社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2007－SE－155 (20)
2007／3／23

形式的概念分析を用いたグルーピングによるCプログラム理解支援手法

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

要 旨

形式的概念分析とは対象とする事物から同じ性質を共有するグループを選択する手法である．本稿は形式的概念分析を用いて，Cプログラムを対象として関数のグルーピングを行い，プログラム理解支援を指向したグルーピングの方法とその提示方法を提案する．形式的概念分析は，要素をその性質を基に分類可能なので，要素と性質を選択，解析することにより，様々なプログラムの保持する情報を分類し，体系付けすることができる．

A Supporting Method for Understanding C Programs with Grouping via Formal Concept Analysis

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

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 and visualizing method for C program functions, which is applicable to software understanding. Formal Concept Analysis can cluster elements by their properties. Therefore, by choosing and analyzing various program elements and properties, we can cluster and schematize information on the programs.

1 はじめに

ソフトウェア開発において，ソフトウェアの保守の重要性が高まっている．ソフトウェア保守は，ソフトウェアの継続的な利用のために，必要に応じてソフトウェアに変更を加えて品質の維持・向上をはかることを目的とする．ソフトウェアの保守の際，修正するべき点を見つけ，改良を行うために，ソフトウェア開発者はプログラム全体の動作を理解しなければならない．そのためにソースプログラムを読むという作業が必要となる．

プログラムは多数の部品から構成されている．Cプログラムはライブラリ，ファイルといった部

品単位で機能的に分割されている．よって通常はこれらの分割を手がかりとしてソースプログラムを解読，探索する．しかし必ずしも直感的な理解に適するようにプログラムが部品化されているわけではない．そこで，プログラムを個々の要素の性質や振る舞いに基づいて分割し，ソフトウェアの解読や探索に役立つようにグループ化する手法が求められる．

形式的概念分析 [1] がプログラムの分割・グループ化手法として提案され，リファクタリング [7] やプログラム・スライシング [4] に応用されている．形式的概念分析は，性質を基に事物を分類する方法である．本稿は，形式的概念分析を用いて，C プログラムを対象としてプログラムのグループ化を行い，ソフトウェア理解に応用する．形式的概念分析は，要素をその性質を基に分類可能なので，要素と性質を選択，解析することにより，様々なプログラムの保持する情報を分類し，体系付けすることができる．

2 形式的概念分析

本節は文献 [3] で定義され形式的概念分析 (Formal Concept Analysis) の基本事項を説明する．オブジェクト o は事象や現象の対象であり，その集合を $\mathcal{O}$ で表す．属性 a とは対象の性質であり，属性の集合を $\mathcal{A}$ で表す．コンテクスト $\mathcal{K}$ はオブジェクト集合 $\mathcal{O}$，属性集合 $\mathcal{A}$，及び $\mathcal{O}$ と $\mathcal{A}$ の 2 項関係 $\mathcal{R} \subseteq \mathcal{O} \times \mathcal{A}$ の組である．$o \in \mathcal{O}$，$a \in \mathcal{A}$ において，$(o_i, a_j) \in \mathcal{R}$ ならば，オブジェクト o_i は属性 a_j を持つという．

関数 σ はオブジェクトの集合 $O \subseteq \mathcal{O}$ 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:

$$\sigma(O) := \{a \in \mathcal{A} \mid \forall o \in O, (o, a) \in \mathcal{R}\} \quad (1)$$

関数 τ は属性の集合 $A \subseteq \mathcal{A}$ 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:

$$\tau(A) := \{o \in \mathcal{O} \mid \forall a \in A, (o, a) \in \mathcal{R}\} \quad (2)$$

$A = \sigma(O)$ 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．最初に式 (5)，(3)，(6) が満たされるコンセプトを生成する．いくつかのコンセプトには名前が付いている．全ての属性を持つオブジェクトからなる

$$(\tau(\mathcal{A}), \mathcal{A}) \quad (3)$$

をボトムコンセプトと呼ぶ．全てのオブジェクトからなる

$$(\mathcal{O}, \sigma(\mathcal{O})) \quad (4)$$

をトップコンセプトと呼ぶ．任意のオブジェクト o について，

$$(\tau(\sigma(\{o\})), \sigma(\{o\})) \quad (5)$$

をアトミックコンセプトと呼ぶ．これは $\mathcal{O}$ が含む外延が最小のコンセプトである．2 つのコンセプト $(O_i, A_i), (O_j, A_j)$ が与えられたとき，

$$(\tau(A_i \cup A_j), A_i \cup A_j) \quad (6)$$

をそれらのスーパーコンセプトと呼ぶ．コンセプトの上に次の半順序関係を定める．

$$(O_1, A_1) \leq (O_2, A_2) \Leftrightarrow (O_1 \subseteq O_2) \wedge (A_1 \supseteq A_2) \quad (7)$$

コンセプトの集合は式 (7) の半順序の下で束をなす．任意の 2 つのコンセプトに対して下限と上限は次式で定められる．

$$(O_1, A_1) \wedge (O_2, A_2) := (O_1 \cap O_2, \sigma(O_1 \cap O_2)) \text{ (下限)} \quad (8)$$

$$(O_1, A_1) \vee (O_2, A_2) := (\tau(A_1 \cap A_2), A_1 \cap A_2) \text{ (上限)} \quad (9)$$

コンセプト束を図式化すると図 1 のようになる．図で上のコンセプトほど大きい，すなわち $C_1 < C_2$ とする．オブジェクトの集合は上にいくほど大きくなる，すなわち $ext(C_1) \subset ext(C_2)$ である．属性の集合は下にいくほど大きくなる．すなわち $int(C_2) \subset int(C_1)$ である．

束構造に基づいて，全てのオブジェクト集合に対するグルーピングを行う．次にコンセプト分割 [2] と水平型分解 [6] によるグルーピング手法を紹介する．

コンセプト分割　文献 [2] で提案されているコンセプト分割について説明する．コンセプト分割

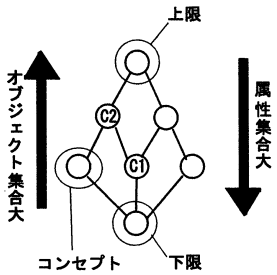

図 1: コンセプト束の図式化

$\mathcal{P}_i$ は 全てのオブジェクトがもれなく分割されているコンセプトの集合

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

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

あるコンテクスト $(\mathcal{O}, \mathcal{A}, \mathcal{R})$ について,

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

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 $!a$ を $\mathcal{A}$ に追加して，*well-formed* なコンテクストを作ることができる．文献 [2] にはコンセプト分割を求めるアルゴリズムが示されている．

水平型分解　文献 [6] で提案されている水平型分解について説明する．水平型分解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) のコンテクストで斜線部にはチェックが付かないとする．このようなコンテクストから生成されるコンセプト束は図 2 (b) の形になる．

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 $\mathfrak{L}_1, \mathfrak{L}_2, \cdots, \mathfrak{L}_n$ の水平和 (*horizontal sum*) $\mathfrak{L}$ は $\mathfrak{L} := \sum_{i=1}^{n} \mathfrak{L}_i = \{\top, \bot\} \cup \bigcup_{i=1}^{n} \mathfrak{L}_i \backslash \{\top_i, \bot_i\}$ である．$\top$ はコンセプト束 $\mathfrak{L}$ の上限のコンセプトを表し，$\bot$ は下限のコンセプトを表す．コンセプト束 $\mathfrak{L}$ が水平和であるならば，コンセプト束 $\mathfrak{L}$ は水平型分解が可能である．

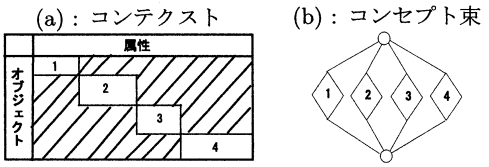

図 2: 水平型の分解構造

3　C プログラムへの形式的概念分析の適用

本節では、2 節で説明した，形式的概念分析に基づく分割・グルーピング手法をソースプログラムの解析と理解支援に応用する．

形式的概念分析を用いた C プログラムの分割・グルーピング手法が文献 [4, 5] で紹介されている．文献 [4] 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う．これにより，プログラムの抽象実行や依存解析に関する情報を獲得する．文献 [5] 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，その束に基づく効率的なソフトウェア構造の探索を提案している．

本稿では関数のグループ化をプログラム構造の理解に利用する．そのために関数をまたがり，ファイル単位で宣言する構造体に着目する．

オブジェクトを関数，属性を構造体型，構造体変数とし，プログラム中の関数の宣言文内に構造体型，構造体変数が含まれるならば，オブジェクトと属性に関係があるものとし，図 3 を基に以下の順序で C プログラムに形式概念分析の適用を行う．

図 3 を基に以下の順序で C プログラムに形式概念分析の適用を行う．

1. C プログラム内の関数をオブジェクトとし，構造体型を属性とする．
2. オブジェクトと属性との関係からコンテクストを組み立てる．その際，同じ内包を持

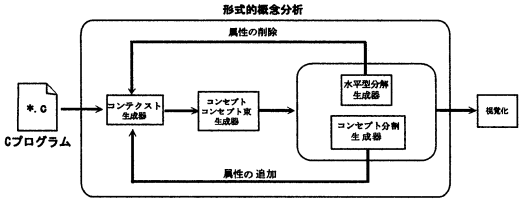

図 3: 概要図

つ複数のオブジェクトの要素，同じ外延の集合を持つ複数の属性の要素が存在したならば，それら複数の要素を 1 つの要素に統合する．文献 [3] にはその正当性が証明されている．

3. コンテクストからコンセプト束を形成する．
4. コンセプト分割，水平型分解により関数のグルーピングを行う．
5. グルーピングされた構造を基に視覚化を行う (4.2 節)．

実際の C プログラムを形式的概念分析した例を示す．例として gzip-1.2.4 を対象として形式的概念分析を行う．gzip-1.2.4 の関数は 95 個，構造体型変数は 11 個ある．

3.1 コンセプト束

構造体を扱わない関数は適切なグループを決定できない．よってそれらの関数は除外する．結果として，gzip-1.2.4 の関数 95 個のうち，84 個がコンセプト束生成の対象外となった．要素統合した後のコンテクストを表 1 に表す．オブジェクトの集合 o_0 の要素 `checkofname`，`treat_file`，`treat_stdin` はそれぞれ同じ内包を持つオブジェクトである．よって 1 つのオブジェクトの要素として統合する．同様に o_6 と o_8，`static_dtree`，`static_ltree` と `d_desc`，`l_desc` もそれぞれ同じ外延を持つため 1 つの属性の要素として統合する．表 1 のコンテクストから図 4 の束構造を形成する．なお，束構造は次の通りに表記する．各ノードはコンセプトを表す．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．あるノードの属性は初出の属性に上位ノードの属性を加えたものである．あるノードのオブジェクトは初出のオブジェクトに下位ノードのオブジェクトを加えたものである．トップのノードは全てのオブジェクトを表示する．ボトムのノードは全ての属性を表示する．

表 1: gzip-1.2.4 のコンテクスト (統合済)

		属性								
		bl_desc	static_dtree, static_ltree	dyn_ltree	dyn_dtree	d_desc, l_desc	istat	configration_table	longopts	bl_tree
オブジェクト	o_0						√		√	
	o_1							√		
	o_2			√						
	o_3			√	√					
	o_4		√	√	√	√				
	o_5			√	√					√
	o_6	√		√	√	√				√
	o_7									√
	o_8		√							
	o_9									

o_0 = `check_ofname`, `treat_file`, `treat_stdin`, o_1 = `main`,
o_2 = `lm_init`, o_3 = `set_file_type`, o_4 = `ct_tally`,
o_5 = `flush_block`, o_6 = `send_all_trees`, `init_block`,
o_7 = `build_bl_tree`, o_8 = `send_tree`, `scan_tree`, o_9 = `ct_init`

3.2 コンセプト分割

not well-formed のコンテクスト (表 1) より，*well-formed* のコンテクストにするには新たに否定の属性 `not-dyn_tree`，(`not-static_dtree`，`not-static_ltree`)，`not-bl_tree` を追加する．否定の属性を追加したコンテクストから図 5 のコンセプト束を形成する．文献 [2] のアルゴリ

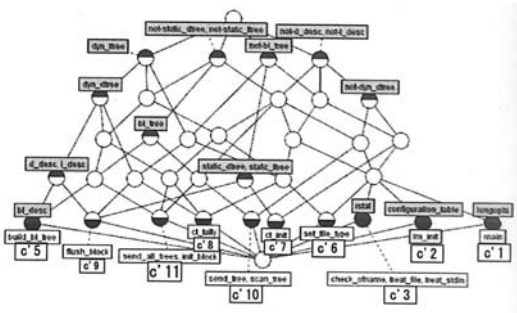

図 5: well-formed のコンテクストから形成されたコンセプト束 gzip-1.2.4

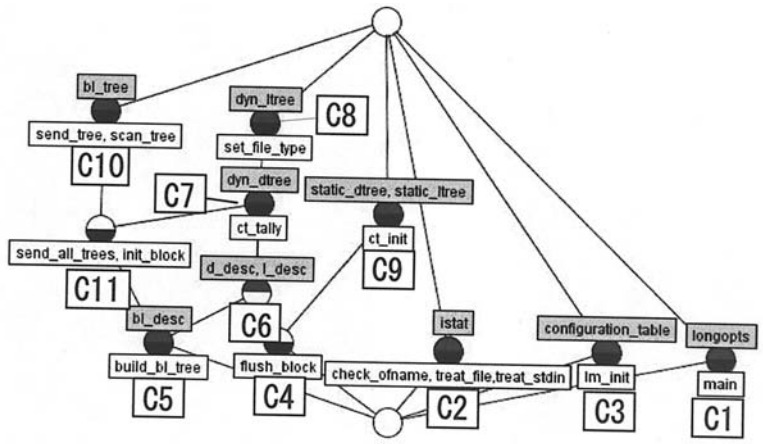

図 4: コンセプト束 gzip-1.2.4

ズムを用いてコンセプト束からコンセプト分割を生成する。もっとも分割数が多い分割をが基本 (atomic) 分割と呼び、その分割を構成するコンセプトをアトミックコンセプトと呼ぶ。図 5 のコンセプト束から 124 種の分割候補が生成され、基本 (atomic) 分割は c'1 から c'11 まで 11 個のコンセプトを含む。

gzip-1.2.4 で 14 個の構造体を持つ関数は {ct_tally}, {send_all_trees, init_block}, {check_ofname, treat_file, treat_stdin}, {main}, {build_bl_tree}, {ct_init}, {set_file_type}, {lm_init}, {scan_tree}, {flush_block} の 10 種のグループに分かれる.

3.3 水平型分解

図 4 のコンセプト束に水平型分解を行う. 図 6 のようにの 4 つの部分束に分解が可能される.

4 プログラム理解支援のためのグルーピング結果の利用法

プログラム理解支援にグルーピングの結果を利用するための方法を説明する.

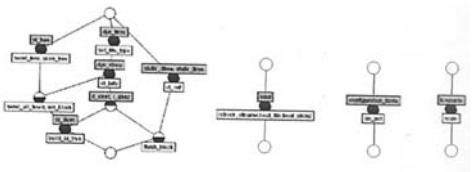

図 6: gzip-1.2.4 のコンセプト束による水平型分解

4.1 グループ間の関係

コンセプト束の構造に基づいて, 取得されたグループ間の独立・包含関係を定義する. この関係を用いて, 構造体変数を手がかりとしたプログラム探索を実現する.

独立 水平型分解により得られるグループ間の関係である. 関数のグルーピングをあらわすコンセプト束の水平型分解により得られた部分束は, 対応する属性である構造体型のみ依存する.

gzip-1.2.4 の場合, 水平型分解により関数群は 4 つの部分束に分解される (図 7). 属性は各関数宣言内で利用している構造体変数であり, 分類された 4 つのグループは相互に独立しているとみなすことができる.

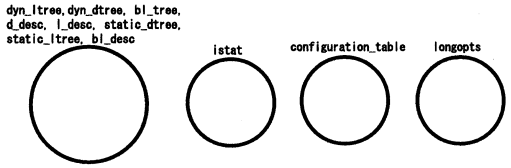

図 7: gzip-1.2.4 の独立関係

包含 包含関係は，コンセプト束の各コンセプトの内包 (intent) の上下関係に基づいて定義される．サブコンセプトの内包 (intent) にスーパーコンセプトの内包 (intent) が含まれている．構造体変数に着目すると，スーパーコンセプト内の構造体変数群は下位の構造体変数群を含む．図 8 中の **(a)** は上位のコンセプトの内包である **dyn_ltree**，下位のコンセプトの内包である **dyn_ltree**，**dyn_dtree** の関係を表す．**(b)** で構造体変数 **dyn_ltree**，**dyn_dtree** の領域と構造体変数 **dyn_ltree** のみの領域の関係を表現する．**(c)** は，上位の 2 つのコンセプトの内包である **bl_ltree** と **dyn_dtree** を下位のコンセプトが継承するので，図 **(d)** で表現される．このとき，**(d)** の構造体変数 **bl_ltree** のみの領域と構造体変数 **dyn_dtree** のみの領域をあわせ，3 つのグループ候補が存在する．

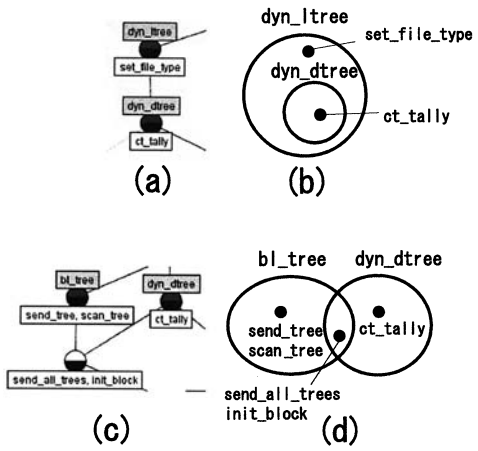

図 8: gzip-1.2.4 の包含関係

4.2 グルーピング結果を用いたプログラムの視覚化

構造体に基づくコンセプト分割は，データフローという側面での関数の間の関連性の大きさを明らかにする．一方，関数間の呼び出し関係はコントロールフローであり，通常コールグラフにより表現される．

関数の間のこれら 2 つの側面を 1 つの図で表現できれば，プログラムの全体像を効果的に把握することが期待される．

本節では文献 [9] で提案されている可読性を重視した複合グラフの描画アルゴリズムを応用し，プログラムの呼び出し関係とコンセプト分割を組み合わせた視覚化について述べる．

$\mathcal{G} := (\mathcal{V}, \mathcal{E}, \mathcal{C}, \psi^+, \psi^-, \phi^+, \phi^-)$
$\mathcal{V}$: ノード集合
$\mathcal{E}$: 隣接エッジ集合
$\mathcal{C}$: 包含エッジ集合
$\psi^+, \psi^- : \mathcal{E} \to \mathcal{V}$ ($\mathcal{E}$ の結合関数)
$\phi^+, \phi^- : \mathcal{C} \to \mathcal{V}$ ($\mathcal{C}$ の結合関数)

複合グラフ $\mathcal{G}$ はノード集合 $\mathcal{V}$，隣接エッジ集合 $\mathcal{E}$，包含エッジ集合 $\mathcal{C}$，$\mathcal{E}$ の結合関数 ψ^+ と ψ^-，$\mathcal{C}$ の結合関数 ϕ^+ と ϕ^- の組である．隣接エッジ集合 $\mathcal{E}$，包含エッジ集合 $\mathcal{C}$ はグラフの二つの側面を表現可能にする．隣接エッジ $e \in \mathcal{E}$ は始点 $\psi^+(e)$ から終点 $\psi^-(e)$ に向いているとする．包含エッジ $c \in \mathcal{C}$ は始点 $\phi^+(c)$ から終点 $\phi^-(c)$ に向くものとする．また包含エッジ集合 $\mathcal{C}$ からなるグラフは tree 構造である．

gzip-1.2.4 のコンセプト束 (図 4) を基に複合グラフを作成する．複合グラフのノードはコンセプトに対応する．すなわち $\mathcal{V} = \{C1, C2, \cdots, C11\}$ である．隣接エッジは関数間の呼び出し関係を表す．コンセプト C_i の関数からコンセプト C_j の関数を呼び出しているとき，$\psi^+(e) = C_i$，$\psi^-(e) = C_j$ である隣接エッジ e があるとする．

包含エッジは属性間の包含関係を表す．コンセプト束において，コンセプト C_i かつ C_j の直上にある，すなわち $\mathbf{covs}(C_j) \ni C_i$ であるとき，$\phi^+(c) = C_i$，$\phi^+(c) = C_j$ である包含エッジ c があるとする．このとき $int(C_i) \subsetneq int(C_j)$ である．

入れ子を許すノードを領域と呼び，特に最外の領域群は図 6 の水平型分解 $P1$ から $P4$ に対

応する．オブジェクト集合の完全な分割は *well-formed* のコンセプト束のアトミックコンセプト $c'1$ から $c'11$ で表される．これらを図 9 の複合グラフに組み込む．各アトミックコンセプトを楕円ノードとして，対応する領域のなかに描画する．また領域が表すコンセプト $C1$ から $C11$ の属性を，図 4 の記法に基づいて記述する．

プログラムの全体構造を表現するために，関数呼び出し関係を用いて各極小コンセプト間にエッジ E を引く．特定の属性 (構造体変数) を持たないコンセプトを $c'0$ とし，複合グラフの外側に配置する．

5 評価

5.1 グルーピングの性能

表 2 は様々な C プログラムを対象に，形式的概念分析を適用した結果である．形式的概念分析の計算には多量のメモリが必要であり，コンテクストの統合が必要メモリ量の軽減に大きく貢献した．

表 2: C プログラムの形式的概念分析の結果

プログラム名 (KLOC [a])	$\lvert\mathcal{O}\rvert$ [b]	$\lvert\mathcal{A}\rvert$ [c]	$\lvert c\rvert$ [d]	$\lvert\mathcal{P}\rvert$ [e]
whetstone (1.28)	6	2	4	2
dhrystone-2.1-bin (0.6)	6	7	7	4
gnugo-1.2 (2.86)	30	2	4	2
patch-2.5 (8.03)	116	17	143	544
bison-1.25 (10.87)	162	21	867	–[f]

[a]: Kilo Line of code
[b]: オブジェクト
[c]: 属性
[d]: コンセプト
[e]: コンセプト分割
[f]: メモリ領域を超えたため生成されず

5.2 ソフトウェア理解支援への応用

コンセプト分割を理解支援ツールへ応用する関連研究として，Müller らが開発している Rigi[8, 11] が挙げられる．Rigi はソフトウェア理解と，再ドキュメント化を目的とした対話式視覚ツールである．Rigi は関数呼び出し図からコンポーネントのグラフ化とその操作を実現する．しかし，Rigi ではプログラムの性質に基づいたコンポーメント化は実現されていない．本稿の形式的概念分析を用いた関数のグルーピングは，プログラムの性質に基づいたコンポーメント化を実現可能にし，これにより，開発者が意味的なグループを識別し，効率的にプログラム理解を進められることが考えられる．

6 むすびに

6.1 まとめ

本稿は C プログラムを対象にオブジェクトを関数，属性を構造体型として，プログラム分割と水平型分解の手法により構造体型に基づいた関数のグルーピングを行った．これらのグルーピングの結果を基に，構造体型の位置関係や，アトミックコンセプト間の関数呼び出し図のコンポーネント化の適用など C プログラム理解支援に役立てるための手法を考察した．

6.2 今後の課題

形式的概念分析を用いて C プログラムを分析するとき、関数や構造体型変数以外の要素からプログラムのグループ化を行うことが可能である。今後はどのような要素とその性質を用いて分析を行えば，どの点でプログラム理解に役立つのかを考察する必要がある．C プログラム以外に C++や Java 言語といったオブジェクト指向言語への適用も考えている．

また，Sapid Project[10] で開発されている SPIE(Source Program Information Explorer) はソースプログラムからクロスリファレンス表を HTML 形式で生成するプログラム理解支援ツールである．形式的概念分析により生成するグルーピング結果と SPIE を連携させた，プログラム理解支援のためのナビゲーションシステムの実現を今後の課題とする．

参考文献

[1] Rudolf Wille, "Restructuring lattice theory: an approach based on hierarchies of concept", In

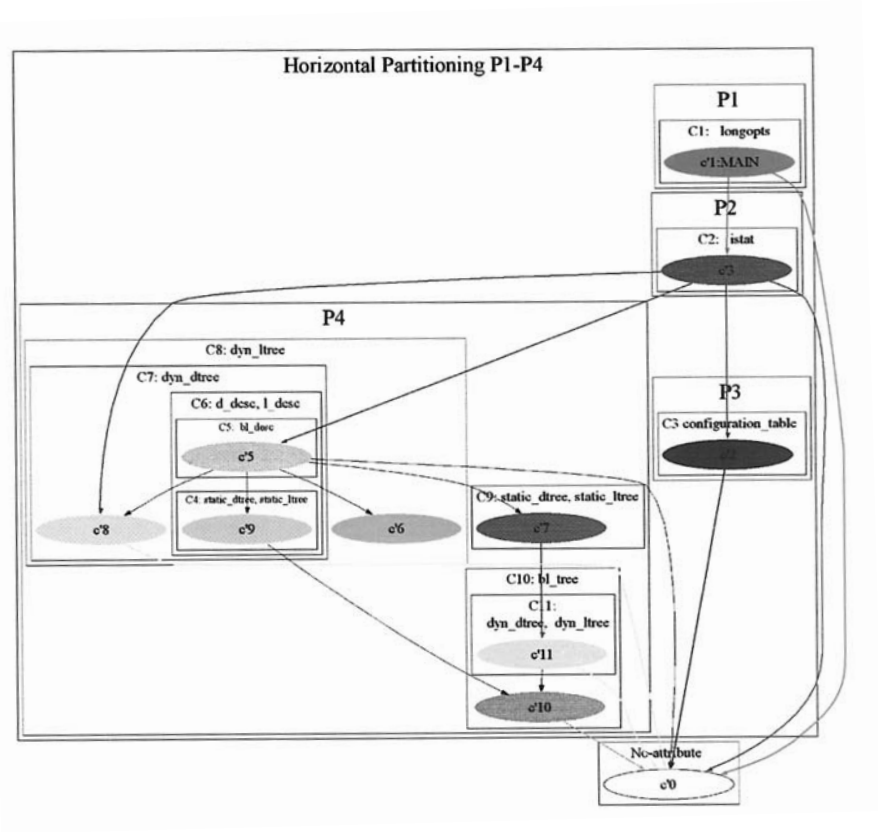

図 9: グルーピング結果を適用した gzip-1.2.4 の関数呼び出し図

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-Dec 1999.

[3] Bernhard Ganter, Rudolf Wille, "Formal Concept Analysis: Mathmatical Foundations", Springer, 1999.

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

[5] 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.

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

[7] 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.

[8] H.A. Müller, S.R. Tilley, M.A. Orgun, B.D. Corrie, and N.H. Madhavji, "A Reverse Engineering Environment Based on Spatial and Visual Software Interconnection Models", Proceedings of the Fifth ACM SIGSOFT Symposium on Software Development Environments, pp. 88–98, 1992.

[9] 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.

[10] Sapid, `http://www.sapid.org/`

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