ソフトウェア工学 115－4
（1997. 7. 31）

関数スライサによるプログラム部品抽出手法とその応用

岩本 奈美　　山本 晋一郎　　阿草 清滋

名古屋大学工学部
464-01 名古屋市千種区不老町

概要

ソフトウェアの再利用はソフトウェアの生産性や品質の向上などの利点がある．このため既存のソフトウェアからある機能を抽出して再利用することは有効なことである．プログラムから部品を抽出するためにはプログラムを理解し，必要な部分を探すプロセスが必要である．このプロセスは手間がかかるため自動化することが望ましい．本稿では，まず既存のソフトウェアから指定した機能の実現に必要な部分を関数単位で抽出する手法を提案し，さらにその実現方法について報告する．

Function slicing and its application

Nami Iwamoto, Shinichirou Yamamoto and Kiyoshi Agusa

School of Engineering, Nagoya University
Furo-cho, Chikusa-ku, Nagoya, 464-01

abstract

Software reuse gives improvement of software productivity and quality. It is effective to extract a function from existing software and reuse it. To extract components from a program, we need understand the program and find necessary parts. But it is difficult, so automation is desirable. In this paper, first we propose a method of extracting parts necessary for realization of a specified function from the software, furthermore report about its implementation.

1　はじめに

ソフトウェアの再利用はソフトウェアの生産性や品質の向上に大きな役割を果たす [3]．再利用可能なソフトウェア構成要素として，ソースプログラム，データ，仕様書，設計書，構成書，テストスイートなどが考えられるが，本稿ではソースプログラムの再利用について考える．本稿におけるソースプログラムの再利用とは，あるソフトウェアからソースプログラムの一部分をプログラム部品として抽出し，それを別のソフトウェアで利用することである．ただし，抽出されたプログラム部品は，そのまま分割コンパイル可能であるものとする．

既存プログラムの中に再利用できる部分が存在するとしても，実際にプログラム部品として使用するためには，その部品の実現方法を理解し，必要な部分を判断し，抽出するというプロセスが必要である．このプロセスを実用的な規模のプログラムを対象として行なうのは困難であるため，その実現方法に深く立ち入ることなく，機能のみから既存のプログラムを再利用することが望まれる．

たとえば，あるプログラムが漢字コードの変換機能を持つこととその機能が関数 convert_jis_to_euc() によって提供されることをプログラマーが知っていたとしても，この関数を再利用するためには，使用されている型やマクロの定義，副関数など多くの構成要素を正確に抽出する必要がある．その作業は，正確さが要求され，あまり創造的ではなく，非人間的な作業とすらいえる．したがって，再利用できる部分をプログラム部品として切り出す作業を自動化すること，あるいはその作業を支援するツールを作成することの意義は大きい．

また，プログラム理解においても，ある機能の実現に関係する部分をプログラムの中から特定できれば，可読性が著しく向上する．その結果，プログラム全体から一部の機能について理解するよりもプログラム理解にかかる労力が軽減されるため，関数スライサの技術はプログラム理解の支援ツールにも有効である．

本稿では，広く用いられている手続き型言語であるC言語を対象として，ソースプログラムから関数を指定することによってプログラム部品を抽出する手法を提案する．すなわち，既存のC言語プログラムから，ある機能を実現する関数 f を指定して，関数 f を利用するのに必要な要素を抽出する．それらの要素を同定しプログラム部品として抽出する．さらに，関数 f' に関しても上記の要素を抽出する．

以下，2章では関数スライサについて，3章では関数単位での部品抽出のためのツールの実現について，4章ではツールの評価，5章では関数スライサの応用と今後の課題について示す．

2 関数スライサ

スライシングとは，プログラム中のある文 s のある変数 v に注目し，s における v の値に影響を与えるすべての文を，変数の依存関係を利用して抽出する技術である [1]．本稿ではプログラム中のある関数 f に注目したときに，その関数の中で使われているすべての識別子 i に関して，i が定義または宣言されている部分をすべて抽出することを関数スライシングと定義する．識別子 i は，関数名，変数名，型名，マクロを指す．

2.1 関数スライサが抽出する要素

ある関数をプログラムの中からスライスするときにプログラムから抽出しなければならない要素を考える．既存のC言語プログラムから，ある機能を実現する関数 f を指定して，関数 f を利用するのに必要な

- 関数 f の定義とプロトタイプ宣言
- 関数 f が呼び出している関数 f' の定義とプロトタイプ宣言
- 関数 f 中で参照・代入している大域変数・構造体などの定義
- 関数 f 中で使われている型の宣言
- 関数 f 中で使われているマクロの定義

などを同定しプログラム部品として抽出する．さらに，関数 f' に関しても再帰的に上記の要素を抽出する．しかし関数 f が再帰的な関数であったときなど，一度抽出された関数については重複して抽出することはしない．

2.2 関数スライサのアルゴリズム

関数スライサのアルゴリズムを以下に示す．

入力：$I = \{i | i$は関数名$\}$
出力：必要な定義の集合G

$D = I$の定義の集合
$G = \phi$
while $(D \neq \phi)\{$
　$D = D - \{d\}, d \in D$：定義の集合から要素 d を 1 つ取り出す
　$G = G \cup \{d\}$
　$D' = \{d' | d$に必要な定義$\}$
　$D = D \cup D' - G$：d に必要な定義 d' が G と D に含まれていなければ D に入れる
$\}$

G 中の定義がヘッダファイルにある場合，その定義を取り出し #include 文を追加する．

このアルゴリズムの停止性はプログラム中の定義は有限であることにより示される．

2.3 アルゴリズムの制約

2.3.1 関数へのポインタの扱い

図 1のようなプログラムを考える．

```
 1 int g1(int i);
 2 int g2(int i);
 3
 4 int f(int x);
 5 {
 6   int (*func)();
 7
 8   if (...)
 9     func = g1;
10   else
11     func = g2;
12   (*func)(x);
13   ...
14}
```

図 1: 関数へのポインタ

図 1で関数 f を抽出する場合，関数 g1, g2 は抽出することができる．もし，ポインタ変数 func が外部で宣言されていて，func への代入が関数 f の外で行なわれていたら g1, g2 を抽出することはで

きない．このような場合，引数の個数と型，戻り値の型を調べて候補となる関数を見つけることはできるが，関数へのポインタの扱いには限界がある．そこで，ユーザの判断によって，必要な関数を複数個指定することで解決できる．

2.3.2 大域変数の扱い

関数 f を抽出するとき，f で参照されている大域変数 v への代入が f によって抽出されない関数 g 中で行なわれている場合，v の代入文を抽出することができない．これは，f の抽出を行なうときに大域変数の値がセットされているかどうか調べ，されていなければユーザに警告することで回避する．また，これを逆に考えると関数でスライスを行なうと大域変数 v の代入を抽出できないことがあるため，大域変数でスライスを行なうことにより，大域変数を参照・代入しているすべての関数を抽出できる．

2.4 関数スライサの例

上記のアルゴリズムでプログラムの中から関数に必要な定義・宣言部分を抽出することができる．例として図2のスタックを用いて入力文字列を逆順で表示するプログラムから関数 push を関数スライサを用いて抽出したものを図3に示す．

```
 1 #include <stdio.h>
 2
 3 #define MAXVAL 100
 4
 5 char val[MAXVAL];
 6 int  sp = 0;
 7 void push(char);
 8 char pop(void);
 9
10 void push(char c)
11 {
12   if (sp != MAXVAL)
13     val[sp++] = c;
14   else
15     fprintf(stderr, "stack full\n");
16 }
17
18 char pop(void)
19 {
20   if (sp != 0)
21     return val[--sp];
22   else
23     return NULL;
24 }
25
26 void main()
27 {
28   char c;
29
30   puts("enter string--->");
31   while((c=getchar())!=EOF && c!='\n')
32     push(c);
33   while((c = pop()) != NULL)
34     putchar(c);
35   putchar('\n');
36 }
```

図 2: 対象プログラム

入力 : I = {push}

$D0$ = {push() の定義とプロトタイプ宣言 }
$G0 = \phi$

$d1$ = push() の定義
$D1$ = {push() のプロトタイプ宣言 }
$G1$ = {push() の定義 }
$D1'$ = {sp, MAXVAL, val[], fprintf() の定義 }

$D2$ = {push() のプロトタイプ宣言, sp, MAXVAL, val[], fprintf() の定義 }

同様に D が空になるまで行なう．

結果
G = {push() の定義, push() のプロトタイプ宣言, sp, MAXVAL, val[], fprintf() の定義 }

fprintf() の定義は #include (stdio.h) の中に入っているので，G から fprintf() の定義を取り出し，#include (stdio.h) を追加する．

```
 1 #include <stdio.h>
 2
 3 #define MAXVAL 100
 4
 5 char val[MAXVAL];
 6 int  sp = 0;
 7 void push(char);
 8
 9 void push(char c)
10 {
11   if (sp != MAXVAL)
12     val[sp++] = c;
13   else
14     fprintf(stderr, "stack full\n");
15 }
```

図 3: 関数 push に関してスライス

3 関数単位での部品抽出ツールの実現

実際に関数スライサの手法を用いて既存プログラム中のある関数とそれに必要な部分を抽出するツールの実現について述べる．関数スライサを作成するためにはソフトウェアの構文的な情報が必要である．これを獲得するためにソフトウェアデータベース (SDB) を利用する．部品抽出ツールを利用する全体の流れを図 4に示す．

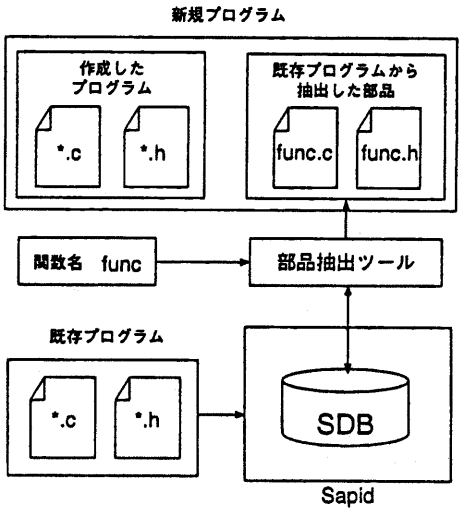

図 4: システム構成

3.1 SDB

Sapid (Sophisticated APIs for CASE tool Development) は，既存のリポジトリが扱うことのできない細かな構成要素を管理する細粒度ソフトウェア・リポジトリに基づいた CASE ツール・プラットフォームである．Sapid は，ソフトウェアの構造・構成要素・関連が格納されている SDB (Software DataBase)，この SDB にアクセスするための C 言語のライブラリである AR (Access Routines)，そして，ソフトウェア操作言語 SML (Software Manipulating Language) で構成される．SDB は，C 言語のソースプログラムを 12 種類の実体と 29 種類の関連としてモデル化したソフトウェアモデルに基づいて解析し，その結果得られた実体・関連情報を管理する．

3.2 複数の関数の指定

対象となる関数を 1 つのプログラムの中にある関数であれば複数指定できる．図 2のスタックを利用した関数 push と pop のように関連のある機能は 1 つのモジュールにすることはモジュールの独立性を高めることになるので望ましい．これを実現するために関数を複数個指定する．

3.3 ファイルの生成

抽出した要素を部品として利用するために関数名にそれぞれ.h, .c をつけたヘッダファイルとソースファイルを生成する. ソースファイルの中には関数の定義部分, static 宣言された大域変数やプロトタイプ宣言と初期化している大域変数を含む. また, ヘッダファイルにはソースファイルに含まれないインクルード文, 型宣言, 構造体型の宣言, 大域変数と初期化されている大域変数の extern 宣言を含む. extern 宣言文は元のプログラムに存在しない場合は, 新たに生成して追加する.

3.4 実行結果

図 2のプログラムで関数 push を部品として抽出した結果を図 5と図 6に示す.

```
 1 #include "push.h"
 2
 3 int  sp = 0;
 4
 5 void push(char c)
 6  {
 7  if (sp != MAXVAL)
 8    val[sp++] = c;
 9  else
10    fprintf(stderr, "stack full\n");
11 }
```

図 5: push.c

```
1 #include <stdio.h>
2
3 #define MAXVAL 100
4
5 char val[MAXVAL];
6 extern int  sp;
7 void push(char);
```

図 6: push.h

4 評価

実際に作成したツールを評価するために既存のプログラムからある機能を実現している関数を抽出し, その部品を利用して新しいプログラムの作成を行なった. この実験を行なった環境は計算機 SGI 社製の Indy, CPU R4600PC(133MHz), メモリ 64MB である.

対象とするプログラム nkf はネットワークでメールやニュースの読み書きをするために作られた, 漢字コードの変換フィルタである. nkf の認識できる漢字コードは JIS コード, EUC コードとシフト JIS コードで, オプションによって変換するコードを選択できる.

JIS コードを EUC コードに変換するプログラムを作成するときに, nkf の EUC コードに変換している関数 e_oconv を利用することを想定して, e_oconv を部品として抽出する実験を行なった. nkf の行数と新しく作成したプログラム convert_jis_to_euc の行数と抽出した部品の要素の種類と個数を図 7に示す. また, 部品作成にかかった時間は 1 分 30 秒であった. 部品抽出にかかる時間の大部分が構文解析を行なうために利用した Sapid の SDB のアクセスに費やされている. 現在 Sapid では高速化を進めており, このツールの高速化も今後期待できる.

ファイル		行数	
対象プログラム nkf.c		1583	
抽出した部品	e_oconv.c	210	351
	e_oconv.h	33	
新しく作成したプログラム main.c		108	

要素	個数
インクルード文	1
関数定義	3
変数定義	9
マクロ	6

図 7: 部品抽出の評価

5 関数スライサの応用と今後の課題

今回は関数スライサを用いて部品抽出ツールの作成を行なったが, 関数スライサは部品抽出だけでなく, 他の用途にも利用可能である.

5.1 main 関数を対象とする関数スライス

プログラムを作成していて, プログラムの一部を修正したりすると, 必要のない大域変数などの定義を消し忘れたり, インクルードしたヘッダファイルが不必要になってしまう場合がある. このようなプログラムの不必要な部分を見つけ出すために関数抽出が利用できる.

プログラムの実行は main 関数の最初の文から開始される. 従ってこのツールの指定する関数名を main 関数にすればこのプログラムで必要な部分はすべて取得できる.

このことを逆に考えるとプログラムの中で必要ないものは抽出されないということになる. そこで抽出されなかった定義部分をプログラムの中から見つけ出してくればプログラムを実行するのに不必要な部分を求めることができる. よって, プログラムの最終的なチェックを行なうことにも利用できる.

また, Sapid ではプリプロセッサによる前処理の条件コンパイルに関して条件が真になる範囲に関してマクロの定義・展開情報が生成される. これにより必要のない条件がすべて取り除かれたプログラムの生成も可能である.

5.2 プログラムのファイルの分割

プログラムは関数を組み合わせて作成されるが, 大きなプログラムが 1 つのソースファイルになっていると読みにくく, また若干の修正を行なっただけでも必要のない関数まで再コンパイルしなければならない. このため, いくつかのファイルに分割することが必要となる. 関数スライサを利用すればファイルを分割することが可能である.

関数 f と f から呼ばれている関数を別のファイルに分割行なうことを考える. まず, 部品抽出の要領で f を指定してそれに必要な要素を抽出する. 次に抽出した関数以外の関数について必要な要素を抽出し, 2 つに共通するものをヘッダファイルに入れ 2 つのファイルにインクルードすることによって実現できる.

5.3 大域変数によるスライス

図 2の push, pop 関数は大域変数であるスタックポインタ sp を参照していることが分かる. この関数が別のモジュールに分かれているとモジュール結合度が高い共通結合となっているためモジュー

ルの独立性が低くなってしまう [4]. モジュール結合度を低くするためには大域変数を参照している関数を1つのモジュールにすることが望ましい. これを実現するために大域変数に関してスライスを行なう. こうすることによってモジュール強度の高い情報的強度のモジュールとなり, モジュールの独立性を高めることができる.

6 おわりに

本稿では, 既存プログラムの中には再利用可能な機能が存在することに注目して, 再利用を行なうために既存プログラムから必要な部分を自動的に抽出する手法を考察した. 具体的にはプログラムから抽出する単位として関数を考え, 関数に必要な部分をプログラム中から抽出する関数スライサを提案した. また, 関数スライサを利用して実際に関数を指定すると部品を作成する部品抽出ツールを作成した. 今後は部品抽出やモジュール分割などをより使いやすいものにしていくことが重要である.

本稿に際して熱心に御討論頂いた阿草研究室の皆さんに深く感謝致します.

参考文献

[1] M.Weiser, "program slicing", IEEE Trans. Software Eng., vol. SE-10, pp.352-357, Jul. 1984

[2] 下村 隆夫, "プログラムスライシング技術と応用", 共立出版, 1995

[3] Carma McClure, ベスト CASE 研究グループ訳, "ソフトウェア開発と保守の戦略 - リエンジニアリング・リポジトリ・再利用 -", 共立出版, 1993

[4] 山田 茂, 高橋 宗雄, "ソフトウェアマネジメントモデル入門 - ソフトウェア品質の可視化と評価法 -", 共立出版, 1993

[5] B.W. カーニハン, D.M. リッチー, 石田晴久訳, "プログラミング言語 C 第2版", 共立出版,1996

[6] 橋本 靖, 山本 晋一郎, 阿草 清滋, "Program Slicing を利用したプログラムカスタマイザ", 電子情報通信学会ソフトウェアサイエンス研究会, Vol.94, No.10, pp.73-80,1994

[7] 山本晋一郎, 阿草清滋, "細粒度リポジトリに基づいたツール・プラットフォームとその応用", 情報処理学会ソフトウェア工学研究会, Vol.102, No.7, pp.37-42, 1995