プログラム理解のためのCプログラムのモジュール化手法とその視覚化

栗田　健士[†]　　大久保　弘崇[††]　　粕谷　英人[††]　　山本　晋一郎[††]　　齋藤　邦彦[†††]

[†]愛知県立大学大学院 情報科学研究科　　[††]愛知県立大学 情報科学部

[†††]滋賀大学 経済学部

Modularization of C Programs for Program Understanding and Visualization

Takeshi KURITA[†]　　Hirotaka OHKUBO[††]　　Hideto KASUYA[††]

Sinichiro YAMAMOTO[††]　　Kunihiko SAITO[†††]

[†]Graduate School of Information Science and Technology, Aichi Prefectural University

[††]Faculty of Information Science and Technology, Aichi Prefectural University

[†††] Faculty of Economics, Shiga University

1　はじめに

ソフトウェアの再利用や保守，検証プロセスでソフトウェア理解支援の重要性が高まっている．開発者のソフトウェア理解を支援する様々なツールが開発されている．その中でソフトウェアの構造を視覚的に表現し，開発者の直感的なソフトウェア理解を支援することで，効率的なソフトウェアの保守と運用が可能となる．

また，ソフトウェアの大規模化，複雑化にともない，開発者がソフトウェアを理解することはますます困難となる．そのため，開発者が直観的に理解できるツールが要求されている．ソフトウェアは多数の部品から構成されるが，ソフトウェア部品を単独で見ても理解できることは限られる．そこで，意味的にソフトウェア部品を再分類したモジュール単位で提供できれば，大規模で複雑なソフトウェア全体の理解に有効であると考えられる．モジュール識別の手法として，概念分析によるプログラム分割手法が Siff ら [2] により提案されている．概念分析手法は，対象とする集合から関連のある要素の集合を概念として抽出し，概念束による順序化や概念の組み合わせによるモジュール分割を行う．Snelting ら [3] は対象プログラムを C 言語とし，集約関係を用いた概念分析により，C プログラムの C++プログラムへのリファクタリングを試みている．

本研究は，同手法を比較的大規模なソフトウェア理解に応用する．関数の集合をモジュール単位とし，集約関係を用いてプログラムを分割する．また，関数呼出グラフと分割を組み合わせることで，ソフトウェア理解に役立つ分割の視覚化を行う．文献 [2] の概念分析では中規模以上のソフトウェアから膨大な分割が生成される．プログラム理解にとって有用な分割を獲得するために，モジュールの集合を分割の抽象度に基づき順序化する．また分割された階層グラフ，複合グラフ [4] を用いて複数のプログラム分割をひとつのグラフとして表現する．C プログラムを対象とし，集約関係を用いた分割，階層化，視覚化を行う処理系を実装を提案する．bison-1.5，gzip-1.3.3，bash など 7 種類のソフトウェアを対象とし概念生成，分割，視覚化を行い，性能の評価を行う．

2 章では，モジュール化手法を例を挙げて説明する．3 章では，モジュール化手法を C プログラムに適用する．そして，視覚化の考察と本研究の実装方法を述べる．4 章では，中規模以上のプログラムを対象に本手法の実験，評価を行う．最後にまとめと今後の課題を述べる．

2　モジュール化手法

本論文ではプログラムのモジュール化手法として文献 [2] で提案された概念分析・分割によるモジュール識別手法を用いる．2.1 章で概念分析の基本的な概念を，2.2 章で概念分割の手法を述べる．

2.1　概念分析

概念分析は，対象とする集合から同じ性質を共有するモジュールを選りだす手法である．集合の要素をオブジェクト，その特徴や性質を属性 (attribute) とし，モジュールを識別する．概念分析は数学的理論のひとつとして 1940 年に Birkhoff により基礎づけられた [1]．Birkhoff はオブジェクトと属性の二項関係から，内在する関係を束構造として構成した．現在，概念分析をソフトウェア工学に応用する研究が数多く提案されている [2][3]．

表 1: 哺乳類を対象にしたコンテキスト

		オブジェクト					
		猫	犬	イルカ	猿	ヒト	鯨
属性	四つ足である	√	√				
	毛で覆われている	√	√		√		
	知的である			√	√	√	√
	水中で生活する			√			√
	親指がある				√	√	

2.1.1 定義

コンテキスト $\mathcal{C}$ は概念分析の基底である．コンテキスト $\mathcal{C}(\mathcal{C} \subseteq \mathcal{O} \times \mathcal{A})$ はオブジェクトの集合 $\mathcal{O}$ と属性の集合 $\mathcal{A}$ の間の二項関係を定義する．あるオブジェクトの集合 $O \subseteq \mathcal{O}$ が共通に持つ属性の集合を式 (1) で定義する．

$$\sigma(O) = \{a \in \mathcal{A} | \forall o \in O : (o, a) \in \mathcal{C}\} \tag{1}$$

ある属性の集合 $A \subseteq \mathcal{A}$ に関して，その全てを持つようなオブジェクトの集合を以下の式 (2) で定義する．

$$\tau(A) = \{o \in \mathcal{O} | \forall a \in A : (o, a) \in \mathcal{C}\} \tag{2}$$

$A = \sigma(O)$ かつ $O = \tau(A)$ であるとき，(O, A) の組を概念と呼ぶ．概念 $c = (O, A)$ の**外延**を $ext(c) = O$，**内包**を $int(c) = A$ とする．最初に概念式 (3)，(4) を満たす概念が生成する．

$$(\tau(\sigma(O)), \sigma(O)) \tag{3}$$

$$(\tau(\sigma(\emptyset)), \sigma(\emptyset)) \tag{4}$$

オブジェクト O の一つの集合が与えられたとき，式 (3) を満たす概念は基本概念と呼ぶ．基本概念はこれ以上単純な概念に分けることが出来ない概念である．式 (4) を満たす概念はすべての属性を持つオブジェクトから成る．そして，他の概念は式 (5) の順序関係で新たに形成される．

$$(O_1, A_1) \leq (O_2, A_2) \Longleftrightarrow O_1 \subseteq O_2 \Longleftrightarrow A_1 \supseteq A_2 \tag{5}$$

概念束は概念の集まりから構成される．概念束 $\mathcal{L}(\mathcal{C})$ の 2 つの要素 $(O_1, A_1), (O_2, A_2)$ は下限と上限が定義される．

$$(O_1, A_1) \wedge (O_2, A_2) = (O_1 \cap O_2, \sigma(O_1 \cap O_2)) \text{ (下限)} \tag{6}$$

$$(O_1, A_1) \vee (O_2, A_2) = (\tau(A_1 \cap A_2), A_1 \cap A_2) \text{ (上限)} \tag{7}$$

2.1.2 概念分析の例

文献 [2] に基づいて，哺乳類を対象とする概念分析の例を示す．オブジェクトを “犬”，“猫”，“イルカ”，“猿”，“ヒト”，“鯨” ，属性を “四つ足である”，“毛で覆われている”，“知的である”，“水中で生活する”，“親指がある” とする．表 1 はこの例のコンテキスト $\mathcal{C}$ を表す．このコンテキストから概念束を形成するのは，基本概念の集合から，式 (5) の順序関係を満たすように下限の概念から上限の概念まで形成する．形成された概念のリスト (表 2) と概念束 (図 1) を以下に示す．

2.2 概念分割

概念分割は与えられた概念束から分割を作成する手法である．分割は概念の外延がそれぞれ重複なくすべてのオブジェクトを含む概念集合である．

表 2: 哺乳類を対象にした概念

概念	オブジェクト	属性
c_0	$\emptyset$	{ 毛で覆われている, 四つ足である, 知的である, 水中で生活する, 親指がある }
c_1	{ イルカ, 鯨 }	{ 水中で生活する, 知的である }
c_2	{ 猫, 犬 }	{ 毛で覆われている, 四つ足である }
c_3	{ 猿, ヒト }	{ 知的である, 親指がある }
c_4	{ 猿 }	{ 毛で覆われている, 親指がある, 知的である }
c_5	{ 猿, 猫, 犬 }	{ 毛で覆われている }
c_6	{ イルカ, 猿, 鯨, ヒト }	{ 知的である }
c_7	{ イルカ, 猿, 鯨 ヒト, 猫, 犬 }	$\emptyset$

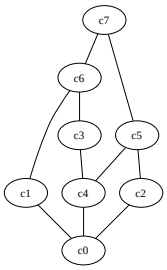

図 1: 哺乳類を対象にした概念束

2.2.1 定義

コンテキスト $\mathcal{C}(\mathcal{C} \subseteq \mathcal{O} \times \mathcal{A})$ に関して，ある概念の集合 $P = \{(O_0, A_0), \ldots, (O_{k-1}, A_{k-1})\}$ が概念分割であるとは，$\bigcup O_i = \mathcal{O}$ として $O_i \cap O_j = \emptyset, i \neq j, O_i, O_j \in P$ であるという．

基本概念で構成する分割は基本分割と呼び，概念 $(\tau(\emptyset), \emptyset))$ で構成する分割は自明の分割と呼ぶ．基本分割の中に $\bigcup O_i = \mathcal{O}$ を満たさない概念が存在する．そのとき，**適格でない**コンテキストであるという．適格なモジュールを提供するためにコンテキストは適格でなければならない．すべての $x, y \in \mathcal{O}$ について $\sigma(\{x\}) \subseteq \sigma(\{y\})$ ならば $\sigma(\{x\}) = \sigma(\{y\})$ であるとき，コンテキストは適格である．適格なコンテキストでないときは，コンテキストに属性を加える．$\sigma(\{x\}) \subsetneq \sigma(\{y\})$ の場合，x は y の属性が少なくとも一つ存在する．そのとき $a \notin \sigma(\{x\})$，$a \in \sigma(\{y\})$ を満たす $a \in \mathcal{A}$ の属性の否定 $\bar{a}$ をコンテキストに追加する．

適格なコンテキストから生成した概念束が与えられたとき，半順序関係において概念 d が概念 c のひとつ上位の概念であるならば，$d = covs(c)$ である．また，概念 d より下位の概念の集合が存在するとき，下位の概念の集合は $subs(d)$ である．図 2 により概念束からすべてのパーティションの集合体を構築する．

2.2.2 概念分割の例

表 1 の例のコンテキストを考える．基本概念 c_1，c_2，c_3，c_4 の外延が重複しているため，適格でないコンテキストである．$\sigma(\{\text{ ヒト }\}) \subsetneq \sigma(\{\text{ 猿 }\})$ かつ $a \notin \sigma(\{\text{ ヒト }\}), a \in \sigma(\{\text{ 猿 }\})$ を満たす属性 a は “毛で覆われている” である．“毛で覆われている” の否定の属性 “毛で覆われていない” をコンテキストに追加

$A \leftarrow covs(\bot)$
$P \leftarrow \{A\}$
$W \leftarrow \{A\}$
while $W \neq \emptyset$ **do**
  W からある p を取り除く
  for all $c \in p$ **do**
    for all $c' \in covs(c)$ **do**
      $p \leftarrow subs(c')$
      if $(\bigcup p') \bigcap c'$ **then**
        $p'' \leftarrow p \cup \{c'\}$
        if $p'' \notin P$ **then**
          $P \leftarrow P \cup \{p''\}$
          $W \leftarrow W \cup \{p''\}$
        end if
      end if
    end for
  end for
end while

図 2: 概念分割生成アルゴリズム

する．追加されたコンテキストから生成した概念，概念束はそれぞれ表 3，図 3 である．

表 3: 追加されたコンテキストから生成した概念

概念	オブジェクト	属性
c_0	$\emptyset$	{ 毛で覆われている, 四つ足である 知的である, 毛で覆われていない, 水中で生活する, 親指がある }
c_1	{ 猫, 犬 }	{ 毛で覆われている, 四つ足である }
c_2	{ 猿 }	{ 毛で覆われている, 親指がある 知的である }
c_3	{ イルカ, 鯨 }	{ 毛で覆われていない, 水中で生活する, 知的である }
c_4	{ ヒト }	{ 毛で覆われていない, 知的である, 親指がある }
c_5	{ イルカ, 鯨, ヒト }	{ 毛で覆われていない, 知的である }
c_6	{ イルカ, 猿, 鯨, ヒト }	{ 知的である }
c_7	{ 猿, ヒト }	{ 知的である, 親指がある }
c_8	{ 猿, 猫, 犬 }	{ 毛で覆われている }
c_9	{ イルカ, 猿, 鯨, ヒト, 猫, 犬 }	$\emptyset$

図 3 で表された概念束から生成した概念分割は表 4 である．

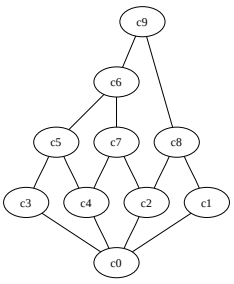

図 3: 表 3 で表される概念束

表 4: 哺乳類を対象にした概念分割

概念分割	概念
P_0	$\{c_1,c_2,c_3,c_4\}$
P_1	$\{c_2,c_4,c_7\}$
P_2	$\{c_1,c_2,c_8\}$
P_3	$\{c_1,c_3,c_6\}$
P_4	$\{c_2,c_5\}$
P_5	$\{c_6,c_7\}$
P_6	$\{c_9\}$

3　プログラムの分割と視覚化

2 章で紹介したモジュール化手法を，C プログラムに適用する．構文解析によりオブジェクトと属性を抽出しコンテキスト，概念，分割を生成する．また取得したモジュールを，関数呼び出しグラフと組み合わせて視覚化する．各処理系は Java，perl プログラムとして実装した．構文解析部の実装には CASE ツールプラットフォーム Sapid[6] を用いた．

3.1　コンテキストの生成

対象プログラムの内部構造を表現するようにオブジェクト，属性を定め，コンテキストを構成する．Sapid[6] を用いることで，ライブラリ，ファイル，関数 (定義文)，ブロックや式といったプログラムの単位をオブジェクトとして簡単に取得できる．ここでは C プログラム中の関数をオブジェクトとして選択する．属性は，関数定義文中の構造体型変数 (パラメータ，ローカル変数) の利用とする．

プログラム利用例として図 4 のスタックとキューの C プログラムを用いる．図 4 の C プログラムは構造体 stack と queue を持つ．オブジェクトを関数名，属性を関数内の構造体変数の返り値，引数，フィールドの利用とし，表 5 のコンテキストを作成した．次にコンテキストから表 6 で表される概念，図 5 の

```
1  #define QUEUE_SIZE 10
2  struct stack { int *base, *sp, size };
3  struct queue { struct stack *front, *back; };
4
5  struct stack* initStack(int sz)
6  { struct stack* s =
7    (struct stack *) malloc(sizeof(struct stack));
8    s->base = s-> =
9    (int*)malloc(sz * (sizeof(int)));
10   s->size = sz;
11   return s; }
12
13 struct queue* initQ()
14 { struct queue* q =
15   (struct queue*) malloc(sizeof(struct stack));
16   q->front = initStack(QUEUE_SIZE);
17   q->back  = initStack(QUEUE_SIZE);
18   return q; }
19 (以下 push, enq, pop, deq の定義)
```

図 4: スタックとキューの C プログラム

概念束を生成する．図 5 の概念束から表 7 で表される概念分割を生成する．

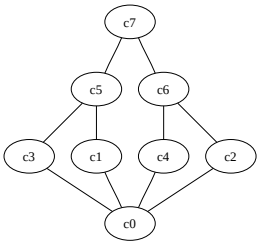

図 5: 表 6 で表される概念束

表 7: 図 4 の概念分割

分割	概念
P1	$\{c_1, c_2, c_3, c_4\}$
P2	$\{c_1, c_3, c_6\}$
P3	$\{c_2, c_4, c_5\}$
P4	$\{c_5, c_6\}$
P5	$\{c_7\}$

表 5: スタックとキューのコンテキスト

	return stack	return queue	has stack arg.	has queue arg.	uses stack fields	uses queue fields
initStack	√				√	
initQ		√				√
isEmptyS			√		√	
isEmptyQ				√		√
push			√		√	
enq				√		√
pop			√		√	
deq				√		√

表 6: スタックとキューの概念

概念	オブジェクト	属性
c_0	$\emptyset$	allattributes
c_1	{initStack}	{return stack, uses stack fields}
c_2	{initQ}	{return queue, uses queue fields}
c_3	{isEmptyS, pop,push}	{uses stack}
c_4	{isEmptyQ, enq,deq}	{uses queue}
c_5	{initStack,isEmptyS, pop,push}	{uses stack}
c_6	{initQ, isEmptyQ, enq,deq}	{use queue}
c_7	allobjects	$\emptyset$

3.2 モジュール分割の階層化

生成された多数の分割からプログラム理解に有用な分割だけを抽出するために分割の順序付けを行う．分割の集合は各分割が含む概念の属性により半順序関係を形成する．この関係は，複雑な概念と単純な概念といった抽象関係を表す．概念分割 P におけるそれぞれの分割 P_i は，ひとつまたはそれ以上の概念の集合となる．基本分割は基本概念から構成された分割であり，最も詳細な分割である．また，ひとつの概念からなる分割は自明な分割と呼ばれ，すべてのオブジェクトをひとつのモジュールとして扱う．すべての分割は基本分割と自明な分割の間を上限と下限とする束構造を構成する．各分割の階層の関係を表現する半順序関係を以下の通り定義する．

定義　ある概念分割 P_A,P_B の上下関係 $\prec$ で $P_A \prec P_B$ と表されるのは，分割 P_B のある概念 c_i に対し，分割 P_A の概念 c_{i_1} と c_{i_2} が下位の概念にあたり，$c_i = c_{i_1} \sqcap c_{i_2}$，$c_{i_1}, c_{i_2} \in P_B$，のときである，かつ $C_A - \{c_j\} \equiv C_B - \{c_{i_1}, c_{i_2}\}$ のときである．

この関係を使って基本分割から自明の分割を結ぶ任意のシーケンス P_{s_i} を概念分割から抽出する．P_{s_0} は基本分割であり，P_{s_n} は自明の分割である．

$$P_{s_0} \prec P_{s_1} \prec P_{s_2} \prec \cdots \prec P_{s_n}$$

3.3 視覚化

生成されたプログラムの概念分割を，ソフトウェア理解支援に役立つように視覚化する．ある分割の各概念が保持するオブジェクトの集合をプログラムモジュールとする．関数呼び出し関係等を用いて，分割の各概念 (プログラムモジュール) の相互関係を表現する有向グラフを作成する，また，抽象度の異なる分割のモジュールを関連付け，複数の視点からプログラムの構造を表現するため，有向グラフの階層描画法による階層化グラフ，複合グラフ [4] を用いる．

3.3.1 関数呼び出しに基づく階層分割グラフ

C プログラムの関数は，関数呼び出しで他の関数と関係を持つ．分割グラフ G_1 はこの呼び出し関係を用いて，概念をプログラムモジュールとして表現し，結合する有向グラフを形成する．各概念のオブジェクト要素である関数が他の概念に含まれる関数と呼び出し関係を持つとき，2 つの概念の間に有向エッジを引く．複数の関係があるときはひとつのエッジで表現する．同様な手順で，別の分割に基づく分割グラフ G_2 を作成する．分割グラフ G_1 と分割グラフ G_2 間にエッジを引き，階層グラフを形成する (図 6)．上位分割 G_2 の概念は，下位分割 G_1 の複数の概念が対応する．また，関数呼び出しグラフを下層とする階層分割グラフを形成できる．このときエッジは概念を表すノードとその概念が含む関数の間に引かれる．

3.2 で導入した分割のシーケンス P_{s_i} を用いて，多層の階層分割グラフが形成される．このとき，基本分割 P_{s_0} の下位に関数呼び出しグラフが置かれる．

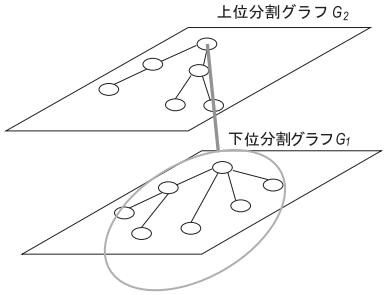

図 6: 階層化グラフ

3.3.2 属性による複合グラフ

概念の属性に注目し，属性を領域，概念をノードとする複合グラフを用いて属性によるプログラム分割図を作成する (図 7)．ある属性に注目し，それに対応する概念を関連付けるとき，いくつかの概念は他の属性と共有されるため，属性を領域で表現し，概念をノードで表現することによって，ネスト可能な複合グラフを用いて，属性同士の包含関係と，属性と概念の連結関係を表現する．この複合グラフは属性の性質に基づくプログラム分割を表現する．

3.4 実装

本手法の実装を図 8 に示す．図のように，本手法は 5 つの要素から構成する．

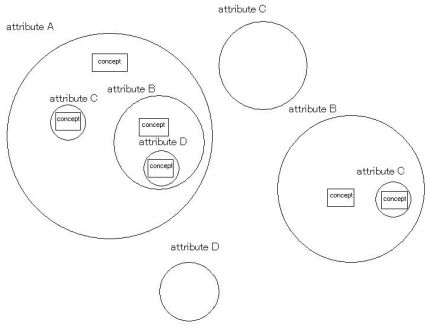

図 7: 複合グラフ

- Sapid により，対象のソースプログラムの構文解析を行い，解析情報をソフトウェアデータベース (SDB) に格納する．
- コンテキスト生成器はソフトウェアデータベースを入力としてオブジェクトとなる情報と，属性となる情報を取り出しコンテキストを出力する．
- 概念分析生成器はコンテキストを入力として概念，概念束を出力する．
- 概念束から概念分割を生成する．コンテキストが適格でないならば，適格なコンテキストになるまで属性を追加する．
- 概念分割を組み合わせて関数呼び出し図を元とする視覚化に利用する．

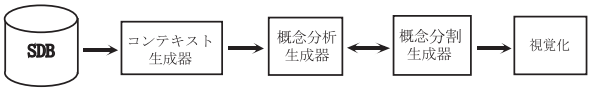

図 8: 概念分析ツール

4 実験と評価

3 章で述べた，概念分析を用いた C プログラムのモジュール識別，分割，視覚化実験を中規模以上のソフトウェアを対象として行い，性能の評価を行った．視覚化実験では，分割を用いて，プログラム分割図を階層グラフ，複合グラフとして作成した．生成された各グラフとハイパードキュメント SPIE を連携させ，ソフトウェア理解ためのナビゲーション機能を例示した．

4.1 実験結果

関数をオブジェクト，関数定義文における構造体型変数の利用を属性としてコンテキスト作成を行い，概念生成，概念分割の実験を行った (表 8)．bison-1.5，fileutils-ls-3.14，gzip-1.3.3 といった 1 MLOC 程度のソフトウェアで概念と分割が生成された．最大規模 (10MLOC) の bash では，メモリ不足のため，分割が生成できなかった．そのため，属性 として利用する構造体を BASH INPUT，COMMAND，Keymap，SHELL VAR の 4 つに制限し，概念，分割を生成した (表 9)．

表 8: 概念分析，概念分割による各ソフトウェアの実験結果

	LOC	オブジェクト	属性	概念	分割	基本概念
whetstone	37.6	6	1	4	2	2
dhrystone-2.1-bin	39	7	2	7	4	4
gnugo-1.2	98.6	30	1	4	2	2
gzip-1.3.3	38.3	74	10	53	255	11
bison-1.5	986.0	162	10	203	68572	20
fileutils-ls-3.14	1870.2	115	15	52	778	-
bash	9550.0	1078	83	60	-	-

表 9: 属性を制限した bash の実験結果

	LOC	オブジェクト	属性	概念	分割	基本概念
bash	9550.0	1078	4	4	6	4

4.1.1 プログラム分割の視覚化例

bison-1.5 の基本分割を例として，プログラム分割を視覚化するグラフを作成する．bison-1.5 の属性は 10 個，基本概念の数は 20 個，定義関数の数は 162 個である．各基本概念に含まれる関数の数を表 10 に，各属性と概念の関係を表 11 に表す．

表 10: 各基本概念に含まれる関数の数

c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9
32	2	5	7	2	1	10	3	4	1
c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}	c_{16}	c_{17}	c_{18}	c_{19}
1	1	10	2	48	1	19	1	1	9

概念分割グラフと関数呼び出しグラフを階層的に表現した階層分割グラフを作成する (図 9,10)．現行のバージョンは 3 次元表示に対応しないため，階層分割グラフ中のノードを関数群に展開した図で代替する．図 10 は概念 c_3 を展開した分割グラフである．楕円ノードは bison-1.5 で定義された関数を表す．このノードと対応する関数ドキュメントとソースコードの間にリンクを形成する．関数ドキュメントは Sapid で開発されたハイパードキュメント生成系 SPIE[5] を用いる．

属性による複合分割グラフを作成する．属性を領域で表現し，その内部に別の属性と概念が包含される．領域は円を，概念ノードは長方形として表示する．表 11 から，包含する概念の大きい順に 4 つ属性を選び，複合分割グラフを作成する (図 11)．このグラフは，属性で表現されるプログラムの特定の性質に注目した分割を支援する．複合分割グラフの各ノードと概念分割グラフを結び，階層グラフを形成することで，ソフトウェア理解支援のためのナビゲーション機能が実現できる．例えば，特定の構造体に注目しながら関連する関数群を検索し，そのドキュメントを参照するといったことが可能となる．

4.2 評価

プログラムが適切な数のモジュールに分割され，ソフトウェア理解支援に役立つ概念分割グラフが提供されるかといった観点から本研究の評価を行う．

プログラムの分割候補は分割の数だけ存在するが，分割グラフの最大ノード数は基本概念数により決まる．プログラム分割図の視認性を考慮すると，基本概念数は概ね 25 個以下であることが

表 11: 属性と概念の関係

属性	概念
FILE	c_2 c_3 c_8 c_{14} c_{15} c_{17} c_{18}
shifts	c_6 c_7 c_{10} c_{11} c_{17} c_{18}
reductions	c_1 c_9 c_{10} c_{11} c_{17}
core	c_7 c_9 c_{15} c_{16}
errs	c_{10} c_{13} c_{17} c_{18}
bucket	c_2 c_3 c_{12}
symbol list	c_2 c_5
shorts	c_4 c_{11}
percent table struct	c_8
option	c_{19}

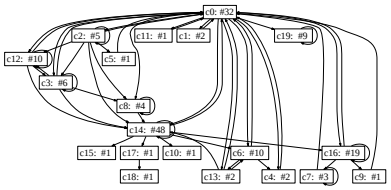

図 9: bison-1.5 の分割

望ましい．表 8 により，gzip-1.3.3，bison-1.5，fileutils-ls-3.14 の基本概念数は適切なサイズである．whetstone，gnugo-1.2 など，プログラムサイズが小さく，概念や分割数が少なくなるソフトウェアは，グローバル変数の出現といった新しい属性を用いて再分割を行う必要がある．

次に，基本概念による概念分割グラフや階層・複合グラフのソフトウェア理解支援のためのナビゲーション機能を評価する．関数呼び出しグラフ自身のナビゲーション機能を考慮すると，gzip-1.3.3，bison-1.5 といった，中規模以上のソフトウェアの関数呼び出しグラフはノードの数が多く，ドキュメントやソースコードといった詳細な情報に導くナビゲーション図としては不適切である．一方，各ソフトウェアの分割グラフは，ノードが 10 個から 20 個程度であり，ナビゲーション機能は有効である．ソフトウェアの規模が大きい場合は，階層分割グラフの階層数を増やすことでナビゲーション機能を活用するアプリケーションが可能である．

5 おわりに

5.1 まとめ

本研究では，プログラム理解のために，特定のモジュール単位でプログラムを分割し，関数呼び出しグラフと分割を組み合わせて視覚化を行った．

bison-1.5，gzip-1.3.3，bash といった 7 種類のソフトウェアを対象として，概念生成，分割，視覚化を行い，性能の評価を行った．プログラムサイズが概ね 1MLOC 程度のソフトウェアで概念と分割が生成された．10MLOC の bash は属性の構造体を制限して，概念，分割を生成した．

5.2 今後の課題

今後の課題として属性の変更に応じた対話的なツールの実装を行う必要がある．本研究では分割の視覚化に階層グラフ，複合

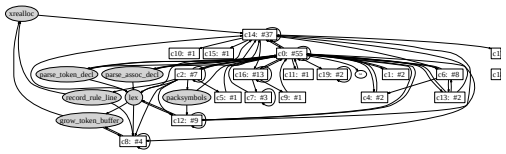

図 10: 概念 c3 の分割グラフ

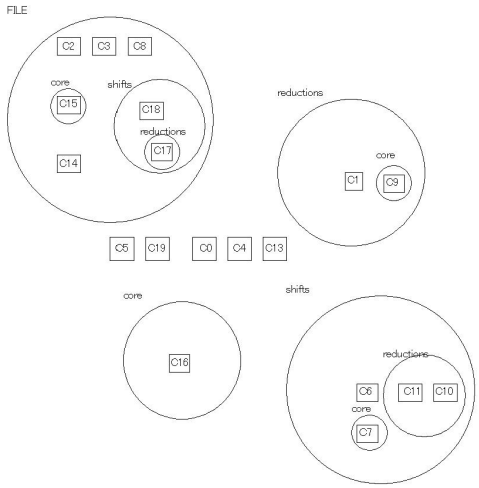

図 11: 複合分割グラフ

グラフを用いたが，他の種類のグラフも取り入れ，様々な見方からソフトウェア理解に役立てる．また，特に構造体型変数の利用を属性として選択したが，他の種類の要素を選択範囲に増やす．また SPIE の連携を行えるアプリケーションを作成する．

参考文献

[1] Garrett Birkhoff, “Lattice Theory”, American Mathematical Society. 1940.

[2] Michael Siff and Thomas Reps, “Identifying Modules via Concept Analysis”, IEEE Transactions on Software Engineering, Vol. 25, pp. 749–768, Nov-Dec 1999.

[3] Gregor Snelting and Frank Tip, “Reengineering Class Hierarchies Using Concept Analysis”, Foundations of Software Engineering (FSE-6), SIGSOFT Software Engineering Notes 23(6), pp. 99–110, 1998.

[4] 三末 和男, 杉山 公造, “図的思考支援を目的とした複合グラフの階層的描画法について”, 情報処理学会論文誌, Vol. 30, No. 10, pp. 1324–1334 1989.

[5] SPIE. `http://www.sapid.org/html2/mkSpec/SPIE-0.html`

[6] Sapid. `http://www.sapid.org/`

[7] Rigi. `http://www.rigi.cs.uvic.ca/`