繰り返しを伴うバッファ操作処理における ポインタ変数の変化の理解支援

2020SE053 酒匂駿　2020SE091 高畑竜馬
指導教員：吉田敦

1　はじめに

プログラミング学習の効果的な手法として既存のソースコードを実行経路を辿るように読み，処理を理解するコードリーディングがある．C 言語の実用的なソースコードではポインタが使われるが，学習者にとっては理解が難しい．ポインタを理解するときは，ポインタがどの領域を指し，そこにはどのような値があるのかを頭の中で思い浮べる．さらに繰り返しが伴うとポインタは継続的に変化するので，頭の中でポインタの状態を保持しつつ，ポインタの値や指す値を更新しながらコードを読み進めていくが，容易ではない．この繰り返しの中でポインタがよく使われる典型がバッファ操作処理である．これを容易にするには，ポインタの変化について，記憶すべき情報の量を削減する必要があり，削減には，ポインタの状態をいつでも確認できるように可視化する必要がある．これにより，理解が容易になりコードリーディングでの学習に貢献できる．

本研究では繰り返し処理におけるポインタ変数の変化を理解支援のために，ポインタ変数の値を部分評価により計算し，ソースコードに対するポインタの値の埋め込みおよび間接参照の値の置き換えの 2 種類の独立する表現から構成される可視化手法を提案する．適用対象は典型例であるバッファ操作に限定する．ただし，ポインタが変化する操作であれば，他のプログラムでも適用可能である．

ポインタの値を埋め込む方法を「抽象展開」，ポインタ変数に初期値を与えて間接参照先の値に置き換える方法を「具象展開」と呼んで区別する．ポインタの値と指す値を部分評価によって計算し，ソースコードに表示を埋め込むことで，1 つの実行例に依らず，実行時のポインタの変化を把握しやすくなる．部分評価とはプログラム最適化手法の 1 つで，事前に静的な情報で計算できる部分を計算する方法である．しかし，繰り返し部分では繰り返し 1 回ごとのポインタの値や指す値を提示できないので展開が必要である．繰り返しの展開とは 1 回の繰り返しで評価される式や文を繰り返し文の前に配置することを基本操作とし，それを繰り返し適用することである．繰り返し回数がわからない場合には，繰り返しの前後数回分のみを展開して表示する．なお，初期値を与えてポインタを理解したいときは，繰り返しが完全に展開される方が理解性を高めることから，初期値は繰り返しの回数に関わるすべての変数に対して与えるものとする．

この提案では，次の 2 つを明確にすることが課題となる．

(1) アドレス値を直接用いずにポインタの値の変化を表現する方法

(2) 繰り返し数や脱出経路が定まらないときの繰り返し内および繰り返し後のポインタ変数の計算方法

(1) の課題は，ソースコードの静的解析では，メモリの割り当て方法は求まらないので，アドレスの値を決められないことにある．(2) の課題は，繰り返し内と後のポインタの状態を把握したいときに問題となる．繰り返し内と繰り返し後のポインタの値は，繰り返しの回数やどの時点で繰り返しを脱出したかによって変わる．静的解析では繰り返し数や繰り返しの脱出時点が決まらないので，ポインタの値を計算できない．

課題 (1) は，ポインタの値を基点からの相対位置で表現することで解決する．基点からの相対位置とは，ポインタ変数のうち，宣言または計算できない値の代入時のポインタの値を基点として，そこからのポインタの変化量を指す．ポインタの変化量とは，ポインタ演算における加減算で求まる値であり，バイト単位のアドレスではない．相対位置を示すことで，繰り返しごとのポインタの値の変化を読み取ることができる．以下では，この基点からの相対位置で表現するポインタの値のことを「ポインタ相対値」と呼ぶ．

課題 (2) は，脱出経路を限定し，繰り返し数をパラメタとする式でポインタを表現することで解決する．繰り返しは条件式の判定で終了するものとし，最初の数回の処理，任意の繰り返し，繰り返し終了の処理の 3 つの部分に展開する．最初と最後の展開でポインタの値を表現し，繰り返し回数は決定できないことから任意の回数を表す特殊な変数で表現する．これにより繰り返し終了後のポインタの値もその特殊な変数を含んだ値で計算する．

2　関連研究

土岐 [4] は，繰り返し文を，展開，変数参照の値の置き換え，実行されない文の削除を順次適用してソースコードの変換を行っているが，ポインタには対応していない．

小山ら [2] は，変数の値の変化を可視化する支援方法を提案している．ソースコード全体を解析し，繰り返し文があった場合はブロック単位で分けて，その中の変数のトレースを表で示す．しかし，表は値の変化のみが表示されているので，元ソースコードの実行文との照らし合わせが必要になる．また，繰り返し回数が多い場合は表示量が多くなり元ソースコードと照らし合わせづらい．

加藤ら [1] は，関数の引数にポインタを用いたプログラムの可視化ツールの提案をしている．プログラムを逐次実行し，変数の情報を取得して各地点の図を提示する．図では変数の値，ポインタの指し先や指す値が表示されるので，ポインタの状態を直感的に理解できるが，ポインタが頻繁

に変化する場合，繰り返しごとに図と元ソースコードの照らし合わせるので，手間がかかる．

以上のように，処理が簡単な繰り返し文の部分実行や，その中の変数の処理，ポインタの可視化は提案されているが，ポインタを含む繰り返し文の対応，照らし合わせる手間も含めた理解支援が不十分である．本研究の手法では，ポインタを含む繰り返し文に対応しており，得られる結果はソースコードに埋め込む形で出力する．これにより，ソースコードのみでの理解が可能になり，入力ソースコードと照らし合わせを行う際にも手間を削減できる．

3　ポインタ変数の変化の理解支援の提案

本研究では，バッファ操作処理でのポインタ変数の変化を把握するためのプログラム理解支援手法として，部分評価を用いてポインタ相対値の埋め込みおよび間接参照の値の置き換えの 2 種類の独立する表現から構成される可視化手法を提案する．バッファ操作処理を前提とするので，ポインタはバッファ上の要素を指すものとし，バッファ内で指し先が変化する．

提案手法の全体像を図 1 に示す．学習者は，ソースコード内の理解したい繰り返し文に印を付け，ソースコードを支援ツールに入力する．支援ツールは印が付いた繰り返し文に対する 2 つの展開表現を生成し，出力する．以下，ポインタ相対値を埋め込む表現を抽象展開表現，ツールを通して与えた初期値に基づいて間接参照を置き換える表現を具象展開表現と呼ぶ．2 つの展開表現と入力ソースコードは，その対応関係を取りやすくするために色付けを行う．対象となるのは，繰り返しの条件式や for 文の変化式などである．

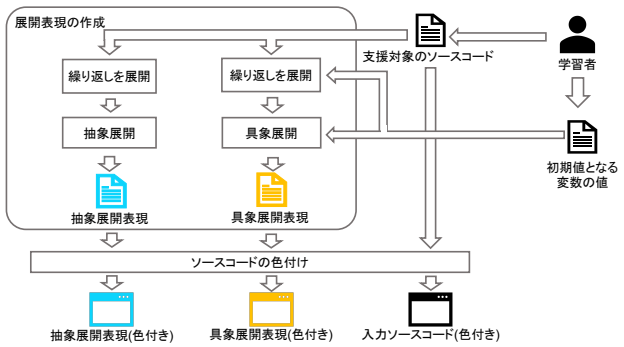

図 1　提案手法の全体像

3.1　前提条件

本研究では，バッファ操作処理の例として C 言語標準ライブラリの string を対象とし，具体的なソースコードとして NetBSD[6] のライブラリを採用する．NetBSD は移植性が高く，アーキテクチャ非依存な書き方だからである．繰り返し文の外で宣言されたポインタ変数や仮引数を含めるために支援の単位は関数ブロックとし，その中でも以下の前提条件に当てはまる関数を本研究の対象とする．

- 1 重，2 重の繰り返しを含む
- 繰り返しにポインタ変数を含む
- if 文の分岐双方に繰り返しを含まない
- 配列，構造体を含まない
- 再帰による繰り返しを含まない
- 前処理指令を含まない

if 文の分岐双方に繰り返しを含む場合，条件分岐の各経路で異なる計算結果になり，それを統合できないので対象としない．配列，構造体については本研究の理解支援の対象ではないので議論の簡素化のために対象外とする．再帰を用いた場合は繰り返しの展開が困難であり，前処理指令については，それを含んだ CFG の構成が困難なことから対象としない．

また，前提として学習者はポインタの概念は理解しているが，バッファ操作処理を読み慣れていない人を想定する．

3.2　指し先未定のポインタ相対値の計算と方法

ポインタ変数にツールを通して初期値が与えられておらず，値が分からない場合のポインタ相対値の計算は，実行経路ごとに部分評価を行い，その過程で次の操作を行う．

- ポインタ相対値は宣言または計算不可能な代入時に基点の 0 とする
- ポインタ変数のインクリメント・デクリメントを評価し，値を更新する
- ソースコード上のポインタ変数の出現の直後にポインタ相対値の表記を挿入する

計算不可能な代入とは右辺の式の詳細や必要な変数の値が揃っていないものを指す．ポインタ相対値の表示例を図 2 に示す．ポインタ相対値は，ポインタ変数の出現を p，その出現のポインタ相対値を N としたとき，$p(+N)$ となる．なお，この表記は C 言語の構文に当てはまらないので，通常のポインタ演算と誤読することはない．また，インクリメント・デクリメントの場合には加減算前，加減算後の 2 つのポインタ相対値の表示を行う．これによりポインタ相対値が加減算する前なのか後なのか誤解を生むのを防ぐ．

3.3　ポインタの指す値が求まる場合の提示方法

学習者がツールを通して変数の初期値を入力し，ポインタの指す値が求まる場合の提示方法は次の通りである．

- 学習者がツールを通して変数の初期値を入力
- 入力された値を代入する式文を追加
- 計算可能な間接参照をその値に置換

間接参照の置換表示例を図 3 に示す．ポインタの指す値を示す方法では，指定された変数に学習者が値を入力することで変数の出現や間接参照を部分評価で求まった値に置き換える．部分評価を用いるので指定された変数すべてに値を入力する必要がなく，理解に必要な変数のみに絞って置き換えることができる．

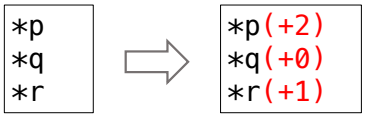

図 2　ポインタ相対値の表示例

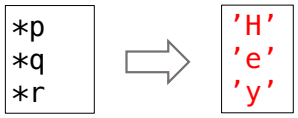

図 3　間接参照の置換表示例

3.4　繰り返し文の展開

繰り返しの展開内容は抽象展開表現と具象展開表現で異なる．抽象展開表現では，繰り返し回数が決まらないので，最初の数回分の処理の展開，その後に任意の回数の繰り返し，最後の処理の展開の 3 つで表現する．具象展開表現では，完全に展開される方が理解性を高めることから，初期値は繰り返しの回数に関わるすべての変数に入力し，繰り返しの条件式が偽になるまで展開する．展開表現の例として，ソースコード 1 の抽象展開表現をソースコード 2 に，具象展開表現をソースコード 3 に示す．

ソースコード 1　strcpy.c[3]

```
1  char *strcpy(char *s1, const char *s2){
2    char *tmp = s1;
3    while (*s1++ = *s2++)
4        ;
5    return tmp;
6  }
```

ソースコード 2　抽象展開表現の例

```
1  char *strcpy(char *s1, const char *s2){
2    char *tmp(+0) = s1(+0);
3    if (*s1(+0)++(+1) = *s2(+0)++(+1)) {
4        ;
5    }
6    if (*s1(+1)++(+2) = *s2(+1)++(+2)) {
7        ;
8    }
9    if (*s1(+2)++(+3) = *s2(+2)++(+3)) {
10       ;
11   }
12   int nα; /*nα は任意の繰り返し回数*/
13   LOOP (int α = 3; α < nα; α++){
14   if (*s1(+α)++(+α+1) = *s2(+α)++(+α+1)) {
15       ;
16   }
17   }
18   if (*s1(+nα)++(+nα+1) = *s2(+nα)++(+
         nα+1)) {
19       ;
20   }
21   if (*s1(+nα+1)++(+nα+2) = *s2(+nα+1)++(+
         nα+2)) {
22       ;
23   }
24   return tmp(+0);
25 }
```

ソースコード 3　具象展開表現の例

```
1  char *strcpy(char *s1, const char *s2){
2    char *tmp = s1;
3      s2 = "hey";
4      if (*s1++ = 'h') {
5          ;
6      }
7      if (*s1++ = 'e') {
8          ;
9      }
10     if (*s1++ = 'y') {
11         ;
12     }
13     if (*s1++ = '\0') {
14         ;
15     }
16   return tmp;
17 }
```

3.4.1　抽象展開表現

抽象展開表現では繰り返し文の展開とポインタ相対値の表示を行う．抽象展開表現の表示例をソースコード 2 に示す．繰り返し文の展開では，繰り返し文の任意の繰り返し回数を n としたとき，n 回までの繰り返し文の展開を行う．ただし，これでは n の回数によっては展開量が多くなる．そこで 1，2，3，n 回目の処理と繰り返し終了の処理を if 文の形で展開し，残りの 4〜$n-1$ 回目までの処理を LOOP としてまとめて 1 つの処理で表現する．LOOP は本研究の独自で設定した表示であり，残りの 4〜$n-1$ 回目までの処理を for 文と同じ形式でまとめている．繰り返し終了の処理とは繰り返し文で条件式が偽になる処理で，条件式の評価のために展開するが，中の文は実行されない．LOOP の繰り返し変数としてソースコードで使用されない α を用い，n は繰り返し変数に合わせて n_α を用いる．

3.4.2　具象展開表現

具象展開表現では学習者が特定の変数に値を入力すると，繰り返し文の展開とポインタの指す値が表示される．具象展開表現の表示例をソースコード 3 に示す．入力できる変数は関数の仮引数と関数内で宣言されている変数である．変数の値を入力する際に，繰り返し回数を決める変数として条件式の評価に必要な変数は具象展開をするのに必須とする．変数に値を入れると，その変数の代入式が支援対象の繰り返し文の直前に追記され，入力値を元に繰り返しを展開する．

また，入力された変数の値によって if 文の評価が可能な場合は行う．評価後，実行される文を残し，実行されない文はそのことを可視化するために薄い色で表示する．条件式の評価をすることで，実行される文を明確化し，学習者が読むコード量を減らす．

3.5　実現

提案手法が実現可能であることを示すために，本研究ではソースコードから抽象展開表現と具象展開表現の 2 つを生成するツールを実装した．ツールの実現にあたり，構文解析やプログラムの書き換えには TEBA[5] のツールを用

いる．また，部分評価については，TEBA の部分実行器を拡張して実現した．

4　検証・評価

検証では NetBSD[6] のライブラリの中の string ディレクトリにある文字列操作関数を対象とする．string ディレクトリには 47 の関数があるが，第 3 章で挙げた前提条件を満すものは 47 の関数にうち 21 であった．これらを本研究の検証の対象とする．検証結果は表 1 に示す．適用可能な関数の割合は小数点第四位以降は切り捨てしている．対象関数 21 個中，抽象展開表現は約 66%，具象展開表現は約 57% が適用可能だった．

表 1　適用可能な関数の割合

展開表現	適用可能な関数	適用可能な関数の割合
抽象展開表現	14	0.666
具象展開表現	12	0.571

理解支援に対する評価として，理解に必要な作業量と記憶量が，提案手法によりどの程度削減されるか，定式化して求めた．作業量は頭の中で計算する作業の量と定義し，記憶量は変数の値やその変化を頭の中で保持する量と定義した．作業量と記憶量についてそれぞれ差を求めたところ，作業量は頭の中で計算する演算子の数に基づいて入力ソースコードとの対応関係を取る作業の負荷に依存して変わることを，記憶量は展開表現によって削減できることを確認できた．

5　考察

ポインタ相対値の基点について考察を行っていく．本研究では C 言語標準ライブラリを対象としたが，以下の点でポインタ相対値の基点が誤解を生む可能性がある．

(1) 別のバッファを指しているときに，それを区別する情報を示していない

(2) 計算できない値の代入時のポインタの値が基点となるので，別のポインタ変数の値を引き継げない

(1) では，ソースコード 4 のポインタ変数 s と t のように複数のポインタ変数が別々のバッファを指していても本研究の支援方法では，それが別のバッファを指しているか分からないので，同一のバッファを指していると勘違いする可能性がある．これを解決するには，ポインタ相対値の基点として代入されたアドレスを用い，異なる基点の値を明確化する必要がある．例えば，図 4 ような表示にすることで，代入されたポインタ変数のバッファを基点としていることが明確化されて，そこからの相対位置を示せる．

ソースコード 4　ポインタ相対値の基点が誤解を生む例

```
1 char sample(char *s1, char *s2) {
2   while(*s1++ != '\0') ;
3   char *s = s1;
4   char *t = s2;
5   return *s;
6 }
```

```
char *s(+s1 + 0) = s1;
char *t(+s2 + 0) = s2;
```

図 4　代入されたアドレスを基点とした例

(2) では，ソースコード 4 のポインタ変数 s のようにポインタ変数に代入した際に，直前に計算したポインタ変数の相対位置が本研究の支援方法では扱えず，すべて 0 からの表示になる．問題の本質としてはバッファの位置を考慮する必要がある点で (1) と同じであり，(1) が解決されれば代入した変数のバッファからの相対位置であることが示され (2) も解決される．

6　おわりに

本研究では，バッファ操作処理でのポインタ変数の変化の理解をするために，ポインタ相対値の埋め込みおよび間接参照の値の置き換えの 2 種類の独立する表現から構成される可視化手法を提案したことで課題を解決した．今後の課題として，ツールの拡張，ポインタ相対値の基点の改善が挙げられる．

参考文献

[1] 加藤ちひろ，松尾翔馬，森本朱音：“関数の引数にポインタを用いたプログラムの可視化による動作理解支援ツールの提案”，南山大学理工学部 2018 年度卒業論文 (2019).

[2] 小山秀明，山田俊行：“変数の値の変化の可視化によるプログラム理解支援”，情報処理学会論文誌，Vol.10 No.4，pp.1–11 (2017).

[3] 柴田望洋：“新・明解 C 言語 入門編”，p.218，SB クリエイティブ株式会社 (2014).

[4] 土岐英暢：“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 –”，南山大学理工学部 2022 年度卒業論文 (2023).

[5] 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).

[6] GitHub：「NetBSD」　　，<https://github.com/NetBSD/src>，(参照 2024 年 1 月 19 日).