プログラミング学習における部分実行を用いた理解支援方法の提案
―繰り返し文を対象として―

2017SE092 土岐英暢
指導教員：吉田敦

1　はじめに

プログラミング学習者にとってプログラミング技術を向上させるための一つの有効な手段がコードリーディングである．複合的な制御文，複数の変数を含むソースコードが対象の場合，慣れていない学習者は，実行状態とその変化を把握することが難しい．そこで，理解の労力を減らすために，プログラムを簡略化する手法を提案し，把握すべき情報を減らす．簡略化には部分実行の技術を活用し，ソースコードの展開，変数の値の反映，文の削除の 3 つを繰り返し適用し，段階的に学習者に提示する．

2　関連研究

部分評価を使った理解支援として PHP を対象としたものがあるが [1]，段階的に提示する仕組みがない．動的解析より変数の値を示す理解支援 [2][3] もあるが，理解しやすくソースコードを変換する支援はない．

3　理解支援方法の提案

3.1　理解支援方法の概要

本研究では，学習者が自身で各処理での実行状態の変化を追っていきながら，全体の処理の流れを把握する過程を重視し，段階的な簡略化の方法を採用する．理解支援の具体的な方法は次の通りである．まず，前提として，対象となるソースコードは，コードリーディングを行う際に学習者が躓きやすい複数の制御構造が入れ子になっており，かつ，複数の変数を含むものとする．ソースコードの変換として，部分実行における，繰返しの展開，変数参照の値への置き換え，実行されない文の削除を順次適用する (図 1)．さらに，元のソースコードとの関係を把握しやすくするために，段階的に変換結果を提示できるよう，各変換ごとのソースコード提示，適用対象や回数の指定を可能にする．これにより，学習者がプログラムを読み解くときに把握すべき実行状態を減らすことができる．

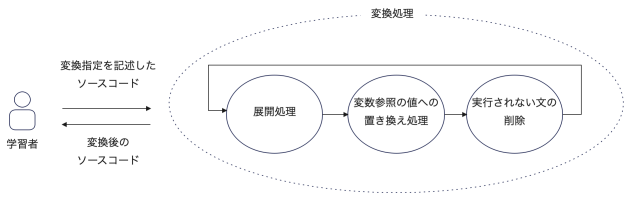

図 1　変換処理の流れ

3.1.1　繰り返しの展開

繰り返しの展開では，対象の繰り返し文の内部処理を制御文の前または後に 1 つ配置する．前の配置は繰り返しの 1 回目を，後の配置は最後の繰り返しを外に出すことを意味する．そのとき，元の繰り返し文も展開後の整合性のために変換を施す．変換の例を図 2 を示す．

この処理は学習者が指定した回数分適用する．本研究では基本的な for 文の形を対象とし，(1) 初期化式は変数への定数の代入式のみ，(2) 条件式は変数と定数との比較式のみ，(3) 変化式ではインクリメント，デクリメントのみ使用とする．なお，展開の方法は for 文以外の制御文にも適用可能である．

変換前

```
1  for (i = 0; i < N; i++) {
2    printf("%d\n", i);
3  }
```

変換後

```
1  i = 0;
2  printf("%d\n", i);
3  for (i = 1; i < N; i++) {
4    printf("%d\n", i);
5  }
```

図 2　展開処理の適用例

3.1.2　変数参照の値への置き換え処理

展開されたソースコードにおいて，定数値が代入された変数について，その参照を値に置換える．ただし，すべての変数参照が置換えられるわけではなく，次のように行う．

- 代入から制御の流れを追い，変数参照をその定数値に置き換える
- 計算可能な式は評価して記録するとともに，新たな変数への代入がある場合は，以降の置換えでその結果も反映させる．
- if 文の分岐では，各分岐先ごとに反映させる．
- if 文の分岐の合流する時点で反映は止める．
- 繰返し文の分岐に遭遇したら反映は止める．

なお，反映を止める理由は，それ以降は，変数参照が複数の値を取り得るからである．変換の適用例を図 3 を示す．

変換前

```
1  x = 1;
2  y = x + 2;
```

変換後

```
1  x = 1;
2  y = 1 + 2;
```

図 3　置き換え処理の適用例

3.1.3 実行されない文の削除

計算可能だった式に基づき，不要な文を削除する．例えば，if 文の条件式が真なら then 節のみを，偽ならば else 節のみを残し，if 文自体は削除する．なお，代入文は，変数の値の反映を止めたあとに変数が参照されている可能性があるので，そのまま残す．変換の適用例を図 4 を示す．

変換前

```
1  x = 1;
2  if (x > 0)
3    printf("plus\n");
4  else
5    printf("minus\n");
```

変換後

```
1  x = 1;
2
3    printf("plus\n");
4
5
```

図 4 置き換え処理の適用例

3.2 学習者への段階的提示

学習者にとっては，理解の進行に合わせて，徐々にソースコードが変換されていくことが望ましい．そこで，段階的に提示を行う．このとき，理解しようと着目している部分は変換する必要があるが，それ以外の部分が変換されると，むしろ理解を阻害することになる．よって，適用対象や適用回数の指定も必要である．適用対象と回数の指定方法として，コメントはソースコードの実行に影響を及ばず，学習者が直感的に指定をできるため，コメントを利用する．対象とする for 文や代入文に続く形で指定コメントを記述する．

4 評価

実装を行い，提案方法が実現可能であることを確認した．実装にあたっては，TEBA[4] を利用し，展開はパターン変換で実現した．また，値の置換えと文の削除は TEBA が提供する部分評価ツールのプロトタイプを利用した．

指定コメントが記述されたソースコード例である Lisitng1 を入力とした際の出力結果を Listing2 に示す．変数の値が把握でき，制御文の入れ子状態が緩和されていることから，理解支援に役立つことが分かる．

5 考察

提案方法では，ソースコードに段階的な適用を施し，出力することは可能であるが，それぞれの適用前後での変化を把握するための支援が存在していない．変換の前後で具体的にどこがどう変わったかが理解できる仕組みが必要である．部分評価を適用する際，計算可能なものをすべて置き換えることは必ずしも望ましくない．計算可能な式をすべて置き換えると，元のソースコードから乖離し，本来理解したいものが理解できなくなる．よって，変換対象を細かく指定できる仕組みも必要である．

本研究では，ソースコード中のコメントについては，配慮をしておらず，変換対象内に存在する指定コメント以外のコメントが消える．また，変換前に存在していた空白なども同様に反映されない．コメントや空白はプログラムを理解する上で重要な要素であり，変換後にも残ることが望ましい．TEBA では，コメントや空白を含め，すべて抽象構文木に含まれるので，実現は可能である．一方で，要素の削除を行う際には関連するコメントが削除されている方が理解しやすい場合もある．これらの制御は今後の課題である．

Listing 1 適用前のソースコード例

```
m = 10;
for(j = 0; j < m; j++)    //@2
{
        num1 = num * j;
}
```

Listing 2 適用後のソースコード例

```
m = 10;
j = 0;
{
        num1 = num * (0);
}
j = 1;
{
        num1 = num * (1);
}
for(j = 2; j < m; j++)
{
        num1 = num * j;
}
```

6 おわりに

本論文では繰り返し文で構成されるソースコード片を対象とし，部分実行を段階的に適用させる支援方法を提案した．追従性を踏まえながら，学習者自身が解釈していく過程を残したソースコードを提示することを目的としている．提示方法については改善していく必要がある．

参考文献

[1] 吉田敦, 蜂巣吉成, 阿草清滋:『ウェブアプリケーション開発のための PHP オンライン部分評価器の試作』, ソフトウェア工学の基礎 XXIII – 日本ソフトウェア科学会, 近代科学社, FOSE 2016, pp.217-222, Dec. 2016.

[2] 長谷川洸也，川地周作：『命令型プログラミングにおける動作理解支援に関する研究』, 南山大学理工学部 2014 年度卒業論文．(2015)

[3] 河本健吾，山田恭裕：『初心者向けのポインタを用いたプログラムの動作理解支援方法の提案』，南山大学理工学部 2015 年度卒業論文．(2016)

[4] TEBA, http://tebasaki.jp/src.