表現の違いを考慮したマクロ逆置換方法の提案

曽我展世[†1] 吉田敦[†1] 蜂巣吉成[†1]
沢田篤史[†1] 張漢明[†1] 野呂昌満[†1]

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．本論文では，マクロによる重複したコード片を解消する際の作業の自動化支援を目的とし，表現の違いに対応したマクロ逆置換作業の自動化を行なう．

A Macro Reduction Method Supporting Multiple Program Representations

Hirotoshi Soga,[†1] Atsushi Yoshida,[†1]
Yoshinari Hachisu,[†1] Atsushi Sawada,[†1]
Han-Myung Chang[†1] and Masami Noro[†1]

Duplicated code fragments in a source code cause to decrease the readability and the maintainability. The purpose of this paper is removal of duplicated code fragments using macro reduction. We propose a method for macro reduction supporting multiple program representations.

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[1)]．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．

†1 南山大学
Nanzan University

重複箇所をマクロでまとめる作業は，発見した重複箇所からマクロを定義する作業と，重複箇所をマクロの参照に置き換える作業で構成される．重複箇所をマクロの参照へ置き換える作業は，大半は手作業による単純な書換えの繰り返しである．書換え作業を自動化できれば解消作業にかかる時間を短縮できる．また，単純な手作業を繰り返すと，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．このような重複したコード片解消における書換え支援は，C言語のリファクタリング[2)–4)] では行われていない．

重複したコード片をマクロで置き換える際には，マクロ定義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．本論文では，マクロ定義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．表現の違いにはソースコードの見栄えを構成する空白や改行の違いやマクロを定義する際に付加される括弧などが含まれ，これらの表現の違いが存在することで単純な文字列置換ではマクロ逆置換作業を自動化できない．

本論文では，重複したコード片を解消する際の作業の自動化支援を目的とし，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，重複コードの解消には関数に置き換える方法もあるが，本論文では対象としない．関数に置換える場合には本論文の手法に対して，型の制約等の処理の追加・拡張が必要になる．マクロは関数に比べ記述の制約が少なく，マクロによる重複したコード片の解消の方が，重複したコード片の記述により柔軟に対応できる．

本論文では重複したコード片とマクロ定義との間の表現の違いについて分類を行い，逆置換対象のマクロ定義から表現の違いに対応した正規表現のパターンを自動生成することで文字列置換によるマクロ逆置換作業の自動化を行なう．表現の違いの分類は第 3 章，逆置換作業の自動化は第 4 章で述べる．ソースコードの書換えを文字列置換により実現することで，前処理前のソースプログラムに対し直接書換えを行なうことができる．前処理前のソースプログラムを直接書換えることで，マクロ定義内の重複したコード片を別のマクロで除去できるなど，プログラマが手作業で行なう重複したコード片解消作業により近い書換えを逆置換ツールの中で行なう．

2. マクロ逆置換について

2.1 マクロ逆置換とは

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．重複コードをマクロに置き換える作業ではこれとは逆の置換えを行なっており，本論文ではこれを「逆置換」と呼び，逆置換により解消する重複コード片を「逆置換対象箇所」と呼ぶ．

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にして，逆置換結果に反映する．

2.2　マクロ逆置換の例

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マクロに置き換え，“ArraySizeOf” など式の意味を表す名前で表現する．

図 1 の下線部 A，B を解消するためのマクロ定義の例を以下に示す．

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

図 1 の下線部 A，B をマクロ “ArraySizeOf” に逆置換した結果が**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実現できる．

2.3　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．図 1 の下線部 A，B の記述からマクロ “ArraySizeOf” を定義する際には配列名をマクロの引数に変更する他に，マクロ定義全体を囲う括弧を追加する．また，マクロ定義にプリプロセッサ演算子を利用するような例の場合では，逆置換対象箇所を構成する字句が置換文字列に含まれないなど，記述が大きく変わる．

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，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．

```
#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

Fig. 1　The source code that have duplicated code fragments.

```
#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

Fig. 2　The source code that duplicated code fragments have removed.

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[5)–10)] とオープンソースの実際の記述を調査し分類した．分類結果を**表 1** に示す．対象とするオープンソースには複数のツールのソースコードが含まれる gnu-coreutils 8.4[11)] を用いた．なお，重複したコード片の解消に利用される文や式を定義したマクロ定義を調査し，定数値に意味を持たせる目的で利用される定数値を定義したマクロは調査対象外としている．また，表 1 の分類には文献やオープンソースの記述には現れていないが，後述するように，C 言語の文法による表現の違いのうち意味レベルの表現の違いなど，表現の違いに含まれると判断できるものについても含んでいる．

3.1　C 言語の文法による表現の違い

C 言語では，空白や改行などの見栄えを構成する字句の違いや，式内の余分な括弧の違いが記述されていても動作に差異がない．置換文字列と逆置換対象箇所との間の違いのうち，C 言語の文法で許されている記述の違いは C 言語の文法による表現の違いとした．また，C 言語の文法による表現の違いを，字句レベルの違いのみが含まれるもの，構文レベルの解釈が必要なもの，意味レベルの解釈が必要なものの 3 種類に分類した．

C 言語の文法による表現の違いの特徴には，逆置換対象箇所に含まれる表現の違いがどのような記述で表れるかを置換文字列から決定できない点がある．例えば字句レベルの違いで

3　表現の違いを考慮したマクロ逆置換方法の提案

表 1　表現の違いの分類
Table 1　Classification of multiple program representation.

表現の違いの分類		具体例
C 言語の文法による表現の違い	字句レベル	空白や改行，コメントの違い
	構文レベル	括弧の違い
	意味レベル	組み換えられた式
マクロ特有の記述方法による表現の違い	プリプロセッサ演算子によるもの	“#”，“##”
	マクロを記述する際の習慣によるもの	“do{...}while(0)”，“{...}”，“\”

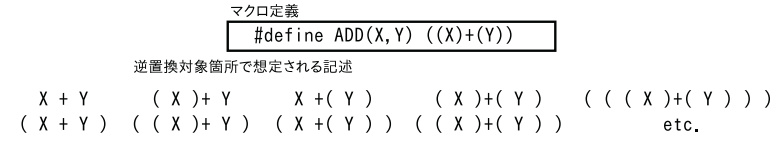

図 3　括弧による表現の違い
Fig. 3　Multiple parenthesis representation.

ある空白は，置換文字列から逆置換対象箇所内に記述される位置，数を決定できない．また，括弧については**図 3** に示すように，逆置換対象箇所で想定される括弧の記述は複数ある．

意味レベルの違いは本論文の調査では発見できなかったが，逆置換対象箇所内の式を組み換えて置換文字列を記述した場合など，意味レベルの解釈が必要である記述の違いを想定できることから表現の違いに加えた．意味レベルの表現の違いについても，字句レベルの表現の違い，構文レベルの表現の違いと同様に置換文字列の並びから逆置換対象箇所の記述を決定できない．

3.2　マクロ特有の記述方法による表現の違い

プリプロセッサ演算子を利用する際に変更される記述や，マクロを記述する際の習慣により変更される記述は，特定の字句をプリプロセッサ演算子に置き換えたり，字句の並びを変更することで表現の違いが生じる．よって，これらの字句の変化を特定することで置換文字列から逆置換対象箇所の記述を決定できる．そこで，これらをマクロ特有の記述方法による表現の違いとした．

前後の字句を結合する演算子である “##” が含まれていた場合には，その前後の字句を結合することで置換文字列から逆置換対象箇所を想定することができる．しかし，“##” を含むマクロ定義を逆置換する際には，表現の違いへの対応の他に，結合された字句の分割箇所を特定する必要がある．

表 2　表現の違いの吸収方法
Table 2　Matching methods of supporting multiple program representation.

想定可否	吸収方法	表現の違い
想定不可	逆置換対象箇所を正規表現で表す	C 言語の文法による表現の違い（字句レベル）
	ソースコード内の記述を統一	C 言語の文法による表現の違い（構文レベル）
想定可能	新たな候補パターンを生成	マクロ特有の記述方法による表現の違い

4.　表現の違いを考慮したマクロ逆置換方法

マクロ逆置換作業の自動化は，置換文字列と逆置換対象箇所との間にある表現の違いを吸収した上でソースコードの書換え操作を行なう必要がある．そこで，逆置換ツール内で表現の違いに対応した形に置換文字列またはソースコード全体を変形し，逆置換対象箇所と置換文字列との記述の違いを取り除く．なお，C 言語の文法による表現の違いの意味レベルとプリプロセッサを利用する表現の違いのうち “##” は逆置換の対象としない．これらの表現の違いはユーザーとの対話が必要であり，ツール化しても手作業にかかるコストと変わらないと考えるためである．

ソースコードの書換え操作は TEBA[12] を用いて，字句の書換え操作として実装した．TEBA を用いることで，ソースコードの書換え操作を字句のパターン変換として実現できる．マクロ逆置換操作においては，逆置換対象のマクロ定義から書換え前後の字句の並びを定義した書換えルールを生成することで書換えを行なえる．書換え後の記述に必要となる実引数は，正規表現を利用し逆置換対象箇所から抽出する．また，TEBA では前処理を行なう前のソースコードをそのまま構文解析でき，C 言語の構文としては不完全なマクロ定義の置換文字列も TEBA を用いて構文解析できる．

4.1　表現の違いの吸収方法

表 2 に表現の違いの吸収方法を示す．表現の違いには，置換文字列から逆置換対象箇所の記述を想定できるものとできないものがある．逆置換対象箇所の記述を想定可能な表現の違いは書換えルールの亜種を追加することで対応する．逆置換対象箇所の記述を置換文字列から想定できないものは，正規表現により逆置換対象箇所を記述することで対応する．しかし，括弧に関しては TEBA の構文解析器が演算子の優先度と結合規則に従った書換えに対応しておらず，そのまま書換えを行うことができない．よって，式に括弧を追加する正規化をほどこすことで，表現の違いを吸収する．

4　表現の違いを考慮したマクロ逆置換方法の提案

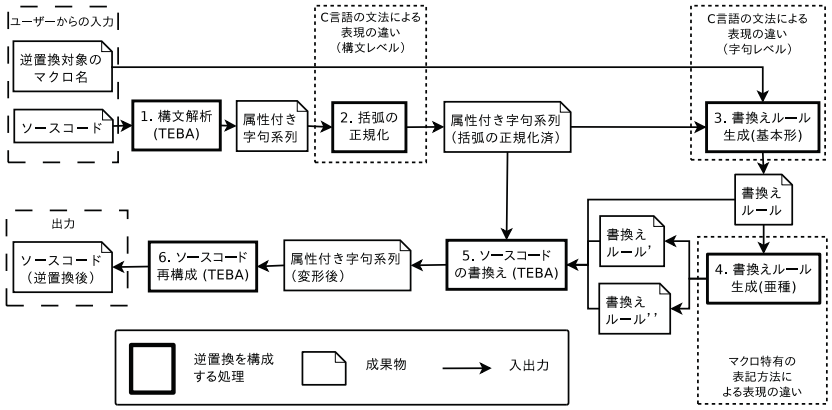

図 4　逆置換操作を構成する処理
Fig. 4　Process of macro reduction.

4.2　逆置換操作

図 4 に逆置換操作を構成する処理を示す．逆置換自動化ツールは，逆置換対象となるソースコードと逆置換をしたいマクロ名を入力することで，ソースコード内に含まれる逆置換対象箇所がマクロ参照へと置き換えられたソースコードを出力する．以下に各処理の詳細を説明する．

4.3　構 文 解 析

ソースコードを TEBA の構文解析器を利用し，属性付き字句系列に変換する．属性付き字句系列は，ソースコードの各字句に種別情報と括弧等の対応付けの情報が付加されており，Perl の正規表現を利用することで構文木による書換えに近い書換えを行なえる．

マクロ逆置換を行なう際には，ソースコード内から逆置換対象箇所となる正しいコード片を判定するために，置換文字列も構文解析する必要がある．ソースプログラムの TEBA の構文解析を行なった後にマクロ定義内の置換文字列のみを抜き出し，置換文字列のみを TEBA で構文解析する．

4.4　括弧の正規化

括弧の正規化とは，ソースコード内のすべて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．**表 3** 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とで括弧の記述を統一する．これにより，文レベルまでの構文解析までしか行われない TEBA の構文解析器を利用して，演算子の順位と優先度に従った書換えを行なえ，括弧の違いを吸収できる．

表 3　括弧のつけ方
Table 3　Patterns of parentheses description.

演算子の種類	括弧のつけ方
単項演算子	（ 演算子（ オペランド ）） （（ オペランド ）演算子 ）
二項演算子	（（ オペランド ）演算子（ オペランド ））
三項演算子	（（ オペランド ）？（ オペランド ）：（ オペランド ））

4.5　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．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．

C 言語の文法による表現の違いの字句レベルの違いは逆置換対象箇所を正規表現で表すことで吸収する．置換文字列内に含まれる空白や改行，コメントを一度削除し，「空白文字[⋆1]，コメントの 0 回以上の繰り返し」というパターンを代わりに挿入することで逆置換対象箇所の記述に対応する．

4.6　表現の違いを考慮した書換えルールの生成

マクロ特有の記述方法による表現の違いでは，置換文字列のパターンを書換えた新たなルールを追加する必要がある．そこで，置換文字列のパターンを書換えた新たなルールを生成し，複数の書換えルールを用意する．具体例として，複数文の置換文字列を囲う記述である “`do{...}while(0)`” が置換文字列に記述されていた場合には，**図 5** に示すように該当箇所を削除したルールを生成し，書換えルール群に追加する．新たなパターンの生成には置換文字列を TEBA の書換え系を用いて書換えることで実現している．よって，あらかじめパターンの変形を定義したメタルールを用意しておくことで，それらを組み合わせた新たなルールが自動的に生成される．

⋆1 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

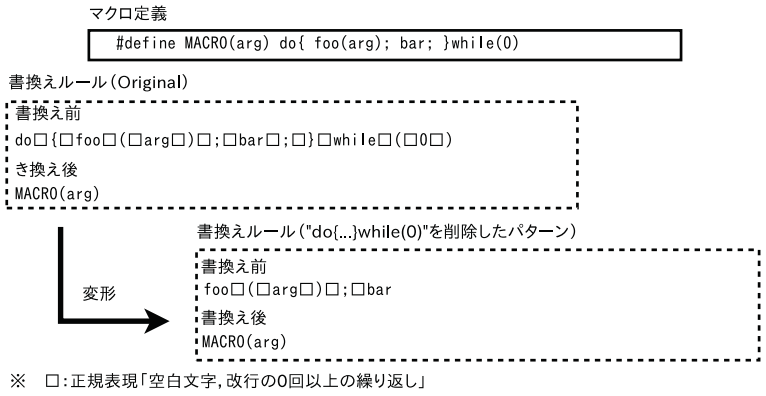

図 5　書換えルール亜種の生成
Fig. 5　Creation of a new rewrite rule from the original rule.

4.7 実　　装

提案方法を Perl を用いて実装し，実装規模は**表 4** の通りである．**表 5** にマクロ逆置換ツールの実行時間を示す[⋆1]．マクロ逆置換ツールの実行時間は，ほぼ括弧の正規化にかかる時間に等しい．括弧の正規化では，ソースコードに含まれる演算子すべてに対し判定と書換えを行っており，ソースコードに含まれる演算子数 n に対して $O(n^2)$ に増えていくと想定される．他の処理に関してはほぼ実用時間内に収まっており，括弧の正規化以外の方法を検討するなど対策が必要である．

5. 評価と考察

逆置換ツールが表現の違いを含む逆置換対象箇所へ正しく逆置換を行なうことができるか評価を行なった．マクロ逆置換が正しく行われれば，第 2.2 節で述べたように，マクロを利用して重複したコード片を除去できる．

マクロ逆置換自動化ツールの評価には，gnu-coreutils 8.7[11] を用いた．理由としては，Gnu のコーディング規約[13] 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．

⋆1 CPU: PentiumD 2.80GHz, Mem: 2.0GiB, OS: CentOS 5.6

表 4　実装規模
Table 4　The number of lines of the macro reduction tool.

処理	行数	関連ファイル	行数
マクロ定義内構文解析	60	括弧の正規化定義ファイル	397
括弧の正規化	91	置換文字列変化パターン定義ファイル	36
書換えルール生成	394	（“do{...}while(0)” の削除）	
書換えルール生成（亜種）	361		

表 5　実行速度
Table 5　The processing speed of the macro reduction tool.

ファイル名	行数	マクロ名	構文解析 (s)	括弧の正規化 (s)	マクロ逆置換 (s)	ソースコード再構成 (s)	全行程連続 (s)
src/cut.c	896	ADD_RANGE_PAIR	2.6	65.4	1.5	0.2	69.6
src/dd.c	1954	output_char	8.9	3638.4	4.6	0.4	3655.6

5.1 評 価 方 法

対象となるマクロ定義を gnu-coreutils 8.7 の中から抽出し，それぞれのマクロ定義に対しマクロ展開を行なう．マクロの展開には独自ツールを作成し利用した．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すべての字句のパターンを確認する．

gnu-coreutils 8.7 から抽出した評価対象のマクロ定義は 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．

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，正しく逆置換が行われたかを確認する．

5.2 評 価 結 果

第 5.1 節に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，一つのマクロ定義を除き，正しく逆置換が行われた．逆置換結果が行われなかったマクロ定義は “`do{...}while(0)`” 内が if 文の中括弧で囲われ，文がセミコロンで終わらない記述であった．このような記述の箇所にマクロ定義を逆置換すると，セミコロンが欠落した箇所が生成される．そこで，逆置換後にセミコロンが不足する箇所に新たにセミコ

ロンを付加する処理を追加し，評価を行ったところ，すべてのマクロ定義において正しく逆置換が行われた．

5.3 考　　察

逆置換が正しく行われない可能性としては以下のものがあると考えていた．

- 不適切な範囲の置換
- 展開前と逆置換後に生じる表現の違い
- 未適用箇所の発見

不適切な範囲の置換とは，同じ字句の並びであるが異なる意味を持つ箇所への逆置換が行われた場合や，逆置換結果が入れ子構造となる逆置換対象箇所にマクロ逆置換が適用されることで可読性が低下するといった問題である．

本論文の調査でこのような不適切な範囲へ置換されなかった要因としては，重複したコード片解消を目的に利用される文や式を定義したマクロ定義に対し評価を行なったことで，定数や単純な式を定義したマクロに比べ置換文字列に含まれる字句が多くなり，不適切な範囲へ逆置換されにくくなったと考えられる．

展開前と逆置換後に表現の違いが生じる場合の具体例としては，逆置換対象箇所に冗長な括弧が記述されていた場合にはマクロ参照の引数内やマクロ参照を覆う形で逆置換後に残る場合がある．しかし，マクロ定義を記述する際にはマクロの副作用を防ぐ目的で，括弧が念入りに記述されていることが多く，多くの場合で逆置換対象箇所より括弧が多く記述される．よって，ソースコード内に通常記述されている括弧の記述においては影響しなかった．

未適用箇所の発見は評価実験の中では見つけることができなかった．ソースコードにすでにあるマクロ定義を展開し，対応している表現の違いを書き加えたことで，未適用箇所が残らなかったことが原因だと考える．評価対象のソースコードを他のものに変更することや，他の評価方法を検討し，より多くの検証を行なうことが必要である．

評価の際に発見したセミコロンの欠落の問題は，表現の違いによって逆置換対象箇所外の記述も変化することを示している．本論文ではマクロ定義と逆置換対象箇所に含まれる表現の違いについてのみ分類したが，セミコロン以外に逆置換対象箇所外で変化する記述がないか調査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，ユーザーによる実引数の分割箇所やマクロ逆置換適用箇所の選択を可能にし，“##” 演算子による文字列結合と意味レベルの表現の違いに対応することが挙げられる．括弧の正規化については別の方法を考察し，実行時間を実用可能なレベルまで上げることが必要である．また，本研究で対象としていない関数を利用した重複したコード片の解消についても，関数を記述する際の表現の違いを考察し，自動化する必要がある．

謝辞　本研究の一部は，文部科学省研究費補助金基盤 (C)(課題番号:21500042, 22500036, 22500037) の助成を受けた．

参 考 文 献

1) Fowler, M.: *Refactoring: Improving the Design of Existing Code,* Addison-Wesley (1999).
2) Garrido, A. and Johnson, R.: Challenges of refactoring C programs, *Proceedings of the International Workshop on Principles of Software Evolution* (2002)
3) McCloskey B. and Brewer, E.A.: ASTEC: a new approach to refactoring C, *ESEC/SIGSOFT FSE*, pp.21–30 (2005).
4) Vittek, M.: Refactoring browser with preprocessor, *European Conf. Software Maintenance and Reengineering*,pp.101–110 (2003)
5) 藤原博文：C プログラミング専門課程，技術評論社 (2000).
6) 西田親生：ANSI C プログラミング，啓学出版 (1991).
7) 佐々木一郎：C 言語入門，朝倉書店 (1992).
8) 柴田望洋：新版 明解 C 言語 実践編，ソフトバンククリエイティブ株式会社 (2004).
9) 内田智史：C 言語によるプログラミング [応用編]，オーム社 (1992).
10) 矢沢久雄：プログラミングのセオリー，技術評価 (2009).
11) Free Software Foundation, Inc.: Coreutils - GNU core utilities (online), available from ⟨http://www.gnu.org/software/coreutils/⟩ (accessed 2010-10-12).
12) 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 (2010).
13) Free Software Foundation, Inc.: GNU Coding Standards (online), available from ⟨http://www.gnu.org/prep/standards/standards.html⟩ (accessed 2010-11-16).