形式的概念分析によるグルーピングのCプログラム理解支援への適用

齋藤邦彦
栗田健士†

Abstract

Formal Concept Analysis is a technique to identify possible groups sharing common properties of the target objects. With Formal Concept Analysis, we present a grouping method for C program functions by “use of struct types” and apply the method to the middle-size open-source software, such as Gzip and Bison. As an example, by combining function call graphs and groupings, we present program overviews. Program overviews are used as a map for program exploration in software understanding tools. We develop a generator of the program overviews from C programs and demonstrate that our method can be efficiently applied to software understanding.

Key words Formal Concept Analysis, Software Understanding, Grouping
キーワード　形式的概念分析，グルーピング，ソフトウェア理解

1．はじめに

大規模ソフトウェアの品質の維持，向上をはかるため，開発者はソースプログラムを読み，プログラム全体の構造や動作を理解する。ソフトウェア理解支援ツールはドキュメントやプログラム構成図の提供を通じて，プログラム探索やソースコード解読作業を支援する。Rigi[9]，SHriMP[10]はグラフ構造を用いてプログラムの概要情報を表示し，ズーミング・フィルタリングといった詳細情報へのアクセス機能を提供する。これらの既存のソフトウェア理解支援ツールは，プログラムの可視化に関数コールグラフやUML図を用いるが，ソフトウェアごとの部品粒度ばらつき，部品構成の違いなどから汎用性，プログラムの概要表現の正確性に問題がある。ソフトウェア理解に役立つプログラムの可視化や文書化のためには，適切なモジュール識別とモジュール間の相互関

† メイテック

連といった情報が必要となる。一方，仕様書中のプログラムの概要や機能構成を表現する図群の多くは手作りであり，関数コールグラフのようにソースコードから自動生成することは難しい。

本研究では形式的概念分析（Formal Concept Analysis)[1]を用いてCプログラムのモジュールの識別とグルーピングを行い，ソフトウェア理解への適用としてプログラム概要図の自動生成系を開発する。ライブラリ，ファイルといったCプログラムの定義関数の分類は，直感的なソフトウェア理解と必ずしも合致しない。ファイルを跨るデータ構造，制御やデータフローを扱うとき，大規模なファイル内でより詳細な分類を行うときは洗練された分類法が必要となる。形式的概念分析は対象の性質を基に分類する手法であり，プログラムの保持する様々な情報を分類し，体系付けすることを可能にする。形式的概念分析のグルーピング手法として水平型分解[1], コンセプト分割[2]が提案され, C++プログラムのリファクタリングやプログラム・スライシング[6]に応用されている。

適用実験として，gzip1. 2. 4，bison1. 28など6種類のオープンソースソフトウェアを対象とし，定義関数内での構造体型データの利用に基づくグルーピングを行なった。またグルーピングをCプログラムの関数コールグラフと組み合わせ，プログラムの概要図を表現した。この実験結果とファイルによる分類や実行時オプションによる分類を比較し，概念分析を用いることでソフトウェアの実像をより正確に表すグルーピングが可能であること，プログラムの可視化素材の自動生成を通じて，ソフトウェア理解ツール開発に適用可能であることを示す。実装としてグルーピングとプログラム概要図の自動生成系をJavaとPerlプログラムを用いて作成した。

第2章で概念分析，第3章でCプログラムのグルーピング，第4章でグルーピングの応用例としてプログラム概要の作成プロセスを示す。第5章で実装を，第6章で本研究のソフトウェア理解への適用可能性の考察を示す。第7章は関連研究，第8章は結語である。

２．形式的概念分析とグルーピング

本章では形式的概念分析と概念束の分割・グルーピングの基本事項を説明する。2.1節で形式的概念分析を，2.2節で水平分解，コンセプト分割，2.3節で詳細グルーピングを説明する。

２．１　形式的概念分析

形式的概念分析は対象の潜在的な関係を分析する手法である。対象からその要素をあらわすオブジェクト，性質をあらわす属性，オブジェクトと性質の間の二項関係を定義する。オブジェクト集合と属性集合の対からなるコンセプトを抽出し，それらのコンセプトをコンセプト束として表現する。概念束はガロア束の構造を持つ。コンテクスト K はオブジェクト全体の集合 O，属性全体の集合 A，及び O と A の二項関係 $R \equiv O \times A$ の三項組である。$o \in O$，$a \in A$ において，$(o, a) \in R$ ならば，オブジェクト o は属性 a を持つという。

$$K := \{A, O, R(= A \times O)\} \tag{1}$$

関数 σ はオブジェクトの集合 $O \in O$ を引数にとり O の全ての要素が共通に持つ属性の集合を返す：

$$\sigma(\mathrm{O}) := \{a \in A \mid \forall o \in O, (o, a) \in R\} \tag{2}$$

関数 τ は属性の集合 $A \in A$ を引数にとり A のすべての属性を持つオブジェクトの集合を返す：

$$\tau(\mathrm{A}) := \{o \in O \mid \forall a \in A, (o, a) \in R\} \tag{3}$$

$A = \sigma(O)$ かつ $O = \tau(A)$ であるとき，(O, A) の組をコンセプトと呼ぶ。コンセプト $C = (O, A)$ の**外延**を $ext(C) = O$，**内包**を $int(C) = A$ と定義する。

$$(O, \sigma(O)) \tag{4}$$

をトップコンセプトと呼ぶ。

$$(\tau(\mathrm{A}), \mathrm{A}) \tag{5}$$

をボトムコンセプトと呼ぶ。任意のオブジェクト o について，

$$(\tau(\sigma(\{o\})), \sigma(\{o\})) \tag{6}$$

をアトミックコンセプトと呼ぶ。

２．１．１　コンセプトの半順序関係

二つのコンセプト（O_i，A_i），（O_j，A_j）が与えられたとき，

$$(\tau(A_i \cup A_j),\ A_i \cup A_j) \tag{7}$$

をそれらのスーパーコンセプトと呼ぶ。次にコンセプト上の半順序関係$\subseteq$を定める。

$$(O_1, A_1) \subseteq (O_2, A_2) \Leftrightarrow (O_1 \subseteqq O_2, A_1 \supseteqq A_2) \tag{8}$$

コンセプトの集合は半順序$\subseteq$の下で束をなす。任意の二つのコンセプトに対して下限$\wedge$と上限$\vee$は次式で定められる。

$$(O_1,\ A_1) \wedge (O_2,\ A_2) := (O_1 \cap O_2,\ \sigma(O_1 \cap O_2)) \tag{9}$$

$$(O_1,\ A_1) \vee (O_2,\ A_2) := (\tau(A_1 \cap A_2),\ A_1 \cap A_2) \tag{10}$$

コンセプト束はガロア束であり，図１のように両方向の順序関係を持つ。すなわちオブジェクトの集合は上向きに，属性の集合は下向きに部分集合に基づく順序関係$\subset$をなす。コンセプト束を形成するために(6)，(7)を満たすコンセプトを形成し，順序関係(8)を基に束を作成してゆく。

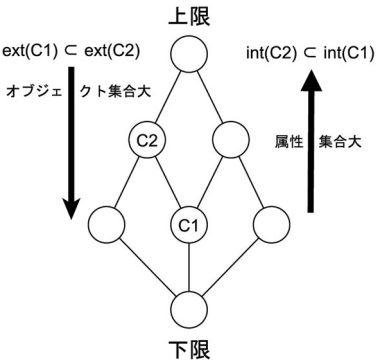


図１　コンセプト束の図式化

２．２　コンセプトの分類

束構造に基づいてオブジェクト集合のグルーピングを行う。ここではコンセ

プト分割[2]と水平型分解[8]によるグルーピング手法を紹介する。

2.2.1　コンセプト分割

コンセプト分割 P_i は全てのオブジェクトがもれなく分割されている分割であり，文献［2］で生成アルゴリズムが提案された。

$$P_i=\{(O_0,\ A_0),\ \cdots,\ (O_{n-1},\ A_{n-1})\},$$

$$where \bigcup_k O_k = O,\ \ O_i \cap O_j = \emptyset,\ \ i \neq j$$

ここであるコンテクストについて，

$$\forall x, y \in O,\ \ \sigma(\{x\}) \subseteq \sigma(\{y\})$$

が成立するとき，コンテクストは *well-formed* であるという。コンテクストがコンセプト分割を持つ必要十分条件はコンテクストが *well-formed* であることである。コンテクストが *well-formed* でない場合，$a \notin \sigma(\{y\})$，$a \in \sigma(\{x\})$ である属性 a に対して，その否定の属性 $\neg a$ を A に追加して，*well-formed* なコンテクストを形成する。

2.2.2　水平型分解

水平型分解[8]はあるコンセプト束を互いに独立なコンセプト束を部分束として持つコンセプト束に分解する手法である。図2（a）のコンテクストで関係が定義されていない部分に斜線をひく。このコンテクストから生成されるコンセプト束は図2（b）の形となる。

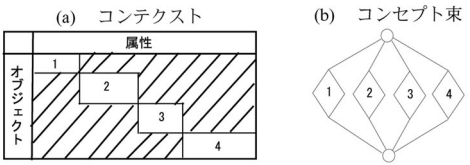


図2　水平型の分解構造

逆に複数のコンセプト束を水平に分解された部分束として組み立てることで1つのコンセプト束を形成することを水平和という。コンセプト束 L_1，L_2，…，L_n の水平和 L は

$$L := \sum_{i=1}^{n} L_i = \{\top, \bot\} \bigcup_{i=1}^{n} L_i \backslash \{\top_i, \bot_i\} \quad (11)$$

である。$\top$，$\top_i$は対応するコンセプト束の上限コンセプト，$\bot$，$\bot i$は下限コンセプトを表す。コンセプト束が水平和Lであるならば水平型分解が可能である。

3．C プログラムグルーピング実験

前節で説明した形式的概念分析に基づくグルーピングを，1000行（1 KLOC: Kilo Lines Of Code）から100万行（1000KLOC）程度の C プログラムに適用する。プログラムのデータ構造・フローの表現という観点からグルーピングを行なうため，定義関数をオブジェクト，関数内での構造体型データ利用を属性として用いる。

3．1　コンセプト束の形成

定義関数を基本要素とし，各関数の構造体の利用に着目してグルーピングを行う。前処理として対象とする C プログラムからコンテクストを生成する。コンテクストから概念束と各種の分割を行う。

3．1．1　オブジェクト

形式的概念分析のオブジェクトとしてプログラムの構成単位であるファイル，関数，ブロック，文が利用できる。Rigi[9]や SHriMP[10]といったプログラム理解支援ツールのモジュール単位はファイル，関数である。またオブジェクトとして文(statement)を用いる例として，Tonella らのインパクト解析や program slicing の適用研究[6]がある。本研究ではグルーピング結果をプログラム理解支援ツールの素材として活用するため関数をオブジェクトとしてプログラムのグルーピングを行う。

3．1．2　属性

関数をオブジェクトとするとき，大域変数や構造体の利用は関数間の共通する性質である。形式的概念分析は属性の数が多くなるとグルーピングの計算量が膨大となるため，プログラムのデータ構造をおおまかに表現する構造体型に着目する。構造体型は対象とするプログラムの重要なデータを表現するときに

用いられる。構造体型の利用とは，大域変数，関数パラメータでの構造体型の利用である。

3．2　gzip-1.2.4への適用実験

表1は，オブジェクトを関数，属性を構造体型とするgzip-1.2.4のコンテクストである。この表では便宜的に属性が共通する複数の関数をひとつのオブジェクトとして表わしている。gzip-1.2.4内の定義関数は99個，構造体型は6個である。このコンテクストから図3のコンセプト束を形成する。各ノードはコンセプトを表し、各ノードの上部に初出の属性，下部に初出のオブジェクトを表示する。

表1　gzip-1.2.4 のコンテクスト

		属性						
		tree_desc	huft	option	ct_data	stat	config	tree-desc ct_data
オブジェクト群	o_1	√						
	o_2		√					
	o_3			√				
	o_4				√			
	o_5					√		
	o_6						√	√
	o_7	√			√			√
	o_8							√

o_1 = build_bl_tree, build_tree, gen_bitlen, flush_block

o_2 = huft_build, inflate_codes, huft_free

o_3 = _getopt_internal, getopt, getopt_long, main

o_4 = init_block, ct_init, ct_tally, scan_tree, send_tree, set_file_type, send_all_trees, gen_codes, flush_block, compress_block build _bl_tree, pgdown_heap

o_5 = check_ofname, name_too_long, treat_file, same_file, do_stat, copy_stat, reset_times, treat_stdin, get_istat

o_6 = lm_init

o_7 = build_bl_tree, flush_block, gen_bilten

o_8 = send_tree, scan_tree

3．2．1　水平分解

図3の概念束は図4のように水平分解で5種類のグループに分類される。各部分束は相互に重複する属性を持たず，またそれぞれのオブジェクトに対応する関数集合は互いに素である。そのため水平分解は対象の大まかな分類を表現する。

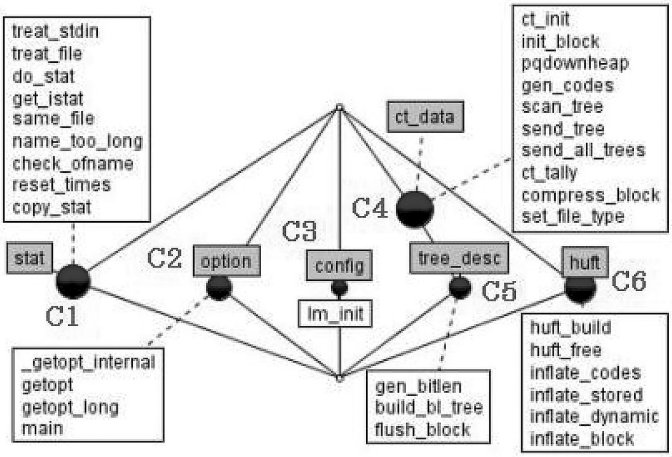
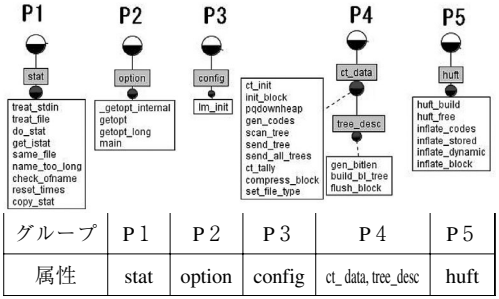


図3　コンセプト束 gzip−1．2．4



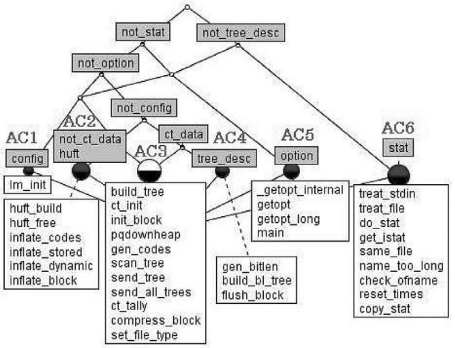
グループ	P1	P2	P3	P4	P5
属性	stat	option	config	ct_data, tree_desc	huft

図4　コンセプト束の水平型分解

3．2．2　コンセプト分割

図5はコンテクストの *well-formed* 化アルゴリズムに基づいて否定の属性

not-config，not-stat，not-option，not-ct_data，not-tree_descを導入し形成したコンセプト束である。図5のアトミックコンセプト集合（AC1～AC6）によるグルーピングはgzip1.2.4の全関数をいずれかのグループに振り分ける。



	属性	否定属性
AC 1	config	not-option, not-stat, not-tree_desc
AC 2	huft	not-ct_data, not-option, not-config, not-stat
AC 3	ct_data	not-option, not-config, not-stat, not-tree_desc
AC 4	ct_data, tree_desc	not-option, not-config, not-stat
AC 5	option	not-tree_desc, not-stat
AC 6	stat	not-tree_desc

図5　well-formed コンセプト束 gzip−1.2.4

4．グルーピングの適用例

本章では，gzip−1.2.4の水平分解，コンセプト分割を用いたCプログラム定義関数のグルーピングをソフトウェア理解に適用する例を示す。コンセプト分割によるグルーピングは対象をその性質に基づいて分類する。一方，水平分解は複数の性質を共有するグループ群をまとめる。対象や属性の選び方によりプログラム要素のさまざまなグルーピングが可能であり，例えば次のようなソフトウェア理解への適用が可能である。

（1）　プログラム仕様書中の関数の項目わけ

（2）　コールグラフやUML図の簡略化

（３）　プログラムの可視化

コールグラフやUML図では関数定義数が増えると視認性が悪化する。定義関数の適切なグルーピングによりこれらの図の視認性が向上する。前章のgzip−1.2.4プログラムのグルーピングは構造体利用という性質による分類でありデータ構造を表現する。本研究では関数コールグラフ上でこの水平分解とコンセプト分割によるグルーピングを表し，データ構造とデータフローを表現するプログラムの概要図を作成する例を示す。

４．１　コールグラフ

gzip−1.2.4のプログラムサイズはおおよそ8200行，定義関数99個のプログラムである。コールグラフは静的解析により作成，C言語の組み込み関数は除外し，同一関数の複数の呼び出しは一本のエッジで表現する。関数ポインタによる関数呼び出しは静的解析で取得されるものを対象とし，実際に呼び出した関数との間にエッジを引く。

４．１．１　コンセプト分割：グループの表現

定義関数99個でもコールグラフは複雑なものとなる。プログラムの概要表現するために，グループをノードとして表現し，またグループに属さない関数を省略することでコールグラフのノード数を減らし，直感的に理解可能なプログラム概要図を作成する。

表２　gzip-1.2.4のコンセプト分割

コンセプト	AC１	AC２	AC３	AC４	AC５	AC６
属性	config	huft	ct_data	ct_data, tree_desc	option	stat
関数の数	1	6	11	3	4	8

ここではコンセプト分割の各コンセプトに対応するグループに属する関数のノードを表２にしたがって表示する。図６でグループは六角形ノードで表現される。エッジはグループに属する関数間で直接コール関係がある場合は実線で，途中で別の関数を含む場合は点線で表現する。

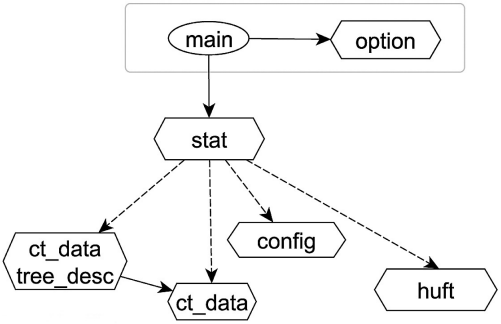


図6　gzip−1.2.4：グルーピング図

4.1.2　水平分解：グループの再編成

作成されるプログラム概要図を関数ドキュメントと連携して活用するためにはグループに属さない関数の取り扱いも考慮する必要がある。大まかな分類を表す水平分解によるグルーピングを用いて，グループに属さない関数をその性質や呼び出し関係に基づきグループに組み入れる。

（1）　関数の呼び出し関係によるグループ編入

C言語の定義関数は手続きの複雑な関数を分割して作成される場合がある。例えば構造体の利用によるグルーピングは特定のデータ構造，データフローに着目するため，データ構造についてグルーピングを行なう場合はこのグループに属さないがメンバ関数の働きをする関数が存在する。ここでは特定のグループからのみ呼ばれる関数をメンバ関数とみなし，当該グループに編入する。その結果，表3のようなグループに再編できる。

表3 gzip−1.2.4 の水平分解とグループ再編成

グループ：属性	P 1:stat	P 2:option	P 3:ct_data, tree_desc	P 4:huft	P 5:config
関数の数	9	3	14	6	1
編入した関数	8	8	5	1	0
合計	17	11	19	7	1

（2）　ユーティリティ関数

一方，複数のグループから呼ばれる関数は汎用性の高いドメイン独立の関数とみなすことができる。例えば入出力やメモリ操作といったユーティリティ関数などである。このような関数を見つけるためにはプログラム解析が必要となるが，Gzip ではユーティリティ関数ファイル util.c が存在するので，これをひとつのグループとして扱う。

（3）　その他：エントリーポイント関数など

main といったデータ処理の開始点であるエントリーポイント関数をノードとしてグラフに加える。一般に main 以外のエントリーポイント関数を特定するためにはソースコードを読解する必要がある。しかし Gzip では関数ポインタという形で zip，unzip 関数といったエントリーポイント関数が特定できる，他にデータ圧縮手続きを実装する関数 lzw，解凍手続きの unlzw，unlzh，unpack 関数が存在する。また各エントリーポイント関数の各ノードは，呼び出し下位の関数も含むものとし，それ以外の所属なし関数は省略する。

表 4　グループ非所属関数

	Utility	zip	unzip	lzw	unlzw	unlzh	unpack	未属
関数の数	17	6	3	1	1	11	3	2

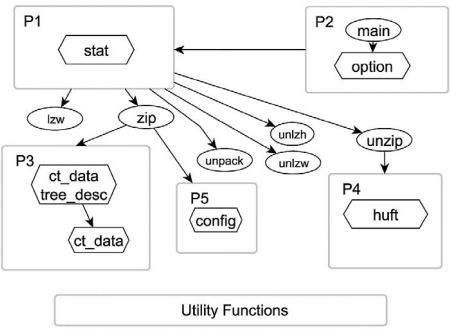


図 7　gzip-1.2.4：プログラムの概要図

水平分解によるグループを長方形領域で表し図7を作成する。グループ再編成を行い，エントリーポイント関数をノードとして加える。図6のグラフ構造はソースコードからgraphml形式ファイルとして自動生成される。図7は図6を基にインタラクティブに作成する。図6，図7はグラフ編集系yEdを用いて加工したものである。

5．実装

本研究で提案したCプログラムを対象としたグルーピングの自動生成系の実装を行った（図8）。自動生成系はCプログラムのソースコードからコンテクスト，コンセプト束やグルーピングを生成する。またソフトウェア理解への適用として関数コールグラフ，グルーピングとの重ね合わせ，グルーピング図を自動生成する。ソースコードの構文解析，コンテクストの生成系はSapidとCプログラムで実装した。また，概念束の生成系はJavaプログラムとPerlプログラムで実装した。

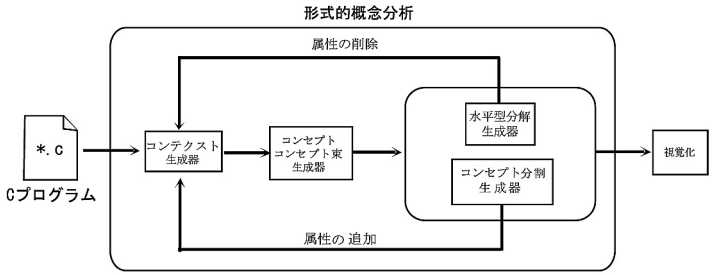


図8　実装図

Cプログラムのソースコードから Sapid のソフトウェアデータベース SDB を経てコンセプト束と各種グルーピングが自動生成される。同じくソースコードから関数コールグラフ，グルーピング図が自動生成される。最終的なプログラムの概要図はエントリーポイント関数とユーティリティ関数の指定により作成する。

６．考察

ソフトウェア理解への適用例として，関数コールグラフ上でグルーピングを表現し，プログラムの概要図を作成した。この概要図は単なるコールグラフの抽象化ではなく，プログラムの構造や要素間の関係を適切に表現することを意図する。この適用例が理解支援ツールとして実用的であることを例証する。6.1節でファイルや実行時オプションによる関数のグルーピングとの比較，6.2節で５個の中規模サイズのソフトウェアのグルーピング例を示す。6.3節で，ソフトウェア理解支援ツールへの形式的概念分析の適用可能性について考察する。

６．１　グルーピングの手法比較

本研究で提示した定義関数群の構造体データの利用によるグルーピングを，所属ファイルによる関数グルーピング，実行時オプションによる関数グルーピングと比較する。

（１）ファイルによる分類との比較

前章で作成したプログラム概要図と所属ファイルによる関数のグルーピングをオーバーラップした図が図９である。概要図で示された水平分割および，そのほかのグループを長方形の領域として，ファイルを点線の枠で囲まれた領域で表す。

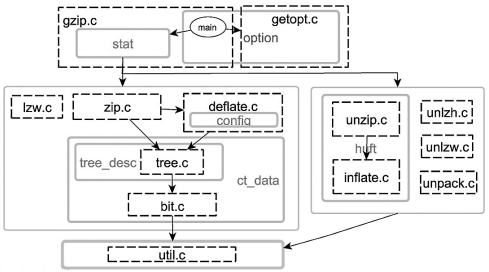


図９　gzip-1.2.4：本分類図とファイル構成

図9から，本グルーピング手法がファイルをまたがるグループ（構造体ct_data）やファイル内の詳細な分類（gzip.cにおける構造体statと構造体optionのグループ）を提供し，プログラムの構成やデータの流れを適切に表現することが示される。

（2）実行時オプションによる分類との比較

実行時オプションを用いて実際に実行される関数を分類できる。gzip−1.2.4のオプションと概要図のグループとの関連表が表5である。表5でオプションなしはデータ圧縮，−dオプションは展開を，−lhVはヘルプやバージョンを表示する。この表より，本グルーピング手法が実行時オプションにより実行される関数群のグループと対応していることがわかる。

表5　gzip−1.2.4のオプションフラグによる分類

	P1	P2	P3	P4,P5 zip, lzh	unlzh, unlzw, unpack
no option	o	o		o	
-lhV		o			
-d(注1)	o	o	o		
-d(注2)	o			o	
-Z	o	o			o

(注1)マクロフラグ PKZIP_MAGIC
(注2)マクロフラグ PACK_MAGIC, LZW_MAGIC, LZH_MAGIC

6．2　他のソフトウェアの適用

表6はさまざまなCプログラムを対象に，前節の形式的概念分析を適用した結果であり，定義関数群の構造体型データの利用によるグルーピングが中規模サイズのソフトウェア（0.6KLOCから80.87KLOC）に適用可能であることが示される。

gzipよりもサイズの大きいbison−1.28のプログラム概要図と概念束が図10である。定義関数の数は378個，4種類の水平分解P1〜P4と13個のコンセプト分割からなり，グループP1は**枝狩り法**[5]でさらに3種類のグループ（Shift-

Reduce, Finite State Machine, Error Processing）に分けられる。

表6　Cプログラムの形式的概念分析の結果

プログラム（KLOC）	オブジェクト数	属性数	概念数	グループ数
dhrystone-bin（0.6）	6	7	7	3
gnugo-1.2（2.86）	30	2	4	2
patch-2.5（8.03）	116	17	143	5
bison-1.28（10.87）	162	21	867	13
bash（80.87）	332	30	382(注3)	20(注3)

（注3）属性分割［5］を適用

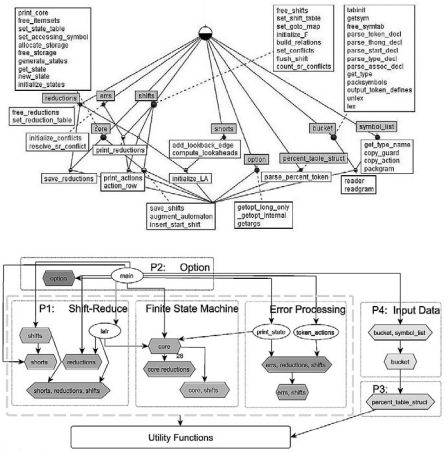


図10　bison-1.28の概要図

6.3　形式的概念分析とソフトウェア理解

Rigi[9]やSHRiMP[10]といったソフトウェア理解支援ツールはプログラム探索の基点となるプログラム概要図をユーザが自由に設計できる編集機能を提供している。しかしモジュール単位の選択やモジュール間の関係の設定はユーザの責任であり，ソフトウェアの適切なモジュール設定と関係付けが課題となっている。6.1節で示したように，本グルーピング手法はプログラムの概要

情報の表現という観点で，ファイルや関数を基礎とするモジュール化よりも適切なモジュールを提供している。

6．2節では中規模サイズのソフトウェアについて概念分析によるグルーピングが適用可能であることを示した。しかし，実用的なツールであるためには，よりサイズの大きいプログラム，構造体型データを利用しないプログラムへの対応も必要である。そのため，属性となる構造体型数の多いbashなどのソフトウェアに概念分析において影響の大きい属性を選択する手法である**属性分析**[5]による概念分割を適用する。またサイズの小さいソフトウェアや構造体型データを用いないソフトウェアについては，グローバル変数を代替の属性とすることで対応する。

概念分析，グルーピングの処理系はSapid［12］を用いて実装した。SapidはCやJavaのソースプログラムからプログラムの要素とその関係を表すデータベースを提供するので，関数，変数といったプログラムの任意の要素を対象とするグルーピングが可能である。データベース中の各プログラム要素はSPIE（Source Program Information Explorer）［18］を通じて詳細情報と連携可能であり，実用的なソフトウェア理解支援ツールが実現できる。

7．関連研究

形式的概念分析を用いたCプログラムのモジュール検出法が文献[1]で紹介されている。オブジェクトを関数，属性を構造体とし，C++プログラムへのリファクタリング手法を提案している。本研究ではこのオブジェクトと属性の選び方を参照し，形式的概念分析をソフトウェア理解に適用する。文献[6]は，オブジェクトをプログラム中の文，属性をスライス対象の文としてプログラムスラシングやインパクト解析を行い，プログラムの抽象実行や依存解析に関する情報を提供する。文献[7]はオブジェクトを関数，属性を関数呼び出し式としてコンセプト束を構成し，その束に基づくソフトウェア構造の探索手法を提案している。本研究のプログラム概要図や詳細図の作成手法と組み合わせることで，本研究の概要図生成プロセスを効率化する。

ソフトウェア理解支援ツールに関する関連研究として，Müller のグループが開発した Rigi[9]を本研究では参照している。Rigi はソフトウェアの理解と，再ドキュメント化を目的とした対話型のソフトウェア可視化ツールである。Rigi を用いて SHriMP[10]，JCosmo[17]といったソフトウェア理解支援ツールが開発されている。Rigi は関数呼び出し図からコンポーネントを抽出し，そのグラフ表現の生成する。しかしプログラムの性質に基づいたコンポーネント化を保証しない。形式的概念分析のよる定義関数のグルーピングは，プログラムの性質に基づいてコンポーネントを識別し，グループを形成する。SHriMP は論文[11]でグラフの幾何学的な省略法により概要図を作成する。グラフの幾何学的簡略化は我々の関数コールグラフ簡略化プロセスに利用可能である。JCosmo はソースプログラムの Code Smell 検出に特化したツールであり，Java ソースコードの品質確認に使われる。

プログラム概要情報の提供と詳細情報への探索を「地図」や「ナビゲーション」といった概念でモデル化し，ソフトウェア理解を平易する試みが GSEE[16]や Portable Bookshelf[15]である。GSEE[16]はソフトウェア探索を旅行 tours に喩え，探索手段をバックパッカ backpacker やリムジン limousines といっ比喩的な表現でソフトウェア探索をモデル化する。Portable Bookshelf[15]は、ソフトウェアを本棚に喩え，視覚化ツールやナビゲーション機能を用いて大規模なソフトウェアを表示する。これらのアイデアは先進的なソフトウェア探索ツールの可能性を示す。

8．結語

形式的概念分析によるグルーピングを用いて C プログラムの関数を分類し，関数コールグラフ上でグループ間の相互関係，階層関係を表現し，ソフトウェア理解に適用した。また同手法を自動化し，プログラム概要図を自動生成する系を開発した。適用実験として 1 KLOC から300KLOC サイズの中規模の C プログラムを対象としてオブジェクトを関数, 属性を構造体型とする概念分析と，コンセプト分割と水平型分解によるグルーピングを行なった。また gzip−1. 2. 4

を対象として，グルーピングを関数コール図と組み合わせ，プログラム理解支援に適用可能なプログラムの概要図を作成した。

プログラムの要素と性質の組み合わせにより，本研究以外のグルーピング，ソフトウェアのモジュール化が可能である。形式的概念分析とグルーピングのプログラム理解支援への適用可能性を幅広く実証することを今後の課題とする。C++やJava言語といったオブジェクト指向言語への適用や，ソフトウェア理解支援ツールSPIEとの組み合わせも今後の課題である。

文　献

［1］Rudolf Wille, “Restructuring lattice theory: an approach based on hierarchies of concept”, In Ivan. Rival, editor, Symposium. on.Ordered Sets, pp.450−475, 1982.

［2］Michael Siff, Thomas Reps, “Identifying Modules via Concept Analysis”, IEEE Transactions on Software Engineering, Vol.25, pp.749−768, Nov-Dec1999.

［3］Michael Siff and Thomas Reps: A greedy approach to obtain object identification in imperative code. In Thrid workshop on Program Comprehension, pp.4−11, 1994.

［4］Thomas. Tilley, Richard. Cole, Peter. Becker, and Peter. Eklund. A: Survey of Formal Concept Analysis Support for Software Engineering Activities. In Proc. 1st International Conference on Formal Concept Analysis, 2003.

［5］Bernhard Ganter, Rudolf Wille, “Formal Concept Analysis: Mathmatical Foundations”, Springer, 1999.

［6］Paolo Tonella, “Using a Concept Lattice of Decomposition Slices for Program Understanding and Impact Analysis”, Software Engineering, IEEE Transactions on, pp.495−509, 2003.

［7］Richard Cole, Thomas Tilley, and Jon Ducrou, “Conceptual Exploration of Software Structure: A Collection of Examples”, Proceedings of 3rd International Conference on Concept Lattices and Their Applications, pp.135−148, 2005.

［8］Pestra Funk, Anke Lewien, and Gregor Snelting, “Algorithms for Concept Lattice Decomposition and their Application”, Technical report, Computer Science Department, Technische Universitat Braunschweig, 1995.

［9］K. Wong: On inserting program understanding technology into the software change process. Workshop on Program Comprehension 1996 Proceedings（WPC1996）.

［10］S. R. Tilley: Domain-retargetable reverse engineering. Ph.D. Dissertation, Department of Com-

puter Science, University of Victoria, 1995.

[11] M.-A. D. Storey and H. A. Muller: Graph layout adjustment strategies. Graph Drawing 1995 Proceedings (GD1995).

[12] Yoshida Atsushi, Yamamoto Shinichirou and Agusa Kiyoshi: A Software Manipulating Language for a Meta CASE , The First International Congress on META-CASE 1995.

[13] Erich Buss and John Henshaw: Experiences in program understanding. Technical Report TR-74-105, IBM Canada Laboratory, Toronto, Ontario, Canada, July 1992.

[14] Arie van Deursen, et al: Viewpoints in software architecture reconstruction. In Proceedings 6 th Workshop on Software Reengineering (WSR04). Bad Honnef, 2004.

[15] R. Holt: Software Bookshelf. Overview and Construction. University of Toronto, March 1997.

[16] Jean-Marie Favre, et. al: A New Approach to Software Exploration: Back-packing with GSEE, Laboratoire LSR-IMAG, University of Grenoble, France.

[17] M.G.J. van den Brand, et al.: The ASF+SDF Meta-Environment:a component-based language development environment. Proceedings of Compiler Construction 2001.

[18] Fukuyasu Naoki, Yamamoto Shinichirou and Agusa Kiyoshi: An Evolution Framework based on Fine Grained Repository, International Workshop on Principles of Software Evolution, 1999.