ソフトウェア工学 105－7
（1995．9．8）

区間限定スライスを用いた部品作成システムの評価

丸山 勝久　島 健一　高橋直久
{maru, kshima, naohisa}@slab.ntt.jp

NTT ソフトウェア研究所
〒180 東京都武蔵野市緑町 3-9-11

部品を用いたソースコード再利用において，我々は区間限定スライスを定義し，このスライスを用いて既存のソースコードから再利用部品を容易に抽出する手法を先に提案した．本稿では，本手法に基づき構築した部品作成実験システムを用いて，実際のプログラムからソフトウェア部品を作成した際の実験結果と考察を述べる．本手法は，部品作成対象プログラムに対して任意の区間を指定して部品抽出が可能であるという点で従来の手法より有利である．また，本手法では再利用対象の部品本体だけでなく，部品を実行するために必要な実行条件例と部品がどのように利用されているかを示す利用例を合わせた付加情報を生成し，部品の実行動作や利用方法の確認が容易になる．本手法の利点を確認するための評価項目として，(1) 抽出した部品は実行に不要なコードを含まないか，(2) 区間を変化させることで機能の異なる部品が作成可能か，(3) 異なるプログラムから抽出した同じ機能の部品はコードが等しいか，(4) 生成される付加情報は部品を再利用するのに役立つか，の 4 点を設定した．

Evaluation of Software Reuse System for Extracting Software Components by Bounded Slices

Katsuhisa MARUYAMA, Ken-ichi SHIMA and Naohisa TAKAHASHI

NTT Software Laboratories
3-9-11 Midori-cho Musashino-shi Tokyo 180, Japan

A previously proposed software reuse system based on extracting reusable software components by bounded slices was experimentally evaluated. In this system, arbitrary bounds and variables can be specified to extract any component from existing source code; code not needed to execute the extracted component are eliminated. Extracted components are the body of the component, the execution conditions, and instances of utilization, which are embedded in a description program, the codes needed to execute the component, and examples of how to use it. The execution conditions and instances of utilization are called additional descriptions; they facilitate understanding and modification of the component. To demonstrate the advantage of our method, this system was evaluated by focusing on four questions: 1) Does the extracted component not include superfluous code? 2) Can various and many components be extracted from one source code by specifying the bounds? 3) Do components having the same function have equivalent code? 4) Are additional descriptions useful in reusing components?

1 はじめに

部品を用いたソースコード再利用では，必要な部品があらかじめライブラリに用意されているとき，プログラムの開発コストは減少する．しかし，あらゆる開発領域で再利用可能な部品をすべて用意しておくことは不可能である．特に，個人あるいは小規模プロジェクトでは，プロトタイプの作成など短期的で実験的な開発を行なう機会が多く，必要な部品は頻繁に変化する．よって，部品作成はソフトウェア開発者に大きな負担を与える．

これに対し，部品作成の負担を軽減するためには，既存のソフトウェアから再利用部品を抽出することが重要であると指摘されている[1,2]．既存のソフトウェアから部品を抽出する場合，以下に示す2つの問題が存在する．

問題1: 作成する部品の機能を満たすコードだけをプログラムから過不足なく抽出することは難しい．

問題2: 部品の機能を理解する際，抽出された部品のコードだけを参照することで，実行の条件や利用方法を取得及び推測することは難しい．

これらの問題に対して，我々はプログラムスライシング[3]を用いたソフトウェア部品の作成手法を先に提案した[4]．ソフトウェア部品とは，再利用の対象となるプログラム部品と部品理解に役立つ付加情報を合わせたものである．問題1に対して，本手法ではプログラムスライシングにより依存関係を持つコードを既存のプログラムから部品として抽出する．その際，従来のスライシングを単純に適用するのではなく，到達可能経路で定義する区間という概念をスライシングに導入して用いる．これにより，部品作成者は抽出する部品の機能やサイズを従来より自由に決定することが可能となり，特定の機能のみを抽出できる．また，問題2に対しては，部品を実行する際や利用方法を取得する際に役立つコードをスライシングによって抜き出し，部品作成時に付加情報として部品本体に付加できる手法を提唱した．この付加コードは，従来の部品作成手法では生成されていない情報である．以上より，本手法は以下に示す利点を持つ．

利点1: 既存のプログラムにおいて，部品作成者が任意に指定した区間から，単独で実行が可能かつもとのプログラムと同じ実行結果を与える部品を半自動抽出可能であり，従来の部品に比べて不要な記述コードが削除できる．

利点2: 部品の動作を確認するために必要な実行条件を提供することで，多様なテストが可能である．また，部品の利用例を提示することで部品を再利用する際の指針を与えることができる．

我々は，本手法に基づき構築した部品作成実験システムを用いて，実際のプログラムから再利用部品を抽出する実験を行い，上記の利点を確認した．本稿の構成は次の通りである．2章で，従来のスライシングを部品作成に適用した際の問題点と，これらの問題を解決するために従来のスライスを拡張した区間限定スライスの定義を述べる．次に，3章で区間限定スライスを用いたソフトウェア部品の作成手法と例を示す．さらに，4章では本手法に基づいて部品作成の評価実験を行なった結果を示し，5章で部品抽出に関する利点1，及び部品理解に対して付加情報が担う利点2に関して考察する．最後に，今後の課題を6章で述べる．

2 プログラムスライシング

2.1 スライシングの適用における問題点

プログラムスライシングとは，プログラム内の任意の文において着目する変数に影響を与える一部のコードだけをもとのプログラムから抜き出すことである[3]．本手法では，部品作成対象プログラムをCFG (Control Flow Graph)[5]で表現し，CFGからPDG (Program Dependence Graph)[6]を作成する．スライスは，PDG上でデータ依存関係 (data dependence) と制御依存関係 (control dependence) をたどることによって得られ[7]，依存関係をたどる向きに応じて，順方向と逆方向の2種類が存在する．本稿では，節点 n の変数 v に関する順方向及び逆方向スライスをそれぞれ $S_f(n,v)$, $S_b(n,v)$ とおく．

プログラムスライシングを部品作成に適用した研究としては，文献8-11)がある．文献8)では，変数のみをスライシング基準として指定し，スライス (decomposition slice) を作成する．また，文献9)では，指定した変数に直接影響を及ぼす節点と，これらの節点を枝に含む分岐節点をスライス (direct slice) として作成する．文献10)では，モジュール間の区切りを維持したまま，スライス (interface slice) を作成する．さらに，文献11)では，プログラム制御の流れを決定する式をスライシング基準に追加し，スライス (conditioned slice) を作成する．

しかし，文献8, 9)のスライシングは，依存関係をたどる際に対象とするプログラムの範囲に関する制約を持たない．また，文献10)では関数の先頭，文献11)では分岐文だけがスライシングの範囲を決定する要素となる．このように，これらのスライシングでは対象範囲を部品作成者が自由に指定することはできない．よって，従来のスライシングを適用することで，依存関係だけに従い抽出した部品の機能やサイズは部品作成対象のプログラム構造に大きく依存し，部品作成者が意図しない不要なコードが含まれることがある．これは，既存のプログラムが部品作成者の希望する機能を含んでいても，その機能だけを持つ部品を抽出することができないことを指す．

また，作成するプログラムが頻繁に変化する領域では，部品をそのままの形であるいは他の部品と単純に合成するだけで目的のプログラムが得られる機会は少なく，部品変更が数多く生じる．部品を変更するためには，部品がどのような機能を持っているのかを理解することが不

可欠である．しかし，従来の手法は部品を作成することだけにスライシングを適用しており，部品を再利用する際に役立つ支援情報の生成は行っていない．すなわち，1章で述べた問題2は残されたままである．

2.2 プログラムスライシングの拡張

抽出された部品を再利用して目的のプログラムを作成する場合，その部品が適当な入力データのもとで実行可能であることが不可欠である．また，既存のプログラムから抽出したにもかかわらず，もとのプログラムと関連を持たない部品は，その機能を特定することが難しく，再利用するのに適さない．このため，抽出した部品はもとのプログラムを実行させたときと同じ結果を出力する必要がある．本稿では，これら部品の持つべき2つの性質をそれぞれ実行可能性及び抽出等価性と呼ぶ．

いま，もとのプログラム P と抽出された部品 C 内に含まれる変数の集合を V_P, V_C とする．このとき，$V_P \supseteq V_C$ となり，$V'_P \equiv V_C$ とおく．実行可能性とは，V'_P 内の各変数の値が P の実行後に決定されるとき，V_C 内の各変数の値が C の実行後に必ず決定されることを指す．抽出等価性とは，実行前の P と C において V'_P と V_C 内の各変数の値がお互いに等しいとき，実行後の P と C において V'_P と V_C 内の各変数の値がお互いにすべて等しいことを意味する．

従来の順方向及び逆方向スライスは，実行可能性及び抽出等価性を備えている[3]．本手法では，既存のプログラムにおいて部品作成者が任意に指定した範囲から部品を抽出可能とするため，従来の逆方向及び順方向スライスに区間という概念を導入する．このとき，単純に指定した範囲のコードをプログラムから切り出し，スライシングを適用して部品を抽出すると，部品の実行可能性や抽出等価性が失われる．例えば，繰り返し構造を含むコードを単純に切り出すと，繰り返し条件に関わる文が不用意に削除され，実行できない部品が作成される場合がある．また，実行できても抽出等価性は保証されない．

このため，本手法では部品作成者がCFGにおいて任意の2節点を指定し，これらの節点間の到達可能経路を考える．このとき，繰り返し文の内部からでも一部の連続した節点だけを抜き出すことを可能とするため，到達可能経路に制約を追加する．これを制約付き到達可能経路と呼び，この経路によりスライシングの対象区間を表す．さらに，制約付き到達可能経路を用いて，区間の指定が可能となるように従来の逆方向及び順方向スライスを拡張し，逆方向及び順方向区間限定スライスを定義する．これにより，スライシングによって抽出される部品は実行可能性と抽出等価性を満たすことが保証される．

2.3 制約付き到達可能経路

本手法では，部品作成者が指定する区間を上限節点 n_u と下限節点 n_l で表現する．このとき，区間はCFG上を節点 n_u から節点 n_l に矢印をたどるとき，通過する制約付き到達可能経路 (constrainted reachable path) とする．制約付き到達可能経路上にある節点の集合 $CRP(n_u, n_l)$ を以下のように定義する．

定義1 制約付き到達可能経路集合

$$CRP(n_u, n_l) \stackrel{\text{def}}{=} CRP_b(n_u, n_l) \cup CRP_f(n_u, n_l)$$

$CRP_b(n_u, n_l)$：下限節点 n_l から上限節点 n_u に下限節点 n_l を中間節点として通らず，逆方向に経路をたどるとき通過する節点の集合

$CRP_f(n_u, n_l)$：上限節点 n_u から下限節点 n_l に上限節点 n_u を中間節点として通らず，順方向に経路をたどるとき通過する節点の集合

いま，節点 n_u から節点 n_l に制約をつけずに到達可能な経路上にある節点の集合を $RP(n_u, n_l)$ と表す．定義1では，制約として中間節点の通過を禁止しているため，$CRP(n_u, n_l) \subseteq RP(n_u, n_l)$ となる．よって，節点 n_u や節点 n_l から到達不可能な節点が CRP に含まれることはない．また，制約は経路をたどる際に利用されるだけで，上限節点 n_u 及び下限節点 n_l はCFGの任意の位置に指定することが可能である．さらに，CRP は経路をたどるとき通過する節点を集めたものなので，必ず一意に決定され，CFG上で連続する節点集合となる．

2.4 区間限定スライス

定義1の制約付き到達可能経路を用いることにより，従来の逆方向及び順方向スライスを拡張したものを区間**限定スライス** (bounded slice)[4] と呼ぶ．

いま，節点 n_1 から節点 n_2 への変数 v に関する定義－参照データ依存関係 (flow dependence) を $n_1 \rightarrow_{d(v)} n_2$ とし，任意の変数に関するデータ依存関係を $n_1 \rightarrow_d n_2$ と表す．また，節点 n_1 から節点 n_2 への制御依存関係を $n_1 \rightarrow_c n_2$ と表す．もとのプログラムにおいて，指定した区間だけに関わる依存関係を区間限定依存関係と呼ぶ．指定した区間に含まれる節点の集合を B とするとき，逆方向及び順方向区間限定依存節点集合 $BD_b(B,n)$, $BD_f(B,n)$ を以下のように定義する．

定義2 区間限定依存節点集合

$$BD_b(B,n) \stackrel{\text{def}}{=} [\{m | m \rightarrow_d n\} \cup \{m | m \rightarrow_c n\}] \cap B$$
$$BD_f(B,n) \stackrel{\text{def}}{=} [\{m | n \rightarrow_d m\} \cup \{m | n \rightarrow_c m\}] \cap B$$

区間限定依存節点集合を用いると，PDG上の区間 B 内において節点 n の変数集合 V に関する通過節点集合 $BDS^k_b(B,n,V)$ と $BDS^k_f(B,n,V)$ の定義は以下のようになる．

定義3 区間限定通過節点集合

$$BDS^k_b(B,n,V) \stackrel{\text{def}}{=} \bigcup_{m \in BDS^{k-1}_b \setminus BDS^{k-2}_b} BD_b(B,m) \cup BDS^{k-1}_b(B,n,V)$$

$$\begin{cases} BDS_b^0(B,n,V) = DS_b^0(n,V) \cap B \\ BDS_b^{-1}(B,n,V) = \emptyset \end{cases}$$

$$BDS_f^k(B,n,V) \stackrel{\text{def}}{=} \bigcup_{m \in BDS_f^{k-1} \setminus BDS_f^{k-2}} BD_f(B,m) \cup BDS_f^{k-1}(B,n,V)$$

$$\begin{cases} BDS_f^0(B,n,V) = DS_f^0(n,V) \cap B \\ BDS_f^{-1}(B,n,V) = \emptyset \end{cases}$$

ここで，$DS_b^0(n,V)$ は節点 n の直後で変数集合 V の各要素が参照されていると仮定したとき，データ依存元となる節点の集合を指す．$DS_f^0(n,V)$ は節点 n の直前で変数集合 V の各要素が定義されていると仮定したとき，データ依存先となる節点の集合に，変数集合 V 内の変数の少なくとも一つが定義されている節点 n を追加したものである．また，$\bigcup_{m \in M} S(m)$ は，集合 M の各要素を $m_1, m_2, \cdots$ としたとき，$S(m_1) \cup S(m_1) \cup \cdots$ を表す．$A \setminus B$ は集合 A から集合 B の要素を取り除いたものである．

定義3において $k = 1, 2, \cdots$ とすることで，節点 n から逆方向及び順方向に依存関係をたどり，区間限定通過節点集合を求める．逆方向区間限定通過節点集合が収束したとき，収束後の集合を $BDS_b(B,n,V)$ とする．順方向についても同様に，収束したときの順方向通過節点集合を $BDS_f(B,n,V)$ とする．これらの区間限定通過節点集合を用いて，逆方向区間限定スライス (backward bounded slice) 及び順方向区間限定スライス (forward bounded slice) を定義する．節点 n で定義される変数集合を $DEF(n)$，参照される変数の集合を $USE(n)$ とすると，区間 $CRP(n_u, n_l)$ の変数集合 V に関する区間限定スライスの定義は以下のようになる．

定義4 区間限定スライス

逆方向区間限定スライス:

$$\hat{S}_b(n_u, n_l, V) \stackrel{\text{def}}{=} BDS_b(CRP(n_u, n_l), n_l, V)$$

順方向区間限定スライス:

$$\hat{S}_f(n_u, n_l, V) \stackrel{\text{def}}{=} \bigcup_{i \in BDS_f(CRP(n_u,n_l),n_u,V)} [S_b(i, USE(i)) \cup \{ i \}]$$

ここで，n_u, n_l は CFG で指定する上限節点及び下限節点，V は着目する変数集合である．逆方向区間限定スライス $\hat{S}_b(n_u, n_l, V)$ は，もとのプログラムの節点 n_l において変数集合 V のすべての変数に影響を与えるコードを，区間 $CRP(n_u, n_l)$ に関して抜き出したものである．また，順方向区間限定スライス $\hat{S}_f(n_u, n_l, V)$ は，節点 n_u において変数集合 V のすべての変数が影響を与える節点を区間 $CRP(n_u, n_l)$ に関して集め，それらの節点に対する逆方向スライスを和集合で結合して作成する．

図1に逆方向区間限定スライス $\hat{S}_b$ と順方向区間限定スライス $\hat{S}_f$ の例を示す．図1の CFG において，$\hat{S}_b(3, 5, \{\text{sum}\})$ は $S_b(5, \{\text{sum}\})$ に対し，節点 1 を含まない．また，$\hat{S}_f(5, 8, \{\text{sum}\})$ は $S_f(5, \{\text{sum}\})$ に対し，節点 2,6,10,11 を含まない．

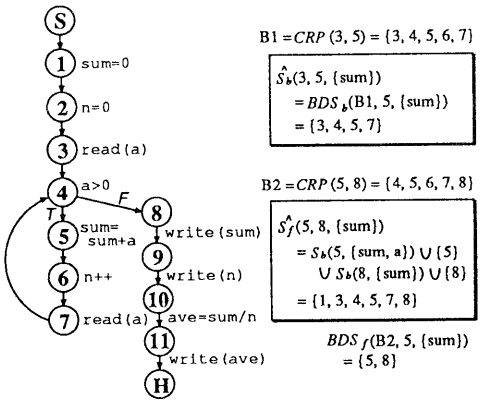

図 1: 区間限定スライスの例

3 ソフトウェア部品の作成手法

3.1 対象プログラム

本手法では，部品作成対象プログラムとして手続き型言語 C のサブセットを用いる．プログラムを構成する文として，式 (代入文，関数呼出し)，複文，選択文 (if, switch)，繰り返し文 (while, do, for)，ジャンプ文 (return, break, goto) を持つ．再帰呼出しに対しては制約付き到達可能経路を求めことができないので，本手法の対象から除く．ジャンプ文を含むプログラムを部品作成対象とする際には，APDG(augmented PDG)[12] を利用してスライスを求める．

また，本手法の適用はデータ依存解析後であるので，本手法が扱える言語の変数の型は対象プログラムに制限されない．しかし，C 言語のすべての変数の型をデータ依存解析可能とみなすのは難しく，本稿ではスカラ型，配列型と一次ポインタのみを扱う．さらに，C 言語の標準ライブラリで提供される関数 (printf, getc など) については依存解析済みとし，引数間の依存関係はあらかじめデータとして持つこととする．

3.2 ソフトウェア部品

本手法に基づく部品作成システムの概要を図 2に示す．ソフトウェア部品は，ソフトウェア開発者が部品作成基準 C_C(component criterion) を指定し，スライシングを適用することで作成する．部品作成基準とは，作成する部品の機能を決定する際に着目する区間 $CRP(N_u, N_l)$ と変数 V を 3 つ組で表現したものである．

部品作成基準 $C_C = < N_u, N_l, V >$ におけるソフトウェア部品の定義を次に示す．

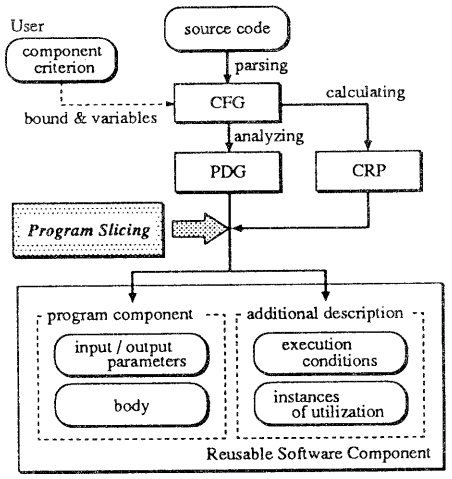

図 2: 本手法に基づく部品作成システムの概要

定義 5 ソフトウェア部品 (SLICE)

部品本体 (body)[節点集合] :

$$C_{body} \stackrel{\text{def}}{=} \hat{S}_b(N_u, N_l, V)$$

入力パラメータ (input parameters)[変数集合] :

$$C_{in} \stackrel{\text{def}}{=} \bigcup_{m \in DD_{in}(C_{body})} DEF(m)$$

出力パラメータ (output parameters)[変数集合] :

$$C_{out} \stackrel{\text{def}}{=} (V\,;\, V_d\,;\, V_s\,;\, V_e)$$

$$\begin{cases} V_d = \bigcup_{m \in C_{body}} DEF(m) \\ V_s = \{w \mid w \in V_d \wedge \hat{S}_b(N_u, N_l, w) \subseteq C_{body}\} \\ V_e = \bigcup_{m \in DD_{out}(C_{body})} DEF(m) \end{cases}$$

実行条件例 (execution conditions)[節点集合] :

$$C_{cond} \stackrel{\text{def}}{=} S_b(N_l, V) \setminus C_{body}$$

利用例 (instances of utilization)[節点集合] :

$$C_{inst} \stackrel{\text{def}}{=} \hat{S}_f(n_r, n_{rk}, V) \setminus C_{body}$$

$$\begin{cases} n_r \in \bigcup_{v \in V} \{n \mid m \rightarrow_{d(v)} n \\ \qquad\qquad \wedge\, m \in C_{body} \;\wedge\; n \notin C_{body}\} \\ n_{rk} \in DS_f^R(n_r, V) \end{cases}$$

ここで，入力データ依存節点集合 $DD_{in}(C)$ は，部品外部の節点から部品内部の節点にデータ依存関係が存在するとき，依存元節点の集合を指す．また，出力データ依存節点集合 $DD_{out}(C)$ は，部品内部の節点から部品外部の節点にデータ依存関係が存在するとき，依存元節点の集合を指す．

出力パラメータにおいて，$A\,;B\,;C\,;D$ は A, B, C, D のうちどれか一つを選択することを意味する．V は部品作成時に指定した変数集合，V_d は部品本体内部で値が定義される変数集合，V_s は部品本体内部で値が一意に定まる変数集合，V_e は部品本体外部で参照される変数集合である．利用例の節点 n_r は，部品本体から外部への変数集合 V に関するデータ依存先となる節点の一つである．R は指定した変数集合 V が利用されている節点を見つけるために，節点 n_r から依存関係を順方向にたどる回数を表し，$DS_f^R(n_r, V)$ は節点 n_r から依存関係を順方向に R 回たどるとき通過する節点の集合を指す．

本稿では，実行条件例と利用例を合わせて付加情報 (additional description) と呼ぶ．これらは，部品本体と同時にスライシングによって生成される．定義 5 より，実行条件例は部品作成時に着目した変数に関する逆方向スライスから部品本体を取り除くことで生成する．よって，部品本体と組み合わせて得られるプログラムは逆方向スライスとなり，必ず実行可能かつ抽出等価である．また，利用例は部品作成時に着目した変数がもとのプログラムにおいて影響を与えていた文と，その文を実行するために必要なコードを集めることで生成する．ここで，順方向区間限定スライスは逆方向スライスの和集合なので，スライス単体で実行可能性と抽出等価性を満たす．よって，利用例と部品本体を組み合わせて得られるプログラムは必ず実行可能かつ抽出等価である．

3.3 実行条件例

部品を再利用してプログラムを作成する場合，その部品を実行させることで動作を確認したいという要求がある．一般に，部品を実行するためには入力変数に適切な値を設定する必要がある．しかし，実行に必要な情報が部品に附属しているとは限らない．そこで，部品を実行させたいという要求に応じるために，本手法では部品本体だけでなく実行条件例を提示することが可能である．

実行条件例により部品の動作確認が可能になると，検索で見つかった類似部品から適切なものを選択したり，部品を変更するのに有利である．部品が目的の機能を満たしているかどうかを調べる場合，もとのプログラムのテストデータを部品本体と実行条件例を合わせたプログラムに入力し，実行経路を確認することが考えられる．また，任意のデータをそのプログラムに入力し，部品本体の入口で実行をとめる．その時の変数の値を記録しておくことで，部品本体に対するテストデータが生成される．このように，部品のテストデータを変数の値でなくコードとすることで，部品を再利用する際に動的にテストデータを生成可能である．さらに，変更により新しく作成した部品に対して，変更前と同じ実行条件例を組み合わせて実行を試みることで，以前の機能に対しても正しく動作するのか，以前の機能に対して変更が反映されているのかを検査することができる．

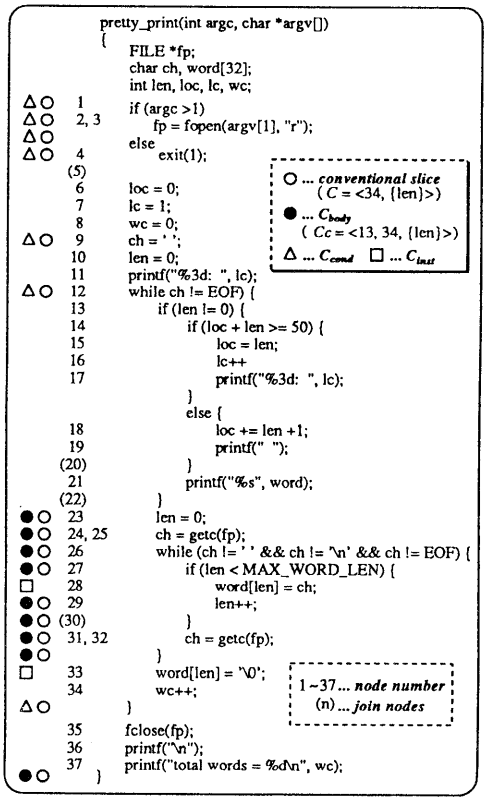

図 3: 部品作成対象プログラムと抽出した部品

3.4 利用例

部品を再利用して作成中のプログラムに組み込む場合，どのように利用可能であるのかを知りたいという要求がある．この要求に応じるために，本手法では利用例を提示することが可能である．

利用例は，部品がどの機能を担っていたのかを理解する指針となる．一般に，部品の機能を理解しようとする場合，部品内部のコードを解析することで出力を明確にする方法と，部品を利用している部品外部のコードから出力を推測する方法が考えられる．利用例は，後者の方法を支援する．また，利用例は部品の機能を理解するためだけでなく，実際に再利用する記述コードであり，部品を埋め込む際の結合コードとなる．よって，利用例まで含めて再利用対象として扱うことが可能である．特に，本手法によって作成した部品本体には定義変数集合が空集合になる関数 (printf など) が含まれないので，実際に利用する際には利用例を結合する機会は多いと考えられる．さらに，部品の利用履歴に基づき利用例の一部を部品本体に結合し，ソフトウェア部品を自動洗練することも可能である[14]．

3.5 ソフトウェア部品の例

図 3に部品作成対象プログラムと抽出した部品を示す．本手法では，代入文が関数を含む場合，関数呼び出し節点と代入文節点に分割する．また，区間として制御が合流する点を指定可能とするため，合流節点を導入する．さらに，標準ライブラリ関数 fopen, fclose, getch, printf に関しては，関数内部の節点を区間として指定できないことにし，関数の展開は行わない．

図 3のプログラムに対して，部品作成基準 C_C = <13, 34, {len}> を指定し，「単語の長さを数える」機能を持つソフトウェア部品を作成する．節点番号の左側の○は従来のスライシングにより抽出した部品，●は本手法により抽出した部品本体を表す．△は実行条件例，□は $R = 3$ として生成した 33 個の利用例の一つを表す．また，入力パラメータ C_{in}={fp}，出力パラメータ C_{out}=({len}, {len,ch}, {len,ch}, {len,ch}) である．

4 評価実験

本部品作成手法の有効性を確認するために，本手法に基づいてワークステーション上にソフトウェア部品作成実験システムを構築し，既存のプログラムからソフトウェア部品を作成する実験を行った．部品作成対象プログラムは，表 1に示す NetBSD のソースコードである．本手法で扱わない再帰構造と高次のポインタ型変数を含むコードは，前処理によって部品作成対象から除いた．

表 1: 実験対象プログラム (ソースコード)

プログラム	LOC	関数の数	節点数	抽出部品
whois	131	2	81	A1 ～ A4
finger	1240	24	799	B1 ～ B5
ftp	5595	128	4399	C1 ～ C6

表 2: 表 1 のプログラムから作成したソフトウェア部品

部品	部品作成基準	機能
A-1	⟨w20,w49,{connect}⟩	ソケットの結合
A-2	⟨w17,w29,{s}⟩	ソケットのオープン
A-3	⟨w30,w69,{sfi}⟩	データポインタの設定
A-4	⟨w20,w75,{ch}⟩	データの読み込み
B-1	⟨n9,n42,{connect}⟩	ソケットの結合
B-2	⟨n26,n36,{s}⟩	ソケットのオープン
B-3	⟨n9,n78,{fp}⟩	データポインタの設定
B-4	⟨n54,n76,{c}⟩	データの読み込み
B-5	⟨n35,n76,{c}⟩	データの読み込み
C-1	⟨f2,f30,{connect}⟩	ソケットの結合
C-2	⟨c37,f30,{connect}⟩	ソケットの結合
C-3	⟨f2,f66,{s}⟩	ソケットのオープン
C-4	⟨f22,f91,{cin}⟩	データポインタの設定
C-5	⟨m68,c49,{port}⟩	ポート番号の取得
C-6	⟨m68,f92,{hostname}⟩	ホスト名の取得

表 1のプログラムから作成した部品の部品作成基準と機能を表 2，部品の節点数及び入力パラメータ (入力引数) を表 3に示す．表 3において，対象関数とは実験対象プログラムを関数単位で分割したもの，従来部品とは対象関数に対して従来のスライシング手法を適用して抽出した部品を指す．さらに，従来部品に含まれる節点数 n に対して，C_{body} に含まれる節点数がどの位減少しているのかを表すために，節点数の割合 C_{body}/n を示す．

表 3: 表 2 のソフトウェア部品の節点数と入力パラメータ

部品	対象関数		従来部品		本手法によるソフトウェア部品					
	節点数	入力引数	節点数 n	入力引数	CRP	C_{body}	C_{body}/n	C_{cond}	C_{inst}	C_{in}
A-1	81	argv,argc	44	argv,argc	13	12	27%	32	1	host
A-2	81	argv,argc	27	argv,argc	5	4	15%	23	6	host,argc
A-3	81	argv,argc	49	argv,argc	19	10	20%	39	3	s,hp
A-4	81	argv,argc	57	argv,argc	30	19	33%	38	1	host
B-1	83	name	37	name	22	21	57%	16	1	host
B-2	83	name	23	name	7	2	9%	21	6	hp
B-3	83	name	42	name	53	23	55%	19	4	host
B-4	83	name	64	name	25	24	38%	40	1	s
B-5	83	name	64	name	35	27	42%	37	1	sin,hp
C-1	96	host,port	22	host,port	18	15	68%	7	1	host,port
C-2	185	argc,argv	40	argc,argv	27	24	60%	16	1	argc,argv
C-3	96	host,port	41	host,port	40	26	63%	15	3	host,port
C-4	96	host,port	50	host,port	30	17	34%	33	5	histaddr,hp
C-5	175	argc,argv	49	argc,argv	12	10	20%	39	2	argc,argv
C-6	271	argc,argv	103	argc,argv	59	28	27%	75	1	argc,argv

部品作成基準の節点 w : whois.c (whois), n : net.c (finger), m, f, c : main.c, ftp.c, cmds.c

5 考察

本章では，図3に示したソフトウェア部品の例と，4章の評価実験の結果について考察する．本手法に対する評価項目として以下の4点を設定した．

(1) 任意の区間を指定した際，抽出した部品本体から不要なコードが削除されるか (節点数の比較).

(2) 区間を変化させることで，機能の異なる部品が作成可能か (入力パラメータの比較).

(3) 異なるプログラムから抽出した同じ機能を持つ部品本体のコードは等しいか (機能とコードとの比較).

(4) 生成される付加情報は部品を再利用するのに役立つか (実行条件例と利用例のコード).

(1) 従来部品と部品本体の節点数

表3において，本手法による部品本体のすべての節点は従来部品に含まれ，部品本体の節点数 C_{body} は従来手法による部品の節点数 N より必ず小さくなる (節点数の割合 $C_{body}/N = 9\% \sim 68\%$). これは，部品の抽出対象を区間 CRP に限定しているためで，削除されたコードは着目した区間の外に存在していたコードか例外処理により部品外部に制御が移るコードであるためである．さらに，C-2,5,6 は関数をまたいで作成した部品であり，C_{body} の節点数は対象関数の節点数と比較してかなり少ない．これは，スライシングを部品作成に適用したことだけでなく，区間が指定可能であることによる効果である．以上より，評価項目 (1) に関して，次に示す効果が確認できる．

(a) 任意の区間を指定して部品本体を抽出することが可能で，指定した区間において着目した変数に影響を与えない記述コードが部品本体から従来より多く削除される．

(2) 指定する区間と入力パラメータ

表3より，従来部品の入力引数は対象関数の入力引数に支配されている．これに対して，本手法は入力パラメータ C_{in} をさまざまな変数に設定できる．これは，対象関数及び従来部品の入力引数と C_{in} を比較すれば明確である．さらに，本手法では着目する変数を同じものに指定しても，指定する上限節点により C_{in} を変化させることができる．例えば，B-4 と B-5 は着目する変数が節点 n76 の {c} で同じであるが，上限節点の位置 (n54 と n35) によって C_{in} は異なる．入力パラメータは部品の機能を決定する要素の一つであるとみなせるので，評価項目 (2) に関して，次に示す効果が確認できる．

(b) 入力パラメータとなる変数の選択肢が広がり，従来のスライシングでは抽出できない機能を持つ部品が作成可能である．

(3) 部品の機能と部品本体コード

本手法により抽出した部品本体の機能とコードとの関係を調べるために，同じ機能を持つように部品作成基準を設定して部品本体を抽出した．表3の A-1 と B-1 のように，部品作成基準を調節することで同じ機能の部品を抽出しても，2つの部品本体の C_{body} 及びコードは大きく異なることがある．これは，部品作成対象プログラムの指定した区間において，プログラム構造が大きく違っていたためである．逆に，A-1 と C-1 では C_{in} が異なりはするが，C_{body} はほぼ等しく，そのコードはほとんど同じであった．これは，指定した区間でプログラム構造が似ていたためである．以上より，評価項目 (3) に関して，次に示す効果が確認できる．

(c) 部品作成対象のプログラム全体で構造が大きく異なっていても，指定した区間においてプログラム構造が似ている場合，抽出された同じ機能を持つ部品本体のコードは類似する．

(4) 実行条件例と利用例

まず，実行条件例について考察する．図3において，C_{body} と C_{cond} を組み合わせたプログラムを C_{prog} とおく．C_{prog} はファイル名を引数として指定可能な「単語

の長さを数える」機能を持つプログラムであり，実際に実行して動作確認ができる．また，引数となるファイルを与えて C_{prog} を実行することで，部品本体のテストデータとして変数 fp の値が取得できる．さらに，部品本体の while(節点 26) 文の条件に ch != '\t' を新しく追加し，実行条件例と組み合わせて実行を試みることで，部品内部の変更が部品外部にどのような影響を与えているかを調べることができる．表 3に，従来部品を抽出した際の対象関数に含まれる実行条件例の節点数 C_{cond} を示す．表 3において，C_{cond} と C_{body} を加算したものが n となることが確かめられる．また，A-1 と C-1 のように同じ機能でコードも類似している部品本体に対して，実行条件例の節点数が大きく異なることがある．これは，機能が同じでも部品本体は独自のテストパターンをそれぞれ持つことを指す．

次に，利用例について考察する．図 3のソフトウェア部品において，変数 len の担う機能を知らずに，部品作成基準変数に len を指定したとする．C_{inst} より，部品本体の出力変数 len は配列型変数 word における位置を指していて，word[len] に一文字変数 ch の値と終端を表す定数'\0' を代入していることが確認できる．このことより，変数 len は変数 word の配列の添字に利用可能であることや，変数 len は変数 word の文字列の長さを表していることを推測する手がかりとなる．表 3に R=1 で生成した利用例の節点数 C_{inst} を示す．R=1 では，節点数が利用パターンと等しくなる．

以上より，評価項目 (4) に関して，次に示す効果が確認できる．

(d) 実行条件例により部品は必ず実行可能となり，多様で柔軟なテストが適用できる．また，利用例は部品内部の変数の機能を推測するのに役立つ．

6 おわりに

区間限定スライスを用いて，既存のプログラムから再利用部品を作成する手法に基づき構築した再利用システムを用いて評価実験を行い，その実験結果と本手法の利点に関する考察を述べた．本手法は，抽出した部品が実行可能性と抽出等価性という部品に不可欠な性質を満たすことを保証しながら，部品作成対象プログラムに対して任意の区間を指定して，部品抽出が可能であるという点で従来の手法より有利である．また，本手法では部品理解を支援するために，実行条件例と利用例を部品利用者に提示する．これらの実行条件例と利用例をそれぞれ部品本体と組み合わせて得られるプログラムは実行可能性と抽出等価性を満たすので，部品の実行動作や利用方法の確認が容易である．

本手法では部品作成基準の区間と変数は部品作成者が指定しなければならず，この基準は作成される部品の有用性に大きく影響を及ぼす．このため，部品作成者と部品作成システムとの対話を頻繁に行ない，過去に行なった部品の抽出及び変更履歴を蓄積し，部品作成基準の指定に対する指針として提示することを検討している．

謝辞 日頃御指導御討論いただく後藤滋樹部長，伊藤正樹リーダーはじめ，グループの皆様に深く感謝します．

参考文献

1) Abd-El-Hafiz,S.K., Basili,V.R. and Caldiera,G.: *Towards Automated Support for Extraction of Reusable Components*, Proc. Conf. Softw. Maintenance, pp.212-219 (Oct. 1991)

2) Dunn,M.F. and Knight,J.C.: *Automating the Detection of Reusable Parts in Existing Software*, IEEE 15th Int. Conf. Softw. Eng., pp.381-390 (May 1993)

3) Weiser,M.: *Program Slicing*, IEEE Trans. Softw. Eng., Vol.10, No.4, pp.352-357 (Jul. 1984)

4) 丸山勝久, 高橋直久: プログラムスライシングによるソフトウェア部品の作成, 情報処理学会ソフトウェア工学研究会 (94-SE-98), Vol.94, No.43, pp.1-8 (May 1994)

5) Aho,A.V., Sethi,R. and Ullman,J.D.: *Compilers : Principles, Techniques, and Tools*, Addison-Wesley (1986)

6) Ferrante,J., Ottenstein,K.J. and Warren,J.D.: *The Program Dependence Graph and Its Use in Optimization*, ACM Trans. Prog. Lang. Syst., Vol.9, No.3, pp.319-349 (Jul. 1987)

7) Ottenstein,K.J. and Ottenstein,L.M.: *The Program Dependence Graph in a Software Development Environment*, Proc. ACM SIGPLAN / SIGSOFT Symp. Practical Programming Development Environments, ACM SIGPLAN Notices, Vol.19, No.5, pp.177-184 (May. 1984)

8) Gallagher,K.B. and Lyle,J.R.: *Using Program Slicing in Software Maintenance*, IEEE Trans. Softw. Eng., Vol.17, No.8, pp.751-761 (Aug. 1991)

9) Cutillo,F., Fiore,P. and Visaggio,G.: *Identification and Extraction of "Domain Independent" Components in Large Programs*, Proc. Working Conf. Reverse Engineering, pp.83-92 (May 1993)

10) Beck,J. and Eichmann,D.: *Program and Interface Slicing for Reverse Engineering*, IEEE 15th Int. Conf. Softw. Eng., pp.509-518 (May 1993)

11) Canfora,G., Climitile,A., De Lucia,A. and Di Lucca, G.A.: *Software Salvaging Based on Conditions*, Proc. Conf. Softw. Maintenance, pp.424-433 (Sep. 1994)

12) Ball,T. and Horwitz,S.: *Slicing Programs with Arbitrary Control Flow*, Proc. Int. Workshop on Automated and Algorithmic Debugging, Lecture Notes in Computer Science, Vol.749, Springer-Verlag, pp.206-222 (May 1993)

13) Horwitz,S., Reps,T. and Binkley,D.: *Interprocedural Slicing Using Dependence Graphs*, ACM Trans. Prog. Lang. Syst., Vol.12, No.1, pp26-60, (Jan. 1990)

14) 丸山勝久, 島健一, 高橋直久: 重み付き依存グラフを用いたソフトウェア部品の洗練, 情報処理学会第 49 回全国大会, 5M-8 (Sep. 1994)