分させる繰り返しを用いたプログラムも挙げられる．複数の模範解答が存在する場合の扱いが十分に考慮されていない．

3　判定木を用いたプルーフリーダの提案

3.1　概略

1 節で問題点として，判定に用いる箇所が限定的であり，教育意図を反映していない場合があることを挙げた．この問題を解決するために本研究では，教育者の判断基準から抽出した要素を用いたプルーフリーダを提案する．

教育者は演算子や変数間の関係性などを見ながら解答を判定しているので，教育者の判定基準に基づいた箇所を抽出し，それを基に判定を行えば教育意図を満たしているか厳密に判定できる．模範解答から抽出した判定要素と判定順序から判定木を生成する．模範解答が複数存在する場合，判定要素が同じ値なら，ノードとエッジを共有し，異なるなら新たにノードとエッジを追加することで判定木を生成する．この操作を最後の判定要素まで繰り返し行う．これによって模範解答が複数存在する場合でも，1 つの判定木を用いて表現でき，判定できる．

3.2　教育者の判断基準

ここでは教育者がどのように解答を判定しているのか（判断基準）について述べる．

累積計算を行うプログラムを例に教育者の判断基準を示す．ソースコード 5 は「合計を求めるプログラム」，ソースコード 6 は「階乗を求めるプログラム」である．

ソースコード 5　合計を求めるソースコード

```
1  sum = 0;
2  for (i = 1; i <= n; i++){
3    sum = sum + i;
4  }
```

ソースコード 6　階乗を求めるソースコード

```
1  fact = 1;
2  for (i = 1; i <= n; i++){
3    fact = fact * i;
4  }
```

合計，階乗，累乗などを求めるプログラムの教育意図は「繰り返しで計算する変数の初期化を繰り返し前に行い，適切な値で初期化する」，「for 文を典型的に繰り返す」である．教育者は解答を判定する際に for 文内の文で使われている演算子に着目する．累乗，階乗を求めるプログラムでは*が使われている．また合計を求めるプログラムでは + が使われている．そのため pow，fact は 1 に，sum は 0 に初期化することが適切である．次に for 文内の文で使われている被演算子に着目する．累乗を求めるプログラムでは x なので，カウンタ変数 i を 0，1 どちらで初期化しても実行結果が正しく，教育意図を満たしている．階乗を求めるプログラムは演算子が*で被演算子がカウンタ変数である i なので，変数 i を 1 で初期化しなければならない．

3.3　判定要素の抽出

前節で示した教育者が行っている判定を自動で行うために，教育者が解答を判定する際に着目している箇所について定義する．

本研究では，教育者が着目している箇所を判定要素とする．合計，階乗，累乗のような累積計算を行うプログラムから「ans_{init}：初期化」，「for_{init}：for 初期化」，「for_{comp}：比較演算子」，「for_{end}：継続条件」，「for_{cnt}：カウンタ変数の更新」，「ans_{op}：演算子」，「ans_{opr}：被演算子」を判定要素とする．ソースコード 7 は累積計算を行うプログラムを一般化したものである．

ソースコード 7　累積計算の一般化

```
1 ans = ans_init;
2 for (i = for_init; i for_comp for_end; for_cnt){
3   ans = ans ans_op ans_opr;
4 }
```

ソースコード 1，5，6 の判定要素を表 1 に示す．

表 1　累積計算の判定要素

判定箇所	判定要素名	合計	階乗	累乗
ans_{init}	初期化	0	1	1
for_{init}	for 初期化	1	1	0
for_{comp}	比較演算子	<=	<=	<
for_{end}	継続条件	n	n	n
for_{cnt}	カウンタ変数の更新	++	++	++
ans_{op}	演算子	+	*	*
ans_{opr}	被演算子	i	i	x

3.4　判定木

ここでは前節で述べた判定要素を用いて判定を行う際に利用する判定木について述べる．ソースコード 7 で定義した判定要素の名前と値の組と教育者が正誤判定を行う判定順序を用いて判定木を生成する．本研究で判定木とは，ノードに判定箇所，エッジのラベルに判定要素の値を持ったラベル付き n 分木である．模範解答が複数存在する場合でも，複数の模範解答から判定要素を抽出して，判定木を合成するので，制御構造が同じならば模範解答がいくつあってもよい．判定木は，判定箇所と判定要素の値，判定順序から生成することで，教育者の判定基準に沿った判定を行うことができる．教育者の模範解答から抽出した判定要素から判定木を生成し，学習者の判定要素を判定木と比較することで学習者の解答を判定する．累乗の模範解答にはソースコード 1 に加え，for 文の条件式が for(i=1;i<=n;i++)，for(i=n;i>0;i−−)，for(i=n;i>=1;i−−) なども存在し，それらから判定要素を抽出し，生成した判定木を図 1 に示す．

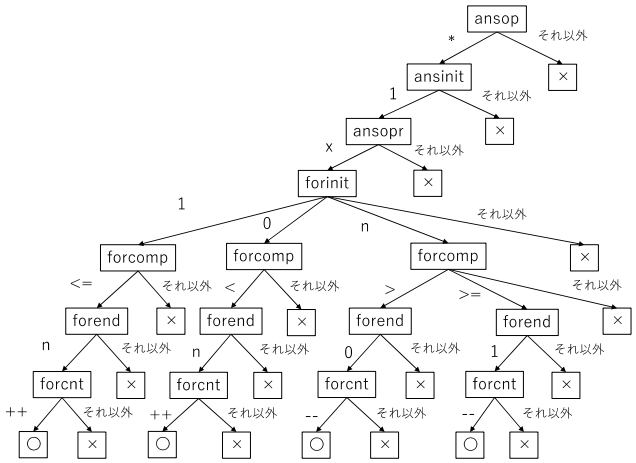

図 1　累乗の判定木

4　判定木を用いたプル―フリーダの設計と実現

4.1　設計

本研究で提案するプルーフリーダは，大学のプログラミング演習で使用されることを想定して設計を行った．C ソースファイルから判定要素を抽出し，正誤判定までのプルーフリーダの全体像を図 2 に示す．全体の処理の流れを以下で説明する．

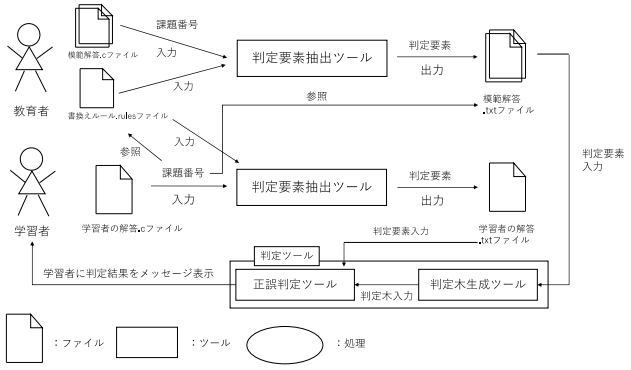

図 2　ツールの全体像

初めに，教育者が模範解答ファイル名，課題番号，教育意図を反映した判定要素を取り出すための書換えルールファイルを判定要素生成ツールに入力する．判定要素生成ツールでは，入力された模範解答から課題番号がファイル名となった判定要素が記述されたテキストファイルを出力する．

次に，学習者が学習者の解答のファイル名，課題番号を判定要素生成ツールに入力する．入力した課題番号に対応する書換えルールファイルを判定要素抽出ツールに入力すると，学習者の解答の判定要素が記述されたテキストファイルが出力される．課題番号に対応する模範解答の判定要素のテキストファイルを判定木生成ツールに入力し，模範解答の判定木を生成する．模範解答の判定木と学習者の解答の判定要素のテキストファイルを正誤判定ツールに入力し，学習者の解答が教育意図を満たしているか判定を行う．

4.2　判定要素抽出ツール

判定要素抽出ツールでは模範解答，学習者の解答どちらも対象とする．C ソースファイルを判定要素抽出ツールに読み込ませることで判定要素の名前と値が 1 行ずつ記述されたテキストファイルを抽出する．

判定要素を抽出する方法について述べる．まず判定要素を抽出したいソースコードに対して正規化を行う．正規化では，{} の有無などの実行結果には影響しない記述の揺らぎを統一する．

判定要素は TEBA[6] の書換えルールを用いて抽出する．書換えルールは，ソースコード 8 のように 1 つ目の波括弧内に書換えたい字句列を，2 つ目の波括弧内に抽出したい判定要素の字句列を記述して書換えルールを作成する．教育者が書換えルールを記述する．抽出したい判定要素に判定順序を記述することで linux の sort コマンドで判定要素を判定順序に並べ替える．

ソースコード 8　書換えルール 1

```
1  {
2    書換えたい字句列
3  } => {
4    抽出したい判定要素の字句列
5  }
```

4.3　判定木生成ツール

判定木生成ツールでは，模範解答から判定木を生成するツールである．3.2 節で示した n 分木の判定木を実装上では，2 分木で表現する．

模範解答から抽出した判定要素のテキストファイルを読み込み，判定要素の名前と値を 1 行ずつリストに格納する．模範解答の判定要素を格納したリストを先頭から順に判定箇所を持つノード，判定要素の値を持つエッジを作成し，結合する．この操作を最後の判定要素まで繰り返し行い，判定木の葉を生成し，末端のエッジに生成した葉を結合することで判定木を生成する．また，模範解答が複数存在する場合は，模範解答の判定要素を格納したリストを先頭から順に既に生成した判定木のエッジの値と比較する．値が等しければ次の判定要素，ノード，エッジに進む．値が異なれば現在の判定要素とそれ以降の判定要素で判定木を生成し，既に生成されている判定木に結合する．この操作を複数ある模範解答すべてに行い判定木を生成することで 1 つの判定木で複数の模範解答に対応できる．

「階乗を求めるプログラムを作成しなさい」という課題の模範解答と想定されるプログラムから判定木を生成した結果を図 3 に示す．図 3 では×となるノードを省略している．

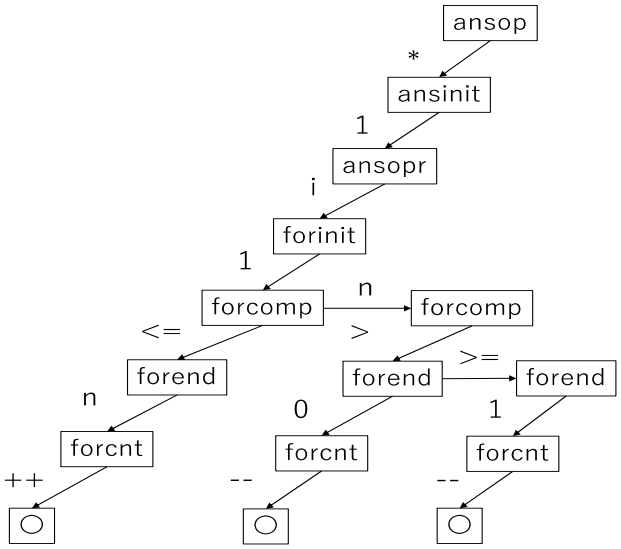

図 3 階乗の判定木の実現

4.4 正誤判定ツール

正誤判定ツールでは判定木を辿り，学習者の解答が教育意図を満たしているか判定を行うツールである．

学習者の解答から抽出した判定要素のテキストファイルを読み込み，判定要素の名前と値を 1 行ずつリストに格納する．学習者の解答の判定要素を格納したリストの値を先頭から順に模範解答で生成した判定木のエッジの値と等しいか比較を行う．値が等しければ次の判定要素，ノード，エッジに進み，同様の条件判定を繰り返し行う．判定要素の値が判定木のエッジの値と最後まで等しければ，○の葉に到達するので，学習者の解答は教育意図を満たす解答であると判定する．判定要素の値が判定木のエッジの値と異なれば模範解答の別解の判定木のエッジの値との比較を始める．値が等しければ次の判定要素に進み，模範解答の別解の判定木のエッジの値との比較を進める．判定要素の値がすべての別解の判定木のエッジの値と異なれば学習者の解答は教育意図を満たさない解答であると判定する．

4.5 評価

学習者を想定して合計，累乗，階乗を求める累積計算のプログラムと配列の最大値，最小値を求めるプログラム，九九の表を表示するプログラムを作成した．本ツールを利用して作成したソースコードの判定を行ったところ正しく判定することができた．

5 考察

5.1 書換えルールの記述による負担

本研究では課題毎に問題を一般化し，抽出したい要素を書換えルールとして記述することで，判定要素の抽出を行った．教育者が書換えルールを記述するには，TEBA で定義されている独自の字句や文法を習得する必要がある．制御構造が複雑な場合，書換えルールが複雑になり，記述量が増加する．教育者が課題ごとに書換えルールを記述することは負担が大きいので，単純な記述で書換えルールを作成できるようにすることが今後の課題である．

5.2 学習者へのフィードバック

本研究では，学習者の解答が教育意図を満たしているか判定を行ったが，判定木を用いて教育意図を満たす，満たさない，実行結果が正しくない解答の 3 段階に分けて評価を行うことが考えられる．図 1 では○と×の 2 つだが，×はさらに実行結果は正しいが教育意図を満たしていない記述，実行結果が正しくないものに分類される．×のノードに対してその親の判定要素に対するメッセージ，例えば，「累積計算の演算子が不適切」，「累積計算の初期化が不適切」，「繰り返し条件の比較演算子が不適切」などを対応づければ，わかりやすいフィードバックを返すことができる．

6 おわりに

本研究では，学習者のソースコードが教育意図を満たしているか教育者が行う正誤判定の判定基準に基づいた判定木によるプログラミング学習用プルーフリーダを提案した．模範解答と学習者の解答から教育者が正誤判定をする時に見る箇所を判定要素として抽出する．教育者の判定要素から判定木を生成し，学習者の判定要素から学習者のソースコードが教育意図を満たしているか判定することができた．今後の課題として，書換えルールの改良やフィードバック方法の提案などが挙げられる．

参考文献

[1] 清水祐輔，加賀弘晃，松井敦紀：“カスタマイズ可能なプログラミング学習用プルーフリーダの提案”，南山大学情報理工学部 2016 年度卒業論文 （2017）．

[2] 後藤悠太，長谷優磨，田原寛隆：“教育意図を利用したプログラムのプルーフリーダの提案”，南山大学情報理工学部 2015 年度卒業論文 （2016）．

[3] 内田公太，権藤克彦：“C-Helper：C 言語初学習者向けツール C-Helper の予備評価”，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] 大須賀俊憲，小林隆志，ほか：“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] 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）．

[6] 吉田敦：“RewriteTokens Ver.2”，TEBA 関係資料，http://tebasaki.jp/doc/RewriteTokens2-Manual.html，2021–11–21，（参照 2022–01–11）．