D-001

プログラムトレース学習支援システムのための問いの自動化手法の提案
A method to automate questioning for a program trace learning support system

立岩 佑一郎† 東本 崇仁‡ 赤倉 貴子*
Yuichiro Tateiwa Takahito Tomoto Takako Akakura

1. はじめに

近年，小学校教育におけるプログラミングの必修化やプログラミング的思考の重要性への気づきが高まり，プログラミング教育に関する需要が高まっている．これらの教育やプログラミング的思考において重要なこととして，単にプログラムの言語的な理解ではなく，アルゴリズムやプログラムの振る舞い（動作の仕方）を理解させることが挙げられる．一方で，プログラミング初学者は，「for 文は複数回繰り返させる」などといったなんとなくの理解が多く，それゆえ for 文の条件が繰り返し条件なのか終了条件なのかといった理解が曖昧であったり，ループを抜けた際のイテレータの状態がどうなっているかなどを十分に理解ができていないことがある．

このように漠然としたイメージではなく，しっかりと振る舞いを理解させるためには，東本らが提案してきたようなプログラムトレース課題（プログラムの実行順序と，各行で起こる処理を記述させる課題）[1]が重要となる．また，東本らはプログラムトレース課題に対して，適切な順序でトレースした場合と，学習者の作業内容にそってトレースした場合のそれぞれを，出力内容と変数の内部状態という２つの観点から可視化（フィードバックと呼ぶ）するプログラムトレース学習支援システムを開発した．学習者は，正解に基づく可視化と自身の作業に基づく可視化の差分を考えることで，自発的に誤りに気付くこととなる．

一方で，このシステムにおけるフィードバック内容は，あらかじめシステム作成者側で問題ごとに用意されており，用意した問題に対してのみしかフィードバックを提示することはできない．今後のより汎用的な利用を考えた場合，多くのソースコードに対応したフィードバック方法の実現が望まれる．そこで，本研究では，任意のソースコードに対して，このようなフィードバックを実現する方法を検討する．

2. プログラムトレース学習支援システム

2.1 問い機能の概要

システムの問いの機能は，学習者へ最初の問いを提示する．そして，学習者からの応答を受け取り，応答に基づいたフィードバック（例えば，式の評価値）を表示し，応答に基づいた問いを提示することを繰り返す．

図 1 はシステムの問いの実行画面例である．領域 α はプログラムの行を選択するための領域であり，選択された行が領域 β に選択された順に上から表示される．領域 γ および領域 δ は問いおよびフィードバックを表示する領域である．また，ボタン ε は一つ前の問い（実行される行を問う）に戻るボタンである．

この実行画面は，システムが実行される行を問い，ユーザが領域 α の 4 行目のボタンをクリックした直後のものである．システムは領域 β の末尾にクリックされたボタンに対応した行を表示し，領域 γ と領域 σ に新たな問いを表示する．この問いに対してユーザが「1」を入力すると，システムが「1 % 2（1 を 2 で割った余り）」の計算結果（フィードバックに該当）と次の問いを表示する（図 2）．図 2 は，図 1 の領域 δ の表示である．「1 % 2」の計算結果が「1」であることを意味し，プログラムの第 4 行目の条件式「i % 2 == 0」を満たすかどうかを問うている．

この機能の実行のためには，教師がプログラムを解析して問いの対象（上述の例では，実行される行や変数 i の値）を見つけ，規則に則って問いの実施順序（上述の例では，実行される行を問うた後に変数 i の値を問うという順序）と問いの実施条件（上述の例では，実行される行の問いへの解答が第 4 行目であることからの変数 i の値の問い）を決定し，それらをシステムへ設定する必要がある．また，フィードバックの生成に用いるプログラムのコード（上述の例では，ユーザからの入力 i において「i % 2」の値を求めるコード）を問いの機能の実装コードへ埋め込む必要もある．これらの作業は教師にとって負担が大きいため，本研究では，問いの自動化手法の開発を目的とする．これにあたり，まず，問いの動作を拡張有限状態機械で定義する．そして，その状態機械をプログラム解析ツールで得られる情報およびデバッガを用いて実現する．

2.2 問いの対象

プログラムトレース学習支援システムは，下記の項目を問う．

- 行の実行順序
- 式の中の変数の値と，関数の引数（引数を式とみなす）
- ユーザ定義関数の呼出先と，その関数からの復帰先
- ユーザ定義関数の実行開始時の引数の値

なお，式は下記の要素からなるものに限定される．

- 演算子（問いの対象外）
- 変数（問いの対象）
- 関数（ユーザ定義関数であれば，戻り値が問いの対象）
- 定数（問いの対象外）

2.3 問いの流れ

問いは次に示す手続き Q に従って行われる．ただし，各問いにおいて一つ前の問いに戻ることができる．

1. 次に実行されるユーザ定義関数 *ud* を問う．最初は main 関数とする．
2. *ud* の引数の値を第一引数から順に問う．

† 名古屋工業大学, Nagoya Institute of Technology
‡ 東京工芸大学, Tokyo Polytechnic University
* 東京理科大学, Tokyo University of Science

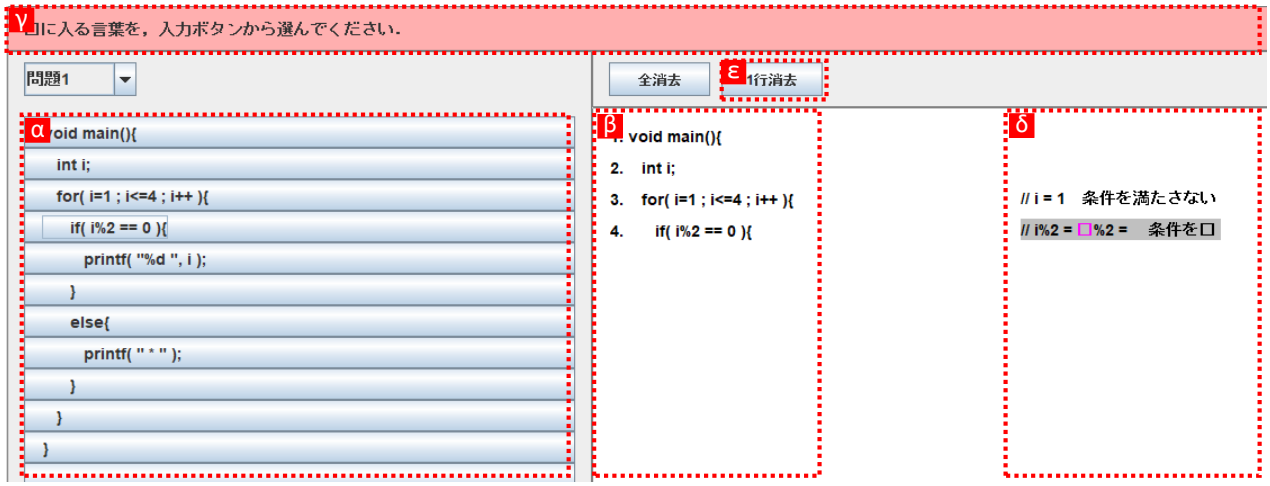

図 1　問いの実行画面例

3. 次に実行される行 *l* を問う．
4. *l* に含まれる式 *e* が条件式であれば両辺の要素を問い，そうでなければ右辺の要素を問う．
5. *e* が条件式であれば評価結果を表示し，そうでなければ右辺の評価結果を表示する．
6. *l* がユーザ定義関数の末尾（"}"）でなければ手順 3 へ戻る．
7. *ud* からの復帰先を問う．

ここで，手順 4 は下記のように実施される．手順 ii におけるユーザ定義関数の問いに対する答えにより，手順 5 の *e* が手順 3 の *e* と異なることがある．

i. *e* の最初に評価される変数または関数を *vf* とする．
ii. *vf* を下記のように問う．
- *vf* が変数であれば，変数の値を問う．
- *vf* がライブラリ関数であれば第一引数から順に問い，戻り値を表示する．
- *vf* がユーザ定義関数であれば第一引数から順に問い，手続き *Q* を実行する．そして，*vf* の戻り値を表示し，*Q* で指定された復帰先に基づいて *e* と *vf* を更新する．

iii. *vf* の次に評価される変数または関数が存在すれば *vf* として手順 ii へ戻り，そうでなければ終了する．

3. 実現法

システム実現における重要点として，1) 行の構成要素の問い（第 2.3 節の手順 2，手順 4，手順 5）の生成，2) 実行される行の問い（手順 1，手順 3，手順 7）と，その行の構成要素の問いとの連携が挙げられる．本稿ではこの 2 点の実現法を述べる．

3.1 必要なデータと関数

下記に示すもののうち，関数 Eval 以外はプログラムの静的解析により求められる．関数 Eval の実現法を第 3.4 節に述べる．

- プログラムに含まれる関数定義の集合を *FD* とする
- プログラムに含まれる式の集合を *E* とする．便宜上，ユーザ定義関数の末尾（"}"）も式とする．
- プログラムに含まれる呼出元関数の集合 *CR*
- 式 *e* に含まれる変数の集合を *V(e)*とする

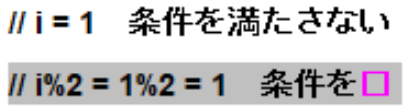

図 2　フィードバックと問いの表示

- 式 *e* に含まれる関数の集合を *F(e)*とする
- 関数 $f \in F(e)$, $e \in E$ の引数を構成する式のリストを *A(f)*とする．先頭から順に第一引数，第二引数．．．と格納されている
- 関数定義 $fd \in FD$ の引数を構成する変数のリストを *AD(fd)*，先頭から順に第一引数，第二引数．．．と格納される
- 定義が *FD* に含まれている関数 $f \in F(e)$, $e \in E$ の集合を *UF* とする
- 式 *e* における $x \in V(e)$または $x \in F(e)$を評価順に並べたリストを *EO(e)*とする
- 関数から復帰する式 $e \in E$ の集合を *RT* とする．
- 式 *x* または関数 *x* の評価結果を返す関数を *Eval(x)* とする．

3.2 システムの動作

システムの動作を拡張有限状態機械 $M=(Q, \Sigma, \Delta, ini, H)$ で定義する．

- Q：状態集合
- Σ：$q \in Q$ への入場時に実行される関数 $\sigma(q)$の集合
- Δ：状態 q においてユーザからの入力を受け取り，条件 α が満たされれば，関数 γ を実行し，状態 q'へ遷移することを四つ組$(q, q', \alpha, \gamma) \in \Delta$で表す．ここで，本システムでは Δ の定義において次の変数を用いる．状態 q においてユーザから入力*(ope, val)* を受け取る．ユーザが次の状態へ進めたいとき *ope=NEXT* であり，一つ前の状態へ戻りたいとき *ope=PREV* となる．また， *val* はユーザが入力した文字列である．
- *ini*：初期状態
- *H*：状態 *q* と文字列 *val* において，*(q, val)*を要素とするリスト．本システムでは，遷移した状態を *q* で

保管することで，現状態から初期状態まで一つずつ戻るために利用される．また，ユーザからの入力値や評価結果を *val* で保管することで，式や関数を評価する状態がその値を評価に利用できるために利用される．

3.3 M の作成方法

3.3.1 準備

M の各要素の作成方法は，下記の表記を用いて述べる．

- 組の要素へのアクセスを演算子．（ドット）で表す．
- *PushH(q, val)*は，組*(q, val)*を *H* の末尾に追加する．
- *PopH()*は，*H* の末尾から要素を一つ取り出す．
- *PushD(q, q', α, γ)*は，組*(q, q', α, γ)*を *Δ* に追加する．
- *TailH()*は，*H* の末尾の要素を返す．
- *GetQV(v)*は，変数 *v* を問う状態 $q \in Q$ を返す．
- *GetEF(f)*は，関数 *f* を評価する状態 $q \in Q$ を返す．
- *GetEE(e)*は，式 *e* を評価する状態 $q \in Q$ を返す．
- *RFindH(q)*は，状態 $q \in Q$ と *H* の *i* 番目の要素 *h=(q', val)*において，*q=q'*かつ最大の *i* となる *h* による *h.val* を返す．
- *Replace(e, x, val)*は，式 *e*，$x \in V(e)$，$x \in F(e)$において，*e* の *x* に該当する部分を *val* で置き換える．
- リスト *L* の *i* 番目の要素へのアクセスを L_i と表す．
- *Next(q)*は状態 *q* において *ope=NEXT* であるときの *γ* の定義に用いられる．
- *Prev(q)*は状態 *q* において *ope=PREV* であるときの *γ* の定義に用いられる．

3.3.2 変数の値を問う状態

変数 *v* と状態 *q* において，下記のように定義する．

- $\sigma(q) \in \Sigma$は，*v* の値を問う
- *GenQV(v)*は，*Q* に *q* を追加し，*Σ*に *σ* を追加し，*q* を返す
- *Next(q)*は，*PushH(q, val)*を実行する
- *Prev(q)*は，*PopH()*を実行する

3.3.3 実行対象を問う状態

集合 *S* および状態 *q* において，下記のように定義する．

- $\sigma(q) \in \Sigma$は，*S* の要素を選択肢として次に実行される対象を問う
- *GenQX(S)*は，*Q* に *q* を追加し，*Σ*に *σ* を追加し，*q* を返す
- *Next(q)*は，*PushH(q, val)*を実行する
- *Prev(q)*は，*PopH()*を実行する

3.3.4 式を評価する状態

式 *e* および状態 *q* において，下記のように定義する．

- $\sigma(q) \in \Sigma$は，動作無しである
- *GenEE(e)*は，*Q* に *q* を追加し，*Σ*に *σ* を追加し，*q* を返す
- *Next(q)*は，図 3 に示される．*e* の変数および関数を具体値に置き換えて評価する．*q* は 2 回以上遷移する可能性があるため，1 行目にて式を複製している．

```
1: e' = e
2: for v in V(e') do
3:     Replace(e', v, RFindH(GetQV(v)))
4: end for
5: for f in F(e') do
6:     Replace(e', f, RFindH(GetEF(f)))
7: end for
8: PushH(q, Eval(e'))
```

図 3　式 e を評価する状態のための Next(q)

```
1: f' = f
2: for a in A(f') do
3:     Replace(f', a, RFindH(GetEE(a)))
4: end for
5: PushH(q, Eval(f'))
```

図 4　関数 f を評価する状態のための Next(q)

- *Prev(q)*は，*PopH()*を実行する

3.3.5 関数を評価する状態

関数 *f* および状態 *q* において，下記のように定義する．

- $\sigma(q) \in \Sigma$は，動作無しである
- *GenEF(f)*は，*Q* に *q* を追加し，*Σ*に *σ* を追加し，*q* を返す
- *Next(q)*は，図 4 に示される．*f* の引数を具体値に置き換えて評価する．*q* は 2 回以上遷移する可能性があるため，1 行目にて関数を複製している．
- *Prev(q)*は，*PopH()*を実行する

3.3.6 状態の生成と遷移の設定

図 5 に示す関数 *Gen* は，問いと評価の状態をすべて生成し（1 行目から 7 行目），その後でそれらの状態間の遷移を設定する（9 行目以降）．6 行目で式を評価する状態を作成し，7 行目で関数を評価する状態を作成している．この順序は，関数の引数は式であり，関数を評価するにあたり引数を評価する状態を参照するためである．9 行目から 17 行目において，関数定義の引数を問う状態間の遷移を第一引数から順に設定している．18 行目から 19 行目では，最後の引数を問う状態と実行式を問う状態との遷移を設定している．21 行目では実行式を問う状態と式を問う状態との遷移を設定し，23 行目から 24 行目では，式を問う状態と呼出元を問う状態との遷移を設定し，26 行目から 27 行目では，式を問う状態と実行式を問う状態との遷移を設定している．

式は関数を含むことがあり，関数の引数は式であることがある．このためこれらを問う状態や評価する状態の設定を再帰関数により実現する．図 6 に示す関数 *ExprTrans* は，式を構成する要素を問う状態の遷移をそれら要素の評価順に従って設定する（1 行目から 10 行目）．その後，最後の要素を問う状態と式を評価する状態の遷移を設定する（11 行目から 13 行目）．

```
1: qfd = GenQX(FD)
2: qx = GenQX(E)
3: qcr = GenQX(CR)
4: QAD = {q | fd ∈ FD, AD(fd) のすべての要素 a に対して，q = GenQV(a)}
5: QV = {q | e ∈ E において，e に含まれるすべての変数 v に対して，q = GenQV(v)}
6: EE = {q | e ∈ E において，q = GenEE(e)}
7: EF = {q | e ∈ E, f ∈ F(e) において，q = GenEF(f)}
8: ini = qfd とする
9: q = qfd
10: for fd in FD do
11:     for i to | AD | do
12:         q′ = GetQV(AD_i)
13:         PushD(q, q′, ope = NEXT, Next(q))
14:         PushD(q′, q, ope = PREV ∧ q = TailH().q, Prev(q′))
15:         q = q′
16:     end for
17: end for
18: PushD(q, qx, ope = NEXT, Next(q))
19: PushD(qx, q, ope = PREV ∧ q = TailH().q, Prev(qx))
20: for e in E do
21:     q = ExprTrans(qx, e, qfd, qcr)
22:     if e ∈ RT then
23:         PushD(q, qcr, ope = NEXT, Next(q))
24:         PushD(qcr, q, ope = PREV ∧ q = TailH().q, Prev(qcr))
25:     else
26:         PushD(q, qx, ope = NEXT, Next(q))
27:         PushD(qx, q, ope = PREV ∧ q = TailH().q, Prev(qx))
28:     end if
29: end for
```

図 5　Gen()

図 7 に示す関数 *FuncTrans* は，まず，関数の引数を問う状態と評価する状態の遷移を第一引数から順に設定する（1 行目から 3 行目）．その後，関数定義に含まれる関数の場合，最後の引数の状態と呼出先を問う状態との遷移を設定した後で，呼出先を問う状態と関数を評価する状態との遷移を設定する（5 行目から 9 行目）．そうでない場合（例えば，ライブラリ関数）は，最後の引数の状態と関数を評価する状態との遷移を設定する（11 行目から 13 行目）．

3.4 関数 *Eval(x)*

関数 *Eval(x)*は，式 *x* または関数 *x* の評価結果を返し，図 3 の 8 行目および図 4 の 5 行目にて用いられている．引数 *x* は学習者の応答に依存し，様々な値となることが多いため，演習実施前に機能への設定値として用意しておくことが難しい．そこで，出題対象のプログラムを GDB のシングルステップ実行モードで実行中に，*x* を引数として print 命令を実行することで，print 命令の出力として *x* の出力を得る．

```
1: for i = 1 to | EO(e) | do
2:     if EO(e)_i ∈ V(e) then
3:         q′ = GetQV(EO(e)_i)
4:         PushD(q, q′, ope = NEXT, Next(q))
5:         PushD(q′, q, ope = PREV ∧ q = TailH().q, Prev(q′))
6:         q = q′
7:     else
8:         q = FuncTrans(q, EO(e)_i, qfd, qcr)
9:     end if
10: end for
11: q′ = GetEE(e)
12: PushD(q, q′, ope = NEXT, Next(q))
13: PushD(q′, q, ope = PREV ∧ q = TailH().q, Prev(q′))
14: return q′
```

図 6　ExprTrans(q, e, qfd, qcr)

```
1: for i = 1 to | A(f) | do
2:     q = ExprTrans(q, A(f)_i)
3: end for
4: if uf ∈ UF then
5:     PushD(q, qfd, ope = NEXT, Next(q))
6:     PushD(qfd, q, ope = PREV ∧ q = TailH().q, Prev(qfd))
7:     q′ = GetEF(uf)
8:     PushD(qcr, q′, ope = NEXT, Next(qcr))
9:     PushD(q′, qcr, ope = PREV ∧ qcr = TailH().q, Prev(q′))
10: else
11:     q′ = GetEF(uf)
12:     PushD(q, q′, ope = NEXT, Next(q))
13:     PushD(q′, q, ope = PREV ∧ q = TailH().q, Prev(q′))
14: end if
15: return q′
```

図 7　FuncTrans(q, f, qfd, qcr)

4. おわりに

本稿ではプログラムトレース学習支援システムにおける問い機能を自動化する手法を述べた．この手法は，問いの動作を拡張有限状態で定義し，その状態機械をプログラム解析ツールで得られる情報およびデバッガを用いて実現するものである．

今後の課題として，提案手法の適用範囲を探りたい．例えば，実際の授業で用いているプログラムのうちどの程度のものに提案手法を適用できるかを，提案法を実装したシステムにて評価する実験を行う．

謝辞

本研究の一部は JSPS 科研費 17K01122 および 18K11586 の助成による．

参考文献

[1] 東本崇仁, 赤倉貴子, “提案するプログラムトレース課題のための学習支援システムの開発とその実践,” 電子情報通信学会論文誌 D, Vol. J101-D, No. 6, pp.810-819 (2018).