Vol.2021-CE-162 No.16
Vol.2021-CLE-35 No.16
2021/12/4

抽象構文木を利用したプログラミング理解度採点の試み

漆原 宏丞[1,a)]　本多 佑希[2]　岸本 有生[3]　兼宗 進[1]

概要：プログラミングの提出課題について、学習者の理解度を評価する手法を提案する。従来から、成績評価やプログラミングコンテストにおいて自動採点の技術が開発されてきた。それらの多くはプログラムの実行結果のみを確認する手法が用いられており、複数の入力データについて正しい結果が出力されることを確認するが、変数名の間違いなどの単なるケアレスミスも不正解と判定してしまう問題があった。また、教員が目視でソースコードを確認することも行われているが、教員が想定していない正解プログラムについては評価が難しいという問題があった。そこで本研究では、「提出された課題プログラムの構文木を分析することでプログラムの構造を抽出し、適切な考え方でプログラムが記述されていることを確認する」評価手法を提案する。過去の学生プログラムを分析したところ、教員による目視等の評価に近い判定を行うことができた。

キーワード：プログラミング教育, 自動採点, 抽象構文木

1. はじめに

小学校から高等学校まで広くプログラミングが必修化され、教員は生徒の理解度を把握することが求められている。出力を機械的に確認する自動採点手法では、手間が少なく生徒のプログラムを採点できる一方で、「構造も理解できていない」「構造は理解しているが単純なケアレスミスで不正解になった」など生徒がどの段階にいるのかを判別することは難しい。そのため、理解度の把握においては理解がどこまで進んでいるのか、どこで躓いているのかといった情報は重要であるが、一方で教員によるソースコードの目視確認は、教員に対する負担が大きい。

本稿では、生徒の理解度を評価する手法として、生徒のプログラムの構文木からプログラムの内部で使われている構造を検出し、課題で意図した考え方でプログラムが記述されているかを確認するシステムを提案する。従来の出力値のみを確認する採点手法と併用することにより、「正解で適切な構造を使っている」「正解だが適切な構造を使っていない」「不正解だが適切な構造は使えている」「不正解で適切な構造も使えていない」など詳細に判別することができる。

[1] 大阪電気通信大学
Osaka Electro-Communication University, Neyagawa, Osaka 572-8530, Japan
[2] 四天王寺大学
Shitennoji University
[3] 大阪電気通信大学高等学校
Osaka Electro-Communication University High School
[a)] eh16a015@oecu.jp

2. 既存の理解度評価手法

プログラムの正誤判定には、出力値を確認する手法と、ソースコードを確認する手法が存在する。機械的な自動採点では、多くの場合で出力値を確認するアプローチが取られており、ソースコードの評価は教員による目視で行われる場合が多い。出力値を確認して正誤判定を行う例としては、パソコン甲子園 [1]、情報オリンピック [2]、Aizu Online Judge[3]、AtCoder[4] などのプログラミングコンテストが挙げられる。この手法では、入力値とそれに対応する正解となる出力値のセットを複数用意し、全ての入力値のパターンで正解が得られるかを確認することでプログラム内の処理が正しいかを確認する。この手法は仕組みが単純であるため扱いやすく、広く用いられているが、正解、不正解は判定できるものの、理解度を段階的に評価することには向いていない。コンテスト用のシステムを利用したり、同等のシステムを開発した授業も報告されている [5][6][7]。課題プログラムを実行して入力値と出力値を調べる形のブラックボックステストを行う採点システムは、Bit Arrow[8][9][10] をはじめ、いくつか報告がある [11][12][13][14]。また、ブラックボックステストとソースコードの検査を組み合わせた方式 [15] やコードの類似度から盗用を判別する試み [16] も提案されている。

ソースコードを分析して評価する方式としては、理解度の段階的な評価を図る手法が提案されている。De-

Gapper[17][18][19] は、学習者が繰り返し文を苦手とすることに注目し、プログラミングの理解に必要な学習ステップの内容に、新しい課題が入っているかを調べるためのツールである。これらは、正しい順番で学習課題が作成されているかを調べることができる。このように学習のステップを定義し、生徒が作ったプログラムがどのステップに相当するかを確認することで、理解度を段階的に評価することができる。

ソースコードから抽象構文木を生成し、抽象構文木のレベルで正答プログラムと比較したり、正誤判定を行うシステムが提案されている [20][21]。また、プログラムの採点に抽象構文木と機械学習を利用したツール [22] も提案されている。機械学習は有用性が高いが、正誤判定をするには多くの学習データが必要となる。

3. 設計

3.1 概要

前章で述べたように、プログラムの理解度を評価するために多くの手法やシステムが提案されている。我々は、この中でも抽象構文木を確認する手法に着目した。プログラムから生成された抽象構文木の部分木を確認することで、そのプログラム内で使われている制御構造を検出することができる。そのため、問題で意図した構造がプログラム内で使われているかを確認すると、制御構造の理解度を測ることができるのではないかと考えた。

3.2 評価の手法

課題のプログラムの中で適切な構造が使われているかを検出するために、提出されたプログラムから抽象構文木を生成する。この構文木を確認することで、プログラム内部で用いた構造を確認することができる。

例として、図 1 に Python で書かれたリストから最大値を求めるプログラムを、図 2 にそのプログラムから生成した構文木の例を示す。プログラムから生成された構文木はこのように巨大であるため、本稿では、説明のために図 3 のように簡略化した構文木を用いる。

```
a=[10, 8, 33, 12, 25]
m = a[0]
for x in a:
  if x > m:
    m = x
print(m)
```

図 1 Python で for 文を用いて最大値を計算するプログラム例

3.3 必要な機能

今回提案する手法をシステムとして動作させる場合、以下のような流れになる。

(1) プログラムから抽象構文木を生成する
(2) 問題が意図する抽象構文木を生成する（場合によっては、教員が正解とする木を指定する）
(3) プログラムから生成された構文木の中に、正解となる構文木が部分木として含まれているかを確認する

そのため、これら 3 つの機能を検討・実装することにした。

4. 実装

本研究では、抽象構文木を作るにあたって Python 標準の ast ライブラリを使用する。

4.1 関数名の参照

提出されたプログラムの抽象構文木に必要な構造を表す部分木が含まれていることを調べれば、構造の確認は概ね可能である。しかし、どの関数がどこで呼ばれているかが重要になるケースもある。例えば、「数を入力し、それらの合計を表示せよ。0 の入力で終了とする」という問題があった場合、「while の中で入力を受け取っているか」「それを数値に変換しているか」を見る必要がある。そのため、図 5 のように、関数呼び出しを行う「Call」という名前に、呼び出された関数名を付け足すことで対応した。

4.2 反復処理の統合

反復処理が必要な課題において、「for 文を使え」のような条件が無い問題もある。その場合、教員は for 文の使用を想定していても、生徒が while 文を使って解答するケースが考えられる。例えば「1 から入力値 n までの偶数のみを表示せよ」という問題の場合、for 文や while 文の条件指定は無い。しかし、教員が for 文での解答を想定していた場合は、while 文を使って解答した生徒は不正解となってしまう。そこで、「Loop(For)」のようなノード名にした。for 文でも while 文でも良い場合は、この「Loop ノード」を指定できるようにし、前述の問題に対応した（図 5）。

4.3 判定用の部分木の指定

詳細な記述ルールは検討中であるが、現時点では正規表現を用いる方針で考えている。再び「リストから最大値を探す問題」を例に考えるが、必要な構造は「For → If → Assign」を含む部分木である。これを正規表現で書くと「`".*(For)(If)(Assign).*"`」のようになる。この部分木の指定にあたっては以下の方法を考えている。

- 問題文から自動生成する方法
- 教員が必要だと思った部分木を手入力する方法

4.4 部分木が含まれていることを調べる

正規表現にマッチするかどうかで部分木の有無を調べる。そのために、まずは根ノード（今回は Module ノード）から各葉ノードまでの部分木の構造を全て文字列で列挙す

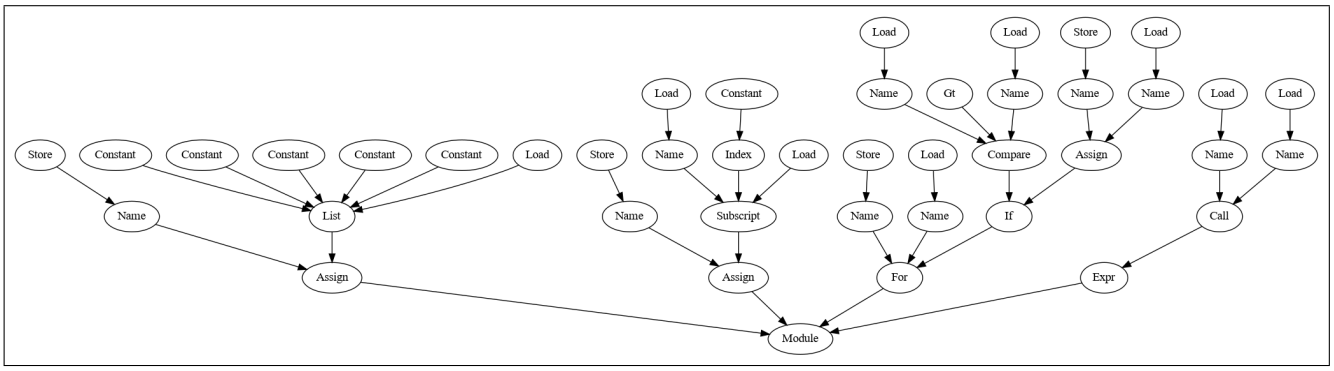

図 2 図 1 のプログラムから生成した構文木の例

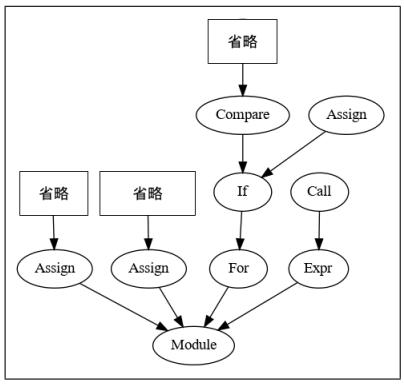

図 3 図 3 を簡略化した構文木の例

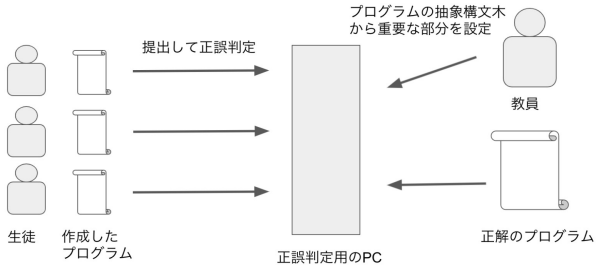

図 4

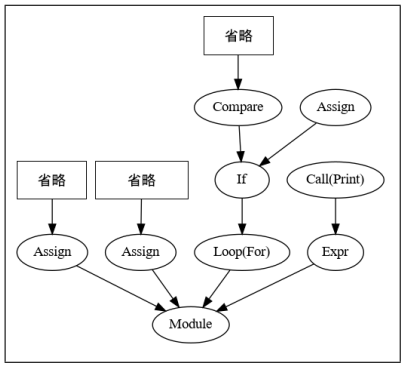

図 5 呼び出される関数名を付け足した例

る。例として、図 2 の部分木を列挙したものが図 6 である。これらに対して、前述のような必要な部分木を表す正規表現を適用することで、必要な構造が含まれているかを調べる。必要な部分木の指定は複数個も可能であり、指定した部分木全てが含まれていれば、その提出プログラムは正解となる。また、マッチしなかった部分木が表す構造については理解が足りていないことになるので、再び解説などを行うことで生徒らの理解度を高めることも可能である。

```
Module Assign Name Store
Module Assign List Constant
Module Assign List Constant
Module Assign List Constant
Module Assign List Constant
Module Assign List Constant
Module Assign List Load
Module Assign Name Store
Module Assign Subscript Name Load
Module Assign Subscript Index Constant
Module Assign Subscript Load
Module For Name Store
Module For Name Load
Module For If Compare Name Load
Module For If Compare Gt
Module For If Compare Name Load
Module For If Assign Name Store
Module For If Assign Name Load
Module Expr Call Name Load
Module Expr Call Name Load
```

図 6 図 3 の根ノードから葉ノードまでの経路一覧（一部）

5. 実験と評価

5.1 実験に使用した課題

システムの有効性を確認するため、工学部の大学 2 年次に開講されるプログラミングの授業課題（表 1）と、その受講生の提出プログラムを使用する。なお、この授業は Python で行い、実行環境はオンライン学習環境の BitArrow を使用していた。

5.2 本システムによる正解率

各課題ごとに筆者が解答例と正規表現パターンを作成し、そのパターンにマッチする割合（本システムによる正解率）を調べた（表 2）。その結果、0621_9 のマッチ率が特に低いことが分かった。提出されたプログラムを見ると、

表 1 今回使用した授業課題の問題文

課題番号	問題文
0607_2	for と if を組み合わせて、0 から 19 の間の、6 の倍数だけを表示せよ
0614_2	円の半径 r を入力し、円周を表示せよ。円周率は 3.14 とする
0614_3	円の半径 r を入力し、面積を表示せよ。円周率は 3.14 とする
0614_4	数を入力し、偶数か奇数かを表示せよ
0614_5	数を入力し、80 以上なら「A」、80 未満で 60 以上なら「B」、60 未満で 40 以上なら「C」、40 未満なら「D」を表示せよ
0614_8	数を入力し、それらの合計を表示せよ。0 の入力で終了とする
0614_9	数を入力し、1 からその数までの整数の合計を表示せよ
0621_2	下のプログラムの 4 行目と 5 行目の print 文のインデントを変更して、実行例の出力が表示されるプログラムを作れ （実行例）a b c a b c c c
0621_3	下のプログラムの 4 行目と 5 行目の print 文のインデントを変更して、実行例の出力が表示されるプログラムを作れ （実行例）a b a b c
0621_4	下のプログラムの 4 行目と 5 行目の print 文のインデントを変更して、実行例の出力が表示されるプログラムを作れ （実行例）a b c a b c b c b c
0621_5	下のプログラムの 4 行目と 5 行目の print 文のインデントを変更して、実行例の出力が表示されるプログラムを作れ （実行例）a b a b b b c
0621_6	下のプログラムの 4 行目と 5 行目の print 文のインデントを変更して、実行例の出力が表示されるプログラムを作れ （実行例）a a b c
0621_8	「a=[10, 8, 33, 12, 25]」でリストを作成し、for と if で偶数の要素を表示せよ
0621_9	「a=[10, 8, 33, 12, 25]」でリストを作成し、平均値と最大値を求めよ
0621_10	「a=[ ]」で空のリストを作成し、for と randint を使い、5 個の 1 から 50 までの乱数値をリスト a に追加して表示せよ
0628_3	入力された番号の生徒の成績を合計して表示せよ（生徒の成績は授業資料に掲載）
0628_4	入力された科目の成績を合計して表示せよ（国語は 0、数学は 1）（生徒の成績は授業資料に掲載）
0628_5	引数 n の二乗を返す関数 nijou() を定義せよ
0628_6	半径 n の円の面積を返す関数 en() を定義せよ。円周率は 3.14

Python の組み込み関数である max 関数や sum 関数を使っていることが分かった (図 9)。しかし、この課題は「for 文を使って合計値を求める」「for 文を使って最大値を探す」

```
for i in range(4):
  if i<2:
    print("a", end=" ")
    print("b", end=" ")
    print("c", end=" ")
```

図 7 表 1 の 0621_2〜6 で使われるプログラム

「求まった合計値をリストの要素数で割る」ことが分かるかを問うた課題だと思われる。よって、max 関数などを使った解答は、たとえ実行結果が合っていても不正解と判定されるべきであり、この判定結果は妥当だと考えられる。また、0614_9 も 59%と比較的低めであるが、こちらは図 10 のような不完全な解答が不正解と判定されており、問題なく判定できていた。

5.3 教員の採点結果との比較

教員の採点結果と基準とし、本システムの採点結果がどの程度合っていたか比較し、表 3 にまとめた。これを見ると課題 0628_4 の割合が特に高いことが分かる。この不一致の詳細を調べると、教員の採点では正解だが、本システムでは不正解となっているプログラムばかりであった。この問題は「キーボードから入力を受け取っている」「受け取ったデータを整数型に変換している」「for 文か while 文を使って合計値を計算し、都度変数に代入している」構造が含まれていれば正解と設定した。よって、図 11 のようなプログラムは、本システムでは不正解となる。しかし、教員の採点では正解となっていたことが要因である。

5.4 本システムによる採点の範囲

本システムで抽象構文木を生成するとき、Python 標準の ast ライブラリを使用している。その関係で、エラーとなるプログラムは抽象構文木にすることができないので、必要な構造が書けていても不正解となってしまう。また、「本当は不正解なのに正解だと判定されたプログラム」がある。例えば、課題 0628_5「引数 n の二乗を返す関数 `nijou()` を定義せよ」の場合、「関数定義の中で二乗の計算を行っている」ことが必要になる。加えて、その日の課題の前提条件として「キーボードから入力した値を引数として計算すること」となっていた。本システムにおいては、これらを以下の正規表現でチェックした。

(1) `".*(Iput).*"`
(2) `".*(Int).*"`
(3) `".*(FunctionDef).*((Mult)|(Pow))"`

図 8 のようなプログラムは、これら全ての正規表現にマッチする構造である。しかし、実際に計算しているのは定数の 3 であり、入力された n ではない。こういったケースを正しく判定するには変数名などの情報が必要であり、本システムでは原理的に対応が困難である。

```
x=int(input('number:  '))
def nijou(n):
  return x*x
print(nijou(3))
```

図 8 課題 0628_5 で不正解なのに正解と判定された例

表 2 本システムによる正解率
（受講生 79 名）

課題番号	正解率
0607_2	84%
0614_2	85%
0614_3	89%
0614_4	73%
0614_5	81%
0614_8	71%
0614_9	59%
0621_2	84%
0621_3	82%
0621_4	84%
0621_5	80%
0621_6	82%
0621_8	80%
0621_9	41%
0621_10	80%
0628_3	78%
0628_4	67%
0628_5	87%
0628_6	86%

表 3 教員の採点結果と本システムの採点結果の比較
（提出者 79 名）

課題番号	違っていた割合
0607_2	3%
0614_2	1%
0614_3	1%
0614_4	5%
0614_5	5%
0614_8	8%
0614_9	5%
0621_2	5%
0621_3	3%
0621_4	3%
0621_5	5%
0621_6	4%
0621_8	5%
0621_9	6%
0621_10	8%
0628_3	6%
0628_4	19%
0628_5	5%
0628_6	3%

```
a=[10, 8, 33, 12, 25]
ave = sum(a)/len(a)
print(ave)
print(max(a))
```

図 9 課題 0621_9 の題意に沿わないがプログラム自体は正しい例

```
print("数値は？")
a=input()
G=0
i = 0
while i<a:
  i = i + 1
  G=i+G
print(G)
```

図 10 課題 0614_9 の不正解判定の例

```
scores=[
  [64, 72, 58, 69, 63],
  [77, 84, 79, 82, 80],
  [72, 93, 79, 88, 89],
  [92, 97, 85, 81, 90],
  [75, 73, 83, 90, 81]
]
x=int(input('student:  '))
sum=0
for i in range(5):
  print(scores[i][1])
```

図 11 課題 0628_4 で教員の採点では正解だが本システムでは不正解になった例

6. 今後の展望とまとめ

生徒のソースコードから構文木を生成し、その部分木を確認することで、制御構造の理解度を測る手法を提案した。この手法を用いることで、変数名の間違いや単純な計算ミスなど、問題で想定している制御構造が使えているかとは関係がない部分を無視することができる。

実際に検証したところ、教員の採点と違う結果になったのは平均して 5%だった。受講生は 79 名だったので、約 75 名程度に対して妥当な判定を行えていることがわかる。採点を担当した教員に意見を求めたところ、「コードを目視する形の採点はミスも起きやすいため、このようなツールで大まかに確認したあとで、誤答と判定された課題だけを目視することで、採点の効率を高めつつ、目視だけよりも採点の精度を高められるかもしれない」とのコメントをもらうことができた。

今回は手法の提案と妥当性の検証に留まったが、今後は実際に学校の授業で活用するために必要な機能の検討や実装などを進めていきたい。

参考文献

[1] パソコン甲子園. https://web-ext.u-aizu.ac.jp/pc-concours/
[2] 情報オリンピック. https://www.ioi-jp.org/
[3] Aizu Online Judge. https://judge.u-aizu.ac.jp/onlinejudge/
[4] AtCoder. https://atcoder.jp/
[5] 倉田英和, 富永浩之, 林敏浩, 垂水浩幸. 実行テストによるプログラム判定を用いた初級 C プログラミング演習支援と授業実践. 情報処理学会研究報告コンピュータと教育 (2007-CE-091), pp.11–18, 2007.
[6] 松永賢次. 導入プログラミング教育におけるオンラインジャッジシステムの活用の試み. 情報科学研究, 31, pp.25–41, 2011.
[7] 坂本一憲, 鷲崎弘宜, 深澤良彰. プログラミング初学者向けコンテストシステム. 日本ソフトウェア科学会第 30 回大会, pp.1–8, 2013.
[8] 長島和平, 長慎也, 間辺広樹, 兼宗進, 並木美太郎. Web ブラウザを用いたプログラミング学習支援環境 Bit Arrow の設計と評価. 情報処理学会論文誌「教育とコンピュータ」, 2017.
[9] 長島和平, 堀越将之, 長慎也, 間辺広樹, 兼宗進, 並木美太郎 プログラミング学習支援環境 Bit Arrow の教員支援機能の設計と試作. 情報教育シンポジウム論文集, 2017(18), pp.129–136, 2017.
[10] 堀越将之, 長島和平, 長慎也, 兼宗進, 並木美太郎. プログラミング学習環境 「Bit Arrow」 における採点支援機能. 研究報告コンピュータと教育 (CE), 2018(9), pp.1–8, 2018.
[11] 松本真吾, 野中美希, 太田剛, 酒井三四郎. 教師が作成したテストケースを用いたプログラムの正誤判定によるプログラミング学習支援システム. 教育システム情報学会誌, 26(1), pp.29–35, 2009.
[12] 石原俊, 田口浩, 高田秀志, 島川博光. マークアップによる C 言語プログラミング試験採点システム. DEWS2007, 2007.
[13] 島袋舞子, 小林史弥, 林康平, 兼宗進. プログラミング課題・作品評価補助システムの提案. 研究報告コンピュータと教育 (CE), 2016(14), pp.1–9, 2016.
[14] 小林史弥, 林康平, 島袋舞子, 兼宗進. プログラミング課題評価補助システムの提案. 2016 年度 情報処理学会関西支部 支部大会 講演論文集, 2016.
[15] 内藤広志. ヒューリステックを用いた, プログラミング演習用の効率的な自動採点システム. 第 66 回全国大会講演論文集, 2004(1), pp.345–346, 2004.
[16] 和田修平, 井上潮. 盗用発見と自動採点によるプログラミング演習課題の評価支援システム. In DEIM Forum Vol.2011, 2011.
[17] Yayoi Hofuku, Shinya Cho, Tomohiro Nishida, and Susumu Kanemune: Why is programming difficult? Proposal for learning programming in “small steps” and a prototype tool for detecting “gaps”. ISSEP2013, Oldenburg, Germany, 2013.
[18] 保福やよい, 西田知博, 長慎也, 兼宗進. なぜプログラミングは難しいのか？ 繰り返しの理解構造と C の教科書分析からのアプローチ. 情報処理学会, 情報教育シンポジウム (SSS2012), 2012.
[19] 長慎也, 保福やよい, 西田知博, 兼宗進. De-gapper - プログラミング初心者の段階的な理解を支援するツール. 情報処理学会論文誌, Vol.55, No.1, pp.1-12, 2014.
[20] 小山昂紘, 原田史子, 島川博光. 階層構造化による C 言語ソースコードの自動採点. In IEICE Conferences Archives. The Institute of Electronics, Information and Communication Engineers, 2012.
[21] 小川弘迪, 小林亜樹. プログラミング課題の自動採点に向けた構文木上のカーネル法による類似度関数の提案. 第 80 回全国大会講演論文集, 2018(1), pp.661–662, 2018.
[22] 竹末裕, 竹内 和広. プログラミング問題における多様な回答の可視化. 情報処理学会, 情報教育シンポジウム SSS2019, 2019.