表現の違いを考慮したマクロ逆置換方法に関する研究

2006MI127 織田 智仁　　2007MI219 曽我 展世

指導教員　沢田 篤史　　蜂巣 吉成

1　はじめに

ソースコード内に含まれる重複したコード片は，ソースコードの可読性や保守性を低下させる一因となる．重複したコード片を解消し，可読性や保守性を高める手段の一つとして，リファクタリングが挙げられる [3]．C 言語で記述されたソースコードでは，重複箇所をマクロや関数でまとめることで重複したコード片を解消できる．

重複箇所をマクロや関数でまとめる作業は，発見した重複箇所からマクロや関数を定義する作業と，重複箇所をマクロや関数の参照に置き換える作業で構成される．マクロや関数を重複箇所へ置き換える作業は，手作業による単純作業の繰り返しであり，作業を自動化することで重複したコード片の解消作業にかかる時間を短縮できる．また，手作業による単純作業を繰り返す際には，誤り混入の可能性と置き換え対象箇所の見落としが発生する可能性があるが，作業の自動化により回避できる．

重複したコード片のマクロや関数での置き換え作業の際には，マクロ定義や関数と重複したコード片との記述の違いを考慮する必要がある．本研究では，マクロ定義や関数と重複したコード片との記述の違いを「表現の違い」と呼ぶ．表現の違いは重複したコード片をマクロや関数として抽象化する際に，マクロや関数の記述方法に合わせることで生じ，一つのマクロ定義や関数に対する重複したコード片での記述は複数ある．

本研究では，重複したコード片の解消を目的とし，表現の違いにも対応したマクロ逆置換作業の自動化を行なう．マクロ逆置換とは，マクロ定義と一致する箇所を探索し，その箇所をマクロ参照へ置き換えることである．なお，本研究では関数による重複箇所の解消を対象とせず，マクロによる解消作業のみを対象としている．重複したコード片に名前を付け，可読性を高める作業において，単なる文字列の置き換えであるマクロはソースコード内の様々な箇所で利用できる．また，関数による重複したコード片解消の実現は，表現の違いを吸収する技術に加え，引数の型の判定などの対策が必要になる．ゆえに，マクロに対する置換技術は関数による重複したコード片解消の基本技術となる．

本研究ではマクロ定義に含まれる表現の違いについて，逆置換対象とするマクロ定義から表現の違いを考慮した記述を複数生成し，想定される記述ごとにソースコードの書換えを行なうことで対応した．

2　マクロ逆置換について

C 言語でのマクロ定義の構文を以下に示す．

```
#define マクロ名 置換文字列
#define マクロ名（引数情報） 置換文字列
```

マクロを定義すると，そのマクロ定義以降に出現するマクロ参照は，プリプロセッサにより置換文字列へと置き換えられる．マクロ逆置換とは，ソースコード内の置換文字列と一致する箇所をマクロ参照へと置き換える操作であり，マクロ置換を行なう置き換えと逆の置き換えが行なわれる．

引数情報がマクロ定義に記述されている場合，置換文字列内に記述されている仮引数は実引数によって置き換えられる．マクロ逆置換においては，置換文字列内に記述された仮引数と一致する箇所を実引数として，逆置換結果に反映する．

2.1　マクロを利用した重複コードの解消

図 1 に重複したコード片を含んだソースコード例を示す．この例では配列の要素数を求めるコードが下線部 A，B に共通して記述されている．このソースコードの可読性を高めるには，下線部 A，B に配列の大きさを求める典型的記述である “ArraySizeOf” など適切な名前を付けることが必要である．

```
#include <stdio.h>

main(){
  int aryA[50],aryB[100];
  int i, *aryPnt;
  .....
  /* aryA配列内繰り返し */
  for( i=0 ; i < sizeof(aryA) / sizeof(aryA[0]) ; i++ ){ ..... }
  .....                          A
  .....
  /* aryPnt : 配列終端のポインタ */
  aryPnt = aryB + ( sizeof(aryB) / sizeof(aryB[0]) );
  .....                          B
}
```

図 1　重複コードを含んだソースコード

図 1 の下線部 A，B を解消するためのマクロ定義を以下に示す．

```
#define ArraySizeOf(ary) \
    (sizeof(ary) / sizeof(ary[0]))
```

このマクロ定義では，下線部 A，B に共通する配列の要素数を求めるコードを “ArraySizeOf” というマクロとして記述している．**図 1** の下線部 A，B に対して “ArraySizeOf” というマクロを逆置換した結果が**図 2** である．**図 1** の下線部 A，B を，“ArraySizeOf” という同じマクロで記述することにより，重複したコード片が解消されている．また，重複したコード片が元々持っていた「配列の要素数を求める」という役割をマクロ名で表現することにより，可読性も向上している．

マクロ逆置換の操作は，ソースコードに対する操作としては重複したコード片からマクロ名への置き換えであり，字句のパターンの置き換えで簡単に実現することができる．

```
#include <stdio.h>
#define ArraySizeOf(ary) ( sizeof(ary) / sizeof(ary[0]) )

main(){
  int aryA[50],aryB[100];
  int i, *aryPnt;
  .....
  /* aryA配列内繰り返し */
  for( i=0 ; i < ArraySizeOf(aryA) ; i++ ){ ..... }
  .....                          A
  .....
  /* aryPnt : 配列終端のポインタ */
  aryPnt = aryB + ArraySizeOf(aryB);
  .....                          B
}
```

図 2　マクロによる重複コードの除去

2.2　逆置換を行なう際の問題

逆置換を行なう際にはマクロ定義と逆置換対象箇所との記述の違いを考慮する必要がある．

マクロを記述する際には元の記述から単純に置換文字列を作るとは限らず，置換文字列の書き方を保守性や可読性などを考慮し変更することが多い．置換文字列の書き方を変更することにより，置換文字列と逆置換対象箇所との間に表現の違いが生じる．この表現の違いにより，単純な字句変換では逆置換がされない．そこで表現の違いを吸収する逆置換方法を提案する．

3　表現の違いの種類

逆置換対象箇所からマクロ定義を記述する際に発生する表現の違いを，マクロについての記述が含まれる文献とオープンソースの実際の記述を調査し，以下の 4 種類に分類した．対象とするオープンソースには複数のプログラムのソースコードが含まれる gnu-coreutils 8.4[1] を用いた．

1. ソースコードの見栄えを構成する記述の違い
2. 複数文の置換文字列を囲う記述の有無
3. 括弧による式の変化
4. プリプロセッサ演算子による字句の変化

3.1　ソースコードの見栄えを構成する記述の違い

ソースコードの見栄えを構成する記述として，以下のものがある．

- 空白
- 改行
- コメント
- マクロ定義内での改行 “\”

これらはソースコードの見栄えを構成する記述であり，逆置換対象箇所を捜索する際には必要ではない．よって，ソースコードの見栄えを構成する記述に対しては，置換文字列，逆置換対象箇所双方において無視をすることで，表現の違いを吸収できる．

3.2　複数文の置換文字列を囲う記述の有無

複数の文で構成される置換文字列に対しては，それを囲う記述の典型として “do{...}while(0)” がある．これは，複数の文が正しい入れ子関係を維持することを目的としている．他にも，“{...}” で複数の文を囲う方法もある [4]．複数の文で構成される置換文字列は**図 3** に示すように，逆置換対象箇所の記述の前後に字句が付加されるのみであり，該当箇所を無視することで表現の違いを吸収できる．

図 3　“do{...}while(0)” の付加

3.3　括弧による式の変化

引数を持つマクロの場合，置換文字列内の引数の参照を括弧で囲い，実引数に式が記述された際の誤結合を防ぐ．一般にマクロ定義の置換文字列は逆置換対象箇所と比べ括弧が多用される．

括弧による式の変化は**図 4** に示すように複数の候補が存在し，置換文字列に対する操作のみでは表現の違いを吸収できない．

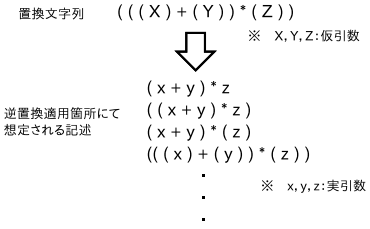

図 4　括弧のによる式の変化

3.4　プリプロセッサ演算子による字句の変化

マクロ定義では以下の 2 種類のプリプロセッサ演算子を用いて，ソースコードの字句に対し特殊な文字列処理を行なえる．これらの演算子については，演算子ごとにマクロ定義を変形することで表現の違いを吸収できる．

- 文字列化 “#”
- 字句の結合 “##”

“#” は引数の前に記述され，実引数を文字列化する演算子である．**図 5** に “#” によるプリントマクロの例を示す．この例では “#” で，printf 文内書式文字列の変化に対応するマクロを定義している．

“#” が含まれていた場合には，“#” のオペランドを文字列として扱うことで表現の違いを吸収できる．また，printf 文等の書式文字列に対する引数として “#” が記述される場合もある．この場合には，“#” のオペランドを文字列として扱う記述に加え，書式文字列内に文字列化したオペランドを展開した記述も考慮する必要がある．

“##” は前後の字句を結合する演算子である．“##” 演算子が含まれていた場合には，前後の字句を結合することで表現の違いを吸収できる．しかし，結合前後の字句は引数として与えられることが多く，逆置換を行なう際には，マクロ参照の実引数を反映させる必要がある．結合前後の字句が双方とも引数の場合には，一つの字句

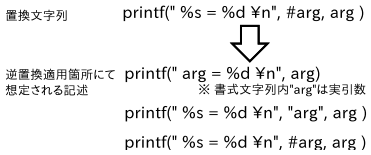

図 5　“#” による表現の違い

を分割して実引数に反映させる必要があるが，字句内の分割箇所が複数想定でき，置換文字列から逆置換結果を一意に特定することはできない．

4　表現の違いを考慮したマクロ逆置換方法

マクロ定義と逆置換対象箇所との表現の違いにより，一つのマクロ定義から記述が異なる複数の逆置換対象箇所を書換える必要がある．この異なる記述に対応するために一つのマクロ定義から想定できるパターンをすべて生成し，各パターンごとに書換えを行う．なお，表現の違いのうち，ソースコードの見栄えを構成する記述については後述する書換えルールの生成で，括弧による式の変化については括弧の正規化操作により対応する．

ソースコードの書換え操作は TEBA[6] を用いた字句の書換え操作として実装した．TEBA を用いることで，逆置換操作を，書換え前の字句の並びと書換え後の字句の並びをマクロ定義から生成する操作で実現できる．また，書換え後の記述に必要となる実引数は，Perl による正規表現を利用し逆置換対象箇所から抽出できる．

4.1　逆置換操作手順

逆置換操作は以下の処理で構成される．

1. 対象のソースコードを構文解析
2. 括弧の正規化
3. マクロ定義から書換えルールの基本形を生成
4. 表現の違いを考慮し，書換えルールの亜種を生成
5. ソースコードの書換え
6. ソースコード再構成

TEBA を利用している箇所は上記手順 1，5，6 である．**図 6** に示す通り，対象となるソースコードと，逆置換をしたいマクロ名を逆置換自動化ツールに入力することで，逆置換操作を行なう．

4.2　括弧の正規化

括弧の正規化とは，ソースコード内の式に対し括弧を付加し，ソースコード内の括弧の記述を統一することである．

TEBA では式の構文解析が行なわれておらず，3.3 節で述べた括弧による表現の違いに対応することができない．そこで，**表 1** に従い，演算子ごとに不足する括弧を仮想的な括弧として付加する．これにより，演算子順位文法に準じた書換えを実現できる．

括弧の正規化により付加された括弧は，属性付き字句系列上では，括弧の属性値を持った空文字[*1]であり，ソースコードの再構成処理の際に破棄する．括弧と同じ種別を持つ空文字は，書換え処理の際には括弧として扱う．

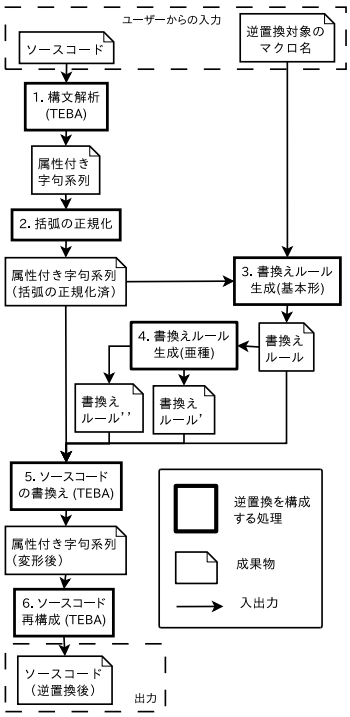

図 6　逆置換操作を構成する処理

表 1　括弧のつけ方（OP はオペランドを表す）

演算子の種類	括弧のつけ方
単項演算子	（演算子（OP）） （（OP）演算子）
二項演算子	（（OP）演算子（OP））
三項演算子	（（OP）?（OP）:（OP））

4.3　書換えルールの生成

マクロ逆置換操作は，逆置換のパターンを字句系列の書換えルールとして記述し，TEBA の字句系列書換え系を用いて実行する．よって，書換えルールの記述は TEBA に従ったものである．書換えルールは，変数等を表す正規表現，書換え前の字句の並び，書換え後の字句の並びから構成されている．マクロ逆置換操作において，書換え前の字句の並びは対象となるマクロの置換文字列から，書換え後の字句の並びはマクロ名と引数情報から生成する．

ソースコードの見栄えを構成する記述は逆置換対象となるマクロ記述を含むソースコード全体に含まれる．逆置換となる書換えを行なう際に，該当する記述を「空白文字[*2]，コメントの 0 回以上の繰り返し」というパターンで表現する．この表現を，置換文字列内の各字句の間

[*1] 種別情報のみを持つ字句

[*2] 属性付き字句系列では改行も空白文字と同じ属性を持つ

に挿入ことで，ソースコードのスタイルを維持したまま，ソースコードの見栄えを構成する記述を無視できる．

逆置換操作後には，逆置換箇所に記述されていたコメントはすべて削除される．しかし，コメントに記述されている情報を失うことで，コードの持つ意味を理解できなくなる恐れがある．よって，逆置換箇所内に含まれていたコメントは一度削除をした後に，本研究室において現在研究されているスタイル維持方法 [5] を用いて逆置換箇所に挿入する．

4.4 表現の違いを考慮した書換えルールの生成

生成された書換えルールを基に，各表現の違いに対応した新たな書換えルールを生成する．逆置換対象箇所で想定されるパターン記述の亜種を複数用意することで，様々な表現の違いに対応する．

4.4.1 複数文の置換文字列を囲う記述

複数文の置換文字列を囲う記述は，該当箇所を除いた書換えルールを追加することで表現の違いを吸収する．すなわち，書換えルールからマクロ定義を記述する際に付加される記述である “`do{...}while(0)`”，“`{...}`” の記述を削除した変形ルールを新たに追加する．

4.4.2 プリプロセッサ演算子による字句の変化

プリプロセッサ演算子が含まれる記述の場合には，各プリプロセッサ演算子に対応する処理を行なうことで表現の違いを吸収する．

置換文字列内に “#” 演算子が含まれていた場合には，“#” 演算子のオペランドの字句を文字列に変形した書換えルールを追加する．また，printf 文等の書式文字列に対する引数として “#” 演算子が記述されている場合には，書式文字列内に文字列化した引数を挿入した変形ルールを新たに追加する．

5 評価と考察

マクロ逆置換自動化ツールの評価には，gnu-coreutils 8.7[1] を用いた．理由としては，Gnu のコーディング規約 [2] にマクロに対する制約がないこと，および，複数の開発者により開発が行なわれているという点から，様々なマクロ定義の記述方法に対して逆置換ツールの評価ができることが挙げられる．

gnu-coreutils 8.7 で対象とした表現の違いが含まれたマクロ定義は 50 個である．これらのマクロ定義のうち，複数文の置換文字列を囲う記述を含むマクロ定義は 10 個，括弧による式の変化を含むマクロ定義は 28 個，プリプロセッサ演算子による字句の変化を含むマクロ定義は 3 個，ソースコードの見栄えを構成する記述の違いはすべてのマクロ定義に含まれていた．

5.1 評価方法

対象となるマクロ定義を gnu-coreutils の中から抽出し，それぞれのマクロ定義に対し独自ツールを利用し，マクロ展開を行なう．また，展開箇所を手作業で書換え，各表現の違いに対し複数の記述を用意することで，各表現の違いに対し想定できる全ての字句のパターンを確認する．

表現の違いを加えた記述に対してマクロ逆置換を行ない，マクロ展開前のソースコードとマクロ逆置換後のソースコードとの差分を確認し，正しく逆置換が行なわれているかを確認する．

5.2 評価結果

上記で挙げた対象となるマクロ定義に対してマクロ逆置換自動化ツールの動作検証を行なった結果，すべてのマクロ定義に対して逆置換を行なうことができた．すべての表現の違いの種類に対し逆置換を行うことができ，表現の違いを考慮したマクロ逆置換ができたと言える．

5.3 考察

プリプロセッサによる処理を行なう表現の違いのうち，“#” 演算子と同程度 “##” 演算子が使われていた．“##” 演算子については，マクロ名とソースコードから実引数の想定ができないとして，実装を行なっていなかったが，マクロを利用する際の一つの利点として “##” 演算子があり，対策を行なう必要がある．

6 おわりに

本研究ではマクロ逆置換自動化ツールを利用して，置換マクロによる重複コード解消作業に含まれる手作業による単純作業の解消を行なった．逆置換作業を自動化する際に問題となる表現の違いについて分類を行なった上で，表現の違いごとにどのような記述が想定できるか考察した．また，マクロ逆置換自動化ツールの中で各表現の違いによって想定される記述を生成し，記述ごとにソースコードの書換え作業を行なうことで対応した．

今後の課題としては，“##” 演算子による文字列結合が記述されていた場合の実引数範囲の特定に対する対策が挙げられる．また，本研究で対象としなかった関数による重複コードの解消についても検討が必要である．

参考文献

[1] Free Software Foundation, Inc., “*Coreutils - GNU core utilities*,” http://www.gnu.org/software/coreutils/, Oct 2010.

[2] Free Software Foundation, Inc., “*GNU Coding Standards*,” http://www.gnu.org/prep/standards/standards.html, Nov 2010.

[3] M. Fowler, *Refactoring: Improving the Design of Existing Code,* Addison-Wesley，May. 1999.

[4] 藤原博文，C プログラミング専門課程，技術評論社，2000.

[5] 本田竜二，石原浩佑，立道昂太，“プログラム書換えツールから独立したスタイル維持方法に関する研究，” 南山大学 数理情報学部 情報通信学科，2010 年度卒業論文要旨集.

[6] 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プログラム書換え支援環境の試作，” ソフトウェアエンジニアリング最前線 *(ソフトウェア・エンジニアリング・シンポジウム 2010 予稿集)*，pp.119-126，Aug. 2010.