部品の抽象仕様記述を用いた類似部品検索の実現

平野亮太[†] 田中譲[†,††]

部品の再利用と，組立て方式によるアプリケーション開発では，アプリケーションの仕様をいくつかの部分的な仕様に分解し，個々の部分的仕様を満たす部品を合成部品に組み立てることで，目的のアプリケーションを構築することができる．こうしたアプリケーション開発を支援するには，分解，再合成可能なアプリケーションの仕様記述法が必要である．また，大量のソフトウェア部品を管理し，仕様の類似性に基づく部品検索を実現する必要がある．本論では，著者らが提案した部品の抽象的な仕様記述を，部品の型記述として利用した類似部品検索の実現方式について述べる．部品の管理手法は，部品の型記述における半順序関係の定義を行い，その関係を表す一般化階層構造（束）を利用するものを提案する．また，ハッシュ法を用いることで，部品検索の高速化を図る．

Software Component Search Using Component-pattern Descriptions

Ryota Hirano[†] and Yuzuru Tanaka[†,††]

In our previous research, we introduced a component–pattern description method, based on IntelligentPad architecture, to describe the interface and the abstract behavior of each component. In this paper, we apply component–pattern descriptions to the search for desired components within a component library. The characteristics of component–pattern description method are the establishment of methods for composition and decomposition of patterns, and the use of the same form both for individual components and composites. Consequently, we can decompose a pattern into subpatterns, and later assemble the components that match those subpatterns to form composite results. For managing patterns, we propose a lattice structure representing the partial order over the patterns. We also introduce a coding method for pattern descriptions, and a hashing method for improving the efficiency of pattern search.

1. はじめに

部品の再利用と，組立て方式によるアプリケーション開発では[11),14)]，アプリケーションの仕様をいくつかの部分的な仕様に分解し，個々の部分的仕様を満たす部品を合成部品に組み立てることで，目的のアプリケーションを構築することができる．こうしたアプリケーション開発を支援するには，分解，再合成可能なアプリケーションの仕様記述法が必要である．また，大量のソフトウェア部品を管理し，仕様の類似性に基づく部品検索を実現する必要がある．本論では，部品の抽象的な仕様記述を，部品の型記述として利用した類似部品検索の実現方式について述べる．

本研究では，あるコンポーネントアーキテクチャを仮定し，そのもとで，組立てと機能連係のための統一的インタフェースを備えた，ソフトウェア構成の基本単位を部品と呼ぶ．部品のインタフェースを，他の部品とデータやシグナルの授受を行うための入出力口であるスロットの集合と定義する．部品の振舞いを，スロット間に存在する参照/更新の依存関係の集合として抽象的に記述する．この部品のインタフェースと部品の振舞いの対をパターン記述と呼ぶ．パターン記述法は，IntelligentPad[15),16)] アーキテクチャに基づいて，著者らが文献 17) で提案した部品の抽象仕様記述法である．部品のパターン記述は一意に定まるが，逆は必ずしも成り立たない．パターン記述が部品の振舞いを厳密に記述するわけではない．

パターン記述法には，以下の特徴があげられる．(1) 単一部品と合成部品に同じ記述形式を与える．このため，あるパターンにマッチする部品として，単一部品と合成部品の両方の検索が可能となる．(2) パターンのグラフ表現を利用したパターンの分解法が確立されている．この分解法は，パターンをいくつかの部分パターンに分解し，分解された各々の部分パターンを満たす部品を合成部品に組み立てることで，元々のパターンを満たす合成部品が組み立てられることを保証

† 北海道大学大学院工学研究科電子情報工学専攻
Graduate School of Engineering, Hokkaido University
†† 北海道大学知識メディアラボラトリー
Meme Media Laboratory, Hokkaido University

する[17]．(3) 半順序関係を定義できるので，パターンに基づく部分構造検索や，半順序関係を類似尺度とした類似部品の検索が可能となる．

クラスや関数の名前による部品のキーワード検索やファセット[9]による部品検索では，名前やキーワードの曖昧さが問題となり，また，部品の振舞いに基づく検索はできない．キーワードやファセットによって検索された部品に対し，さらに，形式仕様記述に基づいて，指定した振舞いを満たす部品のみを選び出す手法もある[5),13]．しかし，ユーザが形式仕様を記述することの難しさと，形式仕様の検査に要する計算コストが問題となる．パターン記述は，部品の振舞いを厳密に記述するわけではないが，部品やスロットの名前に依存せずに，部品の振舞いを抽象的に記述したシグネチャである．文献 12) では，Standard ML の関数やモジュールのシグネチャを用いた部品検索を実現しているが，単一部品や合成部品について，統一したシグネチャ記述を与えていないため，問合せに対して単一部品の検索しか実現できない．

パターンの管理方式には，パターンの半順序関係を表す一般化階層構造（束）を利用する．束による階層管理は，概念グラフ[6]や分子構造など，複雑な構造を持つデータの管理に利用されている[1),2]．この方式の特徴は，半順序関係に基づいて，データ間の類似距離とリンクを定義できることである．あるパターンを表す頂点の近隣に，類似するパターンを表す頂点が集まるため，ある頂点から次のリンクをたどる方向をユーザが決定することにより，探索型の部品検索を容易に実現することができる．

概念グラフや分子構造など，複雑な構造を持つデータを管理する場合，データ間の同型性の判定，準同型関係の判定に要する計算コストが問題となる．本論では，パターンのグラフ表現における，頂点の次数を利用したコーディングを提案し，コードをあらかじめ計算しておき，コードどうしの比較を行うことによって同型パターンであるか否かの判定を効率良く行う．また，束による探索型の部品検索では，問合せパターンについて，それに類似するものを，シグネチャファイル[3]などを用いたフィルタリングによって，すべて探し出すのではなく，問合せパターンと同型なパターンを表す頂点か，それが存在しないときは，問合せパターンの近傍となる頂点を高速に探し出す必要がある．本論では，ハッシュ法を利用して同型パターン検索の高速化を図る．

以下，2 章ではパターン記述法と，正規パターンの定義を行い，正規パターン間の半順序関係の定義とパターンの分解法についてその概要を述べる．3 章では束を用いたパターンの管理手法を示す．4 章では，ハッシュ法による同型パターン検索の高速化について述べ，ハッシュ法の評価を行う．5 章では，パターンのコーディング法を示し，コードを用いた同型判定により，同型パターンをどの程度判別できるかを評価する．6 章では本論が提案する部品検索機構のアプリケーション開発への応用例を示す．

2. 部品の型記述

2.1 パターン記述法

パターン記述法[17]では，部品のインタフェースを，他の部品とデータやシグナルのやりとりを行うためのスロットの集合と定義し，部品の振舞いをスロット間に存在する参照/更新の依存関係の集合によって抽象記述化する．ある部品 P のパターン記述を $ptrn(P) = \langle S, D\rangle$ で表す．S は部品 P のスロット集合を表し，各スロット $s \in S$ は名前とデータ型を持つ ($\#\mathsf{slotName} : \mathsf{type}$)．$D$ は依存関係集合を表す．各依存関係 $d \in D$ は，スロットアクセスされた 1 つのスロット $access \in S$ に対し，そのアクセスを契機に更新される 1 つのスロット $update \in S$ と，この更新時に参照される可能性のあるスロットの集合 $Ref \subseteq S$ との関係を，式 (1) の形式で記述する．

$$access \rightarrow update\ (Ref) \tag{1}$$

$access$ をアクセススロット，$update$ を更新スロット，Ref を参照スロット集合と呼ぶ．$access \neq update$ とする．アクセススロットは，更新スロットが更新される際に，参照可能と仮定し，$access \notin Ref$ とする．依存関係 $\#\mathsf{a}{\rightarrow}\#\mathsf{a}\ (Ref)$ は，新しいスロット名 $\#\mathsf{a}'$ を導入し，$\#\mathsf{a}{\rightarrow}\#\mathsf{a}'\ (Ref)$ と $\#\mathsf{a}'{\rightarrow}\#\mathsf{a}\ (Ref)$ の 2 つの依存関係で表す．

参照スロット集合 Ref を無視したとき，依存関係は推移的であるとする．$\#\mathsf{s}_1 \rightarrow \#\mathsf{s}_2$，$\#\mathsf{s}_2 \rightarrow \#\mathsf{s}_3$ が成り立つとき，$\#\mathsf{s}_1 \rightarrow \#\mathsf{s}_3$ も成り立つ．

2 つの部品 A，B の合成は，各々の部品が持つ 1 つずつのスロットを結合することで実現される．部品 A，B の $ptrn(A) = \langle S_A, D_A\rangle$，$ptrn(B) = \langle S_B, D_B\rangle$ について，各スロットの名前はユニークであり，重複はないものとする（$S_A \cap S_B = \phi$）．A のスロット $\#\mathsf{a}$ と，B のスロット $\#\mathsf{b}$ がスロット結合されたとき，合成部品 AB のパターン記述 $ptrn(AB)$ を式 (2) で表す．

$$ptrn(AB) = \langle S_A \cup S'_B,\ D_A \cup D'_B\rangle \tag{2}$$

式 (2) の S'_B，D'_B は $ptrn(B)$ におけるスロット $\#\mathsf{b}$ の名前を $\#\mathsf{a}$ に変更したものである．合成部品 AB は，A を台紙（親）にしたとき，子となる B の

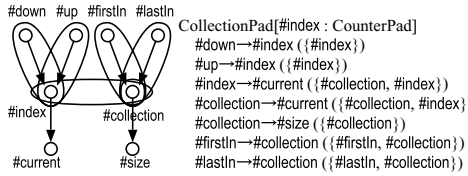

図 1 合成部品パターンの依存関係集合とパターングラフの例
Fig. 1 Dependencies for an example composite component, and its pattern graph.

結合スロット #b を省略し，$A[\#\mathsf{a} : B]$ と書く．結合スロットを示す必要がないときは $A[B]$ と書く．B を台紙としたときは $B[\#\mathsf{b} : A]$，もしくは $B[A]$ と書く．**図 1** 右に，合成部品の依存関係集合の例を示す．

2.2 パターングラフと正規パターン

パターン $p = \langle S, D\rangle$ から冗長な記述をなくすため，一意に定まるグラフ表現を作成する．この表現をパターングラフ $G(p)$ と呼び，以下の手順によって得る．

(1) 各依存関係の参照スロット集合を無視したアクセススロット–更新スロットの関係を半順序関係と見なし，与えられた依存関係集合 D に対応したハッセ図を作成する．推移性から明らかな依存関係は冗長である．

(2) 冗長な依存関係を除く各依存関係について，(1) で作成したハッセ図に参照スロットが存在していなければ，これを追加し，アクセススロットと参照スロットを囲む特殊辺を作成する．ただし，アクセススロット–更新スロットの関係が同一である複数の依存関係 $\#\mathsf{x} \rightarrow \#\mathsf{y}$ (R_1), $\#\mathsf{x} \rightarrow \#\mathsf{y}$ $(R_2), \cdots, \#\mathsf{x} \rightarrow \#\mathsf{y}$ (R_n) が存在するとき，これらを 1 つの依存関係 $\#\mathsf{x} \rightarrow \#\mathsf{y}$ $(R_1 \cup R_2 \cup \cdots \cup R_n)$ と見なした特殊辺を作成する．

(3) スロット集合 S 中に，(2) で拡張したハッセ図に出現していないスロットがあれば，これらを孤立点として加える．

図 1 左にパターングラフの例を示す．$G(p)$ は，連結な場合と，非連結な場合がある．非連結な場合，$G(p)$ には 2 つ以上の独立な連結成分が存在する．このようなパターンを持つ部品は，互いにスロット結合することなく，独立に存在するいくつかの部品の集合と等価であると考えられる．このため，本論では，対応するパターングラフが連結となるようなパターン記述を持つ部品のみを基本部品として考える．

定義（正規パターン） 上記の手順 (1)〜(3) によって得られるパターングラフ $G(p')$ が連結であるとき，パターン p' を正規パターンという．

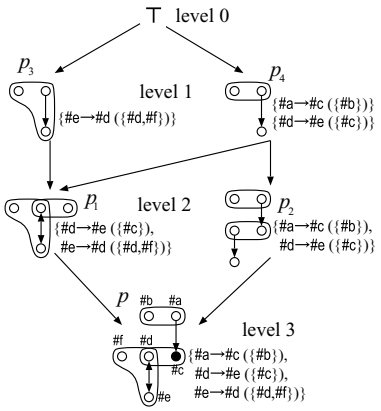

図 2 一般化階層の例
Fig. 2 A generalization hierarchy for a pattern.

2.3 正規パターンの半順序関係

定義（同型パターン） 正規パターン $p = \langle S, D\rangle$ について，S の各スロットの名前を変更する全単射写像 $f : S \rightarrow S'$ によって得られる $p' = \langle S', D'\rangle$ を p の同型パターンと呼び，$p' \equiv p$ と表す．

定義（サブパターン） 正規パターン $p' = \langle S', D'\rangle$, $p = \langle S, D\rangle$ について，S' の各スロットの名前を変更する単射写像 $g : S' \rightarrow S$ によって D' から得られる依存関係集合 D'' が，$D'' \subseteq D$ の関係を満たすとき，p' を p のサブパターンと呼び，$p' \subseteq p$ と表す．

定義（レベル） 正規パターン $p = \langle S, D\rangle$ が持つ依存関係集合 D の要素数を p のレベル $|p|$ という．

2.4 正規パターンの一般化階層

正規パターン p について，そのすべてのサブパターンを正規化して集めた集合を $\beta(p)$ で表すと，$\beta(p)$ は半順序集合となる．この関係を図示したハッセ図を $\beta(p)$ の一般化階層と呼ぶ．$\beta(p)$ は，スロットと依存関係を持たないレベル 0 のパターン $\top$ を含むものとする．$\top$ は任意のパターン w に対して，$\top \subseteq w$ の関係を満たす．$\beta(p)$ の一般化階層では，関係 $\subseteq$ の意味で小さいものを上方に位置する．$u, v \in \beta(p)$ について，$u \subseteq v$ ならば，u は v の上方に位置する．$u, v \in \beta(p)$ について，$u \subseteq v$ かつ，そのレベル差が $|v| - |u| = 1$ のとき，u から v への有向辺が存在する．このとき，u を親パターン，v を子パターンと呼ぶ．**図 2** に一般化階層の例を示す．図 2 では，パターングラフと依存関係集合を各頂点に示している．

ある頂点 $u \in \beta(p)$ の先祖，親，子，子孫のパターンを表す集合は，式 (3)〜(6) のように定義できる．

$$ancestors(u, \beta(p)) = \{w \mid w \in \beta(p) \ \wedge \ w \subseteq u\} \qquad (3)$$

$$parents(u, \beta(p)) = \{w \mid w \in \beta(p) \ \wedge$$

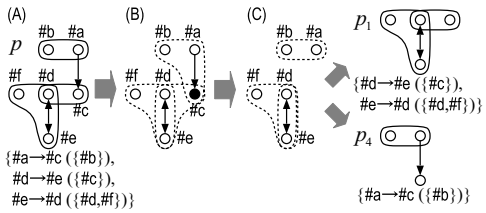

図 3 拡張関節点によるパターンの分解例
Fig. 3 An example decomposition of a pattern graph.

$$w \subseteq u \ \wedge \ |w| = |u| - 1\} \quad (4)$$

$$descendants(u, \beta(p)) = \{w \mid w \in \beta(p) \ \wedge \ u \subseteq w\} \quad (5)$$

$$children(u, \beta(p)) = \{w \mid w \in \beta(p) \ \wedge \ u \subseteq w \ \wedge \ |w| = |u| + 1\} \quad (6)$$

2.5 パターンの分解法

パターングラフ $G(p)$ にある分解点が存在すれば，パターン p を 2 つのサブパターン p_1，p_2 に分解することができる．この分解点を拡張関節点と呼ぶ[17]．拡張関節点は，パターングラフにおける各依存関係の特殊辺を，アクセススロットと参照スロットに加えて，更新スロットをも取り囲む拡張特殊辺とする．このとき，ある頂点とその頂点に接続する有向辺の除去に加えて，その頂点を取り囲む拡張特殊辺からその頂点のみを除去することで，グラフが非連結となる頂点である．$G(p_1)$，$G(p_2)$ にも拡張関節点が存在すれば，これらも同様に分解することができる．

図 3 に拡張関節点の例を示す．図 3 (B) の各々の破線は，図 3 (A) の特殊辺を拡張した拡張特殊辺である．頂点 #c と，#c に接続する有向辺を除去し，#c を囲む拡張特殊辺から #c を除去することで，グラフは非連結となる（図 3 (C)）．つまり，#c は拡張関節点であり，p を p_1，p_4 の 2 つのサブパターンに分解できる．このため，パターン p を持つ部品が存在しなくても，サブパターン p_1，p_4 を持つ部品 P_1，P_4 が存在すれば，合成部品 $P_1[P_4]$，$P_4[P_1]$ を組み立てることができ，これらの合成部品のパターンは p となる．

3. 類似部品検索

本章では，一般化階層を用いた正規パターンの検索，登録手法を示す．以降では，正規パターンを対象に議論を進めることにし，特に明記しない限り，パターンはすべて正規パターンであると仮定する．

3.1 Pattern Lattice

1 つのパターンだけでなく，複数のパターンの一般化階層を含んだ階層構造を考える．これを Pattern Lattice と呼ぶ（以降 PL と省略）．**図 4** に PL の概要を示す．β を PL に存在する頂点集合とする．各頂点 $p \in \beta$ は，パターン p を持つ部品が格納されているファイルへのポインタの集合を持つものとする．

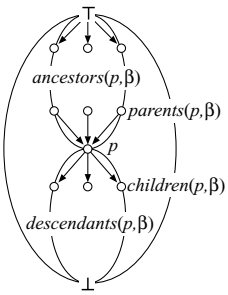

図 4 Pattern Lattice の概要
Fig. 4 An overview of the Pattern Lattice.

$\perp \in \beta$ は任意のパターン w について $w \subseteq \perp$ を満たすパターンとする．ある $p \in \beta$ について，その先祖集合，親集合，子集合，子孫集合は，式 (3)〜(6) における $\beta(p)$ を β とすることで同様に定義できる．PL では，$p \in \beta$ ならば，$\beta(p)$ の一般化階層を含むため，$ancestors(p, \beta) \subseteq \beta$ である．$p \notin \beta$ ならば，p の子や子孫は存在せず，$children(p, \beta) = \phi$，$descendants(p, \beta) = \phi$ である．

$p \in \beta$ から $ancestors(p, \beta)$ や $descendants(p, \beta)$ の空間を探索することで，p のサブパターンを持つ部品や，p をサブパターンとするパターンを持つ部品を検索することができる．

3.2 パターンの検索

問合せパターン q の検索は，PL における q の同型パターンを探し出す．同型パターンが存在しないとき（$q \notin \beta$），PL では $descendants(q, \beta) = \phi$ が成り立つため，PL 中に存在する q の先祖集合 $ancestors(q, \beta)$ を検索し，その中から $\beta(q)$ において q からのパスの長さが短い極小元を求める．これを q の**近傍パターン集合** IA（Immediate Ancestors）と呼ぶ．

文献 1)，2) では，一般化階層構造（本論では PL）を頂点 $\top$ から探索することで，同様の目的を果たすアルゴリズムが示されている．しかし，本論では，1 つのパターンについて，その同型パターンの検索を，PL の頂点を探索せずに高速に実現することを目指す．このため，q から $\beta(q)$ の一般化階層を生成し，頂点 q から有向辺を逆にたどる幅優先探索を行うことで，q の近傍パターン集合 IA を求める．この方法では，探索空間は $\beta(q)$ の頂点数 $|\beta(q)|$ 以下に限定され，PL の頂点数 $|\beta|$ には影響されない．

問合せパターン q の近傍パターン集合 IA を求める検索アルゴリズムを**図 5** に示す．$\beta(q)$ の一般化階

```
function getIA(q)
  Q.create(); Q.enqueue(q);
  IA := {} ;
  while not Q.empty() do
    v := Q.dequeue();
    if 同型パターン v の検索に成功 then
      if {w| w ∈ IA ∧ v ⊆ w} = φ then
        IA := IA + v;
      end;
    else  // 検索に失敗
      foreach w ∈ parents(v, β(q)) do
        Q.enqueue(w);
      end;
    end;
  end;
  return IA;
end;
```

図 5 問合せパターンの検索アルゴリズム
Fig. 5 Searching algorithm for a query pattern.

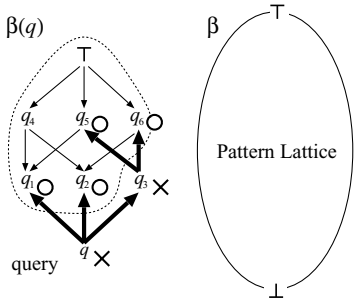

図 6 問合せパターンの検索例（$IA = \{q_1, q_2\}$）
Fig. 6 Searching for an example query pattern q.

層の有向辺を q から逆にたどる幅優先探索では，ある頂点 $v \in \beta(q)$ の訪問は，v の PL における同型パターン検索を行うことによって実施される．検索に失敗した場合，v に隣接する頂点 $parents(v, \beta(q))$ を訪問する．検索に成功した場合，隣接する頂点の訪問は行わずに，v のサブパターンが IA に含まれていない場合にのみ（$\{w \mid w \in IA \ \wedge \ v \subseteq w\} = \phi\}$），$v$ を IA に追加する．

図 6 に検索例を示す．この例では，問合せパターン q の PL における同型パターン検索に失敗し（図中×），隣接する q_1，q_2，q_3 が順に訪問される．q_1，q_2 の PL における同型パターン検索に成功し（図中○），$IA = \{q_1, q_2\}$ となるが，q_3 の同型パターン検索は失敗する（図中×）．次に，q_3 に隣接する q_5，q_6 が訪問され，それらの同型パターン検索に成功するが（図中○），$q_1, q_2 \in IA$ に対し，$q_5 \subseteq q_1$，$q_6 \subseteq q_2$ が成り立つため，$IA = \{q_1, q_2\}$ のまま探索が終了する．図 6 の波線で囲まれたパターンの集合が，PL 中に存在する q の先祖集合であり，その極小元 $\{q_1, q_2\}$ が近傍パターン集合になっている．

```
procedure regist(q)
  Q.create(); Q.enqueue(q);
  while not Q.empty() do
    v := Q.dequeue();
    if 同型パターン v の検索に失敗 then
      頂点 v を PL に追加;
    end;
    foreach w ∈ children(v, β(q)) do
      if 有向辺 (v, w) が PL に存在しない then
        有向辺 (v, w) を PL に追加;
      end;
    end;
    foreach w ∈ parents(v, β(q)) do
      Q.enqueue(w);
    end;
  end;
end;
```

図 7 問合せパターンの登録アルゴリズム
Fig. 7 An algorithm for inserting a query pattern.

探索すべき頂点数は，$\beta(q)$ のすべての頂点を訪問するときに最大となる．レベル $k = |q|$ における頂点数 $|\beta(q)|$ は，各頂点のパターンの依存関係集合の集合全体が，q の依存関係集合の巾集合となるときに最大 2^k となる．このため，検索のコストは k の指数関数オーダになるが，PL の頂点数 $|\beta|$ には依存しない．

3.3 パターンの登録

パターン q を持つ部品 Q の登録は，$\beta(q)$ の一般化階層について，PL 中に存在していない頂点と有向辺があれば，それらを PL に追加する．登録アルゴリズムを**図 7** に示す．3.2 節の検索アルゴリズムと同様に，$\beta(q)$ の一般化階層の q から幅優先探索を行う．ある頂点 $v \in \beta(q)$ の訪問は，v の PL における同型パターン検索を行う．検索に失敗した場合，頂点 v を PL に追加する．次に，v から子 $w \in children(v, \beta(q))$ への有向辺 (v, w) に相当するものが PL 中に存在しなければ，有向辺 (v, w) を PL に追加する．

登録アルゴリズムは，検索アルゴリズムとは異なり，$\beta(q)$ のすべての頂点を訪問する．これは，部品 Q と，そのすべての部分部品が格納されているファイルへのポインタを，PL 中の対応するパターンを表す頂点のポインタ集合に追加するためである．図 6 の $\beta(q)$ の一般化階層を登録する場合，検索アルゴリズムでは PL における同型パターン検索に失敗した頂点とその頂点の親のみしか訪問しないが，登録アルゴリズムではすべての頂点が訪問される．登録アルゴリズムでは，探索において，同型パターン検索に失敗した頂点を PL に追加し，追加された頂点とその親を結ぶ有向辺を PL に追加する．図 6 の例では，PL における同型パターン検索に失敗した頂点（図中×）と，図中の太線の有向辺のみが PL に追加される．

PL に登録すべき頂点数は，$\beta(q)$ のすべての頂点を

訪問するときに最大となる．レベル $k = |q|$ における頂点数 $|\beta(q)|$ の最大値は 2^k であり，有向辺の数の最大値は $k \cdot 2^{k-1}$ となる．このため，登録のコストは検索の場合と同様に k の指数関数オーダになるが，PL の頂点数 $|\beta|$ には依存しない．

4. 同型パターン検索の高速化

本章では，3 章で示した PL による類似部品検索について，ハッシュ法を利用することで，1 つの同型パターン検索を高速に行う方法を述べる．

4.1 パターンのレコード表現

PL における，1 つのパターン p を表す頂点を，1 つのレコード $rec(p)$ で表す．$rec(p)$ は，以下の 4 つのフィールドを持つ．

$$rec(p) = \langle code(p),\ PC,\ CC,\ L \rangle \quad (7)$$

$code(p)$ はパターン p から生成されるコードであり，レコード $rec(p)$ のキーである．$code(p)$ は 128-bit 固定長とし，その生成法は 5 章で示す．PC は PL における $parents(p, \beta)$ に含まれるすべての親パターンに対して，そのコード表現を集めた親コード集合とする．CC は $children(p, \beta)$ に含まれるすべての子パターンに対してそのコード表現を集めた子コード集合とする．L はパターン p を持つ部品が格納されているファイルを指すポインタ集合とする．

4.2 ファイル構造とレコードの検索

PL の頂点集合 β をパターンのレベルごとに分類し，各レベルにハッシュ表を用意する．ハッシュ関数の値が同一になるレコードの管理は，チェイン法を用いて，連結リストでつなぐ．ハッシュ表の 1 つのアドレスには，連結リストの先頭を指すポインタを格納する．

図 8 にファイル構造を示す．各ハッシュ表のサイズを N 項目，ハッシュ関数を $h(k)$ とする．パターン p のレコード $rec(p)$ の検索は，レベル $|p|$ に対応するハッシュ表を選択し（図 8 (1)），キー $k = code(p)$ のハッシュ値 $h(k)$ を計算する（図 8 (2)）．選択したハッシュ表のアドレス $h(k)$ から連結リストを選び出し（図 8 (3)），連結リストの線形探索を行うことでキー $k = code(p)$ を持つレコードを探し出す．

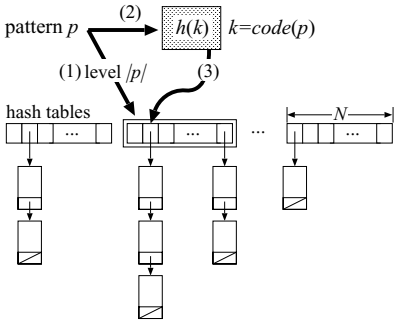

図 8 ファイル構造
Fig. 8 The hash file structures.

4.3 レコードの追加と更新

レコードの追加は，3.3 節で示したパターン q の登録アルゴリズムにおいて，ある頂点 $v \in \beta(q)$ のレコード $rec(v)$ の検索に失敗したときに行われる．このとき，$\beta(q)$ における v の親集合と子集合からレコード $rec(v)$ を新規に生成し，検索失敗時に選択した連結リストの末尾に $rec(v)$ を連結する．

レコードの更新は，$\beta(q)$ における v から $w \in children(v, \beta(q))$ への有向辺 (v, w) に相当するものが PL 中に存在していないときに行われる．このとき，$rec(v)$ の子コード集合に $code(w)$ を追加し，$rec(v)$ を更新する．また，v に対応する部分部品が存在し，その部分部品へのポインタが $rec(v)$ のポインタ集合に含まれていないとき，そのポインタを $rec(v)$ のポインタ集合に追加し，$rec(v)$ を更新する．

4.4 検索コスト

サイズ N のハッシュ表に r 個のレコードを格納したとき，チェイン法の場合，レコードの連結リストの長さの平均は r/N である．連結リスト上の線形探索の際に調べるレコードの数は，連結リストの長さに比例するため，検索の計算量が比 r/N だけに依存するのが普通である．この比 r/N はハッシュ表の占有率（load factor）と呼ばれる．

ハッシュ表のサイズを固定し（$N = 2^{16}$），異なる占有率のもとで，ハッシュ値が同一になるレコードの連結リスト長 m と，連結リスト長が m であるハッシュ表のアドレスの数 $addr(m)$ の分布を調べると，**図 9** 左に示すようになった．ハッシュ関数は 4.5 節で示すものを利用した．この分布から，異なる占有率での，レコードの平均探索長を求める．長さが m の連結リスト上の線形探索では，探索長の合計は $1 + 2 + \cdots + m = m(m+1)/2$ となる．また，このようなアドレスは $addr(m)$ 個存在するため，平均探索長は式 (8) で求めることができる．

$$\text{平均探索長} = \frac{1}{r} \sum_{m=1}^{M} \frac{m(m+1)}{2} addr(m) \quad (8)$$

式 (8) の M は，図 9 の各分布での最大連結リスト長を表す．式 (8) で求めた平均探索長と，占有率との関係を図 9 右に示す．平均探索長–占有率の関係は線形となっており，平均探索長が占有率 r/N に比例して大きくなることが確認できる．

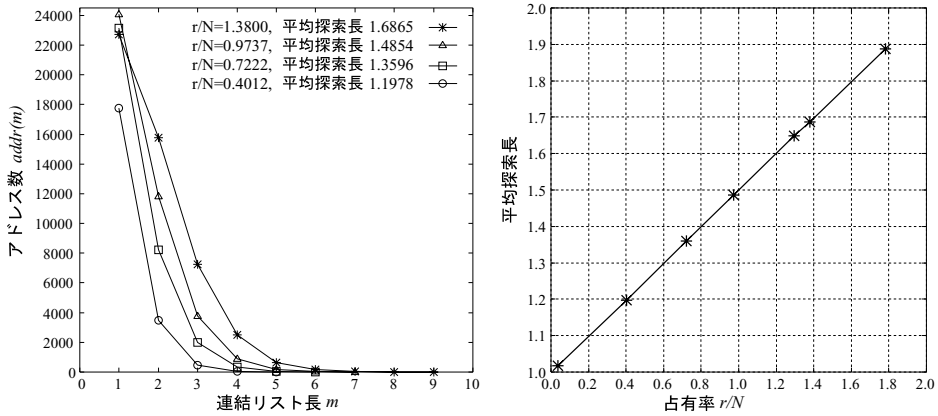

図 9 異なる占有率（$N = 2^{16} = 65536$）での連結リスト長 m と，そのアドレス数の分布（左）と，占有率と平均探索長の関係（右）

Fig. 9 The distribution of length of connected list and number of such lists for various load factors (left). The relation between load factors and average search lengths (right).

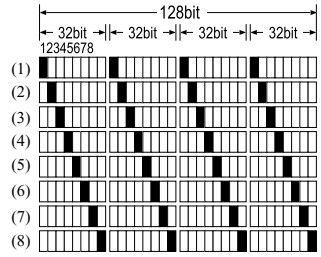

図 10 8 種類のハッシュ関数

Fig. 10 Some hash functions that output a 16-bit code from a 128-bit pattern code.

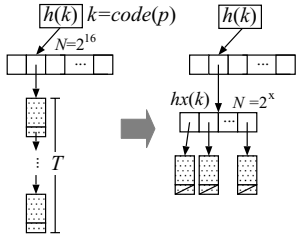

図 11 空間数の局所的な拡張方法の概要

Fig. 11 The local extension method for hash tables.

4.5 ハッシュ関数の選択

ハッシュ関数は，ハッシュ表のサイズ $N = 2^{16}$ に対し，128-bit 長のキー $k = code(p)$ から，16-bit の数値形式のキーを生成する．**図 10** に示すように，128-bit 長のキーを 32-bit の 4 つのブロックに分割し，各ブロックをさらに 4-bit の 8 つのサブブロックに分割する．各ブロックから 1 つの 4-bit のサブブロックを選び，それら 4 つを連結したものをハッシュ値とする．4-bit のサブブロックの選択方法を変えた 8 種類のハッシュ関数について，4.4 節の図 9 で示したような，連結リスト長 m と，連結リスト長が m であるハッシュ表のアドレスの数 $addr(m)$ の分布を調べたが，8 種類のハッシュ関数は同じような分布となり，大きな差は見いだせなかった．このため，図 10 の (8) に示す選択法をハッシュ関数として利用した．

4.6 ハッシュ表の容量の拡張

4.4 節でレコードの連結リスト長の平均値 r/N が増えると，これに線形的に比例して，平均探索長が大きくなることを示した．このような場合，平均探索長を減らすために，連結リストの線形探索にインデクスファイルを利用した二分探索を利用する方法と，ハッシュ表の空間数を拡張し，すべてのレコードを再ハッシュする方法が考えられるが，本論では，以下に示す局所的な容量拡張法を提案する．

図 11 に示すように，ハッシュ表のあるアドレスの連結リスト長がしきい値 T となったとき，そのアドレス用にサイズ $N' = 2^x$ の二次ハッシュ表と，x-bit の二次ハッシュ関数 $hx(k)$ を用意して，T 個のレコードを二次ハッシュ表に再ハッシュする．$hx(k)$ が出力する x-bit の値は，128-bit 長のコード $k = code(p)$ について，4.5 節で示した $h(k)$ が出力する 16-bit 以外のビットとする．このため，T 個のレコードは，$16 + x$-bit の空間にハッシュされることになる．

二次ハッシュ表において，あるアドレスのレコードの連結リスト長がしきい値を越えた場合，そのアドレス用に別のハッシュ表を用意するのではなく，二次ハッシュ表のサイズを $2^{x'}$ $(x' > x)$ に再拡張し，二次ハッ

シュ関数 $hx'(k)$ を用いて，二次ハッシュ表のすべてのレコードを再ハッシュする．

5. パターンのコーディング

同型パターンの判定には，2.3 節で示したように，パターンのスロット集合について，各スロットの名前を変更する全単射写像を求める必要がある．本章では，同型パターンであるか否かの判定を効率良く行うため，パターンのコーディング法を提案し，コードの一致，不一致による同型パターン判定を実現する．このため，いくつかのコーディング法を提案し，コードを用いた同型パターン判定の評価を行った．

5.1 スロットの指標

パターン $p = \langle S, D\rangle$ のパターングラフ $G(p)$ における各頂点 $s \in S$ について，以下の 3 つの頂点 s の次数に注目する．

- Ad ··· 頂点 s から出力する有向辺数（出力次数）
- Ud ··· 頂点 s に入力する有向辺数（入力次数）
- Rd ··· 頂点 s を包囲する特殊辺数（包囲次数）

この 3 つの次数からなる各スロット $s \in S$ の指標を $w(s) = \langle Ad, Ud, Rd\rangle$ で表す．

5.2 パターンコード

$p = \langle S, D\rangle$ について，$G(p)$ から得られる各スロットの指標と，p の依存関係集合 D に注目する．

単一の依存関係 $d \in D$ について，d だけを依存関係集合に持ち，d に出現するスロットだけをスロット集合に持つパターンを考える．このパターングラフは，**図 12** に示すように，スロット数 x のとき，特殊辺が囲むスロット数 y が，$y = x$ か $y = x - 1$ の 2 種類しか存在しない．このため，単一の依存関係 d の種類を表す識別子を $id(d) = \langle x, y\rangle$ と定義する．

各依存関係 $d \in D$ について，各スロットの指標と，識別子 $id(d)$ から，式 (9) の形式のコードを生成する．

$$[t(d)]\{w(s_1), w(s_2), \{w(s_3), \cdots, w(s_x)\}\} \quad (9)$$

式 (9) の $w(s_1)$ は $d \in D$ のアクセススロット s_1 の指標であり，$w(s_2)$ は更新スロット s_2 の指標である．$\{w(s_3), \cdots, w(s_x)\}$ は，各参照スロットの指標を次数の組と見なし，それらを辞書式順序で並べた参照スロットの指標の集合である．$t(d)$ は識別子 $id(d) = \langle x, y\rangle$ に対し，値 $t(d) = x - y$ を表す．その値は 0 か 1 である．式 (9) では，その指標の数からスロット数 x が分かるため，$t(d) = x - y$ によって依存関係 d の種類の相違を明らかにする．

各 $d \in D$ について得られる式 (9) の形式のコードを文字列と見なし，それらをソートして連結したものを p のパターンコード $SC(p)$ と定義する．**図 13** にパターンコードの例を示す．

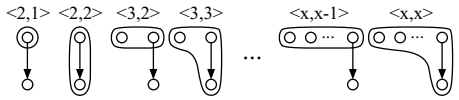

図 12 単一の依存関係の識別子
Fig. 12 Identifiers for a single dependency.

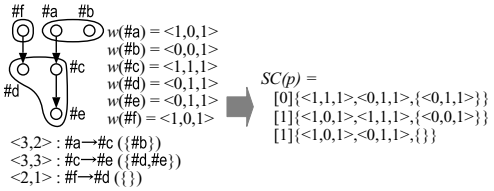

図 13 パターンコードの例
Fig. 13 An example pattern code.

表 1 平均縮退率の評価結果
Table 1 Average overlap ratios in coding patterns by different methods.

$\|P\|$	$\|C_{SC}\|$	Ω_{SC}	$\|C_{HC}\|$	Ω_{HC}
27	27	1.000	27	1.000
114	114	1.000	114	1.000
477	453	1.053	477	1.000
2108	1809	1.165	2103	1.002
9590	7131	1.345	9528	1.007
45003	27771	1.621	44487	1.012
215733	106653	2.023	212019	1.018

5.3 階層コード

パターン p について，$\beta(p)$ の一般化階層を p を特徴づける属性と見なし，$\top$ を除く $\beta(p)$ の各要素について，5.2 節で定義したパターンコードを生成した集合 $\{SC(u) \mid u \in \beta(p) \wedge u \neq \top\}$ を考える．この集合の各要素であるパターンコードを，パターンのレベルの降順に並べ，かつ，同じパターンレベルのパターンコードは，それらを文字列と見なしてソートしたものを，パターン p の階層コード $HC(p)$ と定義する．

5.4 コードによる同型パターン判定の評価

識別子 $\langle 3, 2\rangle$ の 1 種類の単一依存関係をランダムにスロット結合させて生成したパターンの集合 P の各要素についてコード生成を行う．P の要素数 $|P|$ と，生成されたコードの種類数から，1 つのコードに何個のパターンが縮退されるか，その平均値を求める．この平均値を平均縮退度と呼び，式 (10) で得られる．

$$\text{平均縮退率 } \Omega = |P|/|C| \quad (10)$$

式 (10) の C は，P の各要素についてコードを生成したコードの集合とし，C は互いに素なコードの集合とする．その要素数 $|C|$ はコードの種類数を表す．**表 1** に評価結果を示す．表 1 の Ω_{SC} はパターンコー

ドを用いた場合の平均縮退度，Ω_{HC} は階層コードを用いたときの平均縮退率である．

平均縮退度は，同型パターンか否かを識別するパターン数 $|P|$ が多くなるほど増加する．Ω_{SC} では急激に増加し，$|P| = 215733$ のとき，1 つのコードに平均 2.023 個の互いに非同型なパターンが縮退する．一方，Ω_{HC} では，平均縮退度は緩やかに増加し，1.0 に近い値のままである．この結果から，パターンの同型判定に階層コードを用いることができると判断できる．

各コード $SC(p)$，$HC(p)$ を文字列で表現した場合，特に階層コード $HC(p)$ は，コードのサイズが問題となる．たとえば，レベル数が 9 の階層コードのサイズは平均 4 kbytes となる．そこで，$HC(p)$ に一方向性ハッシュ関数である MD5 [10] を適用し，128-bit 固定長への圧縮変換したものをパターン p のコード $code(p)$ とする．MD5 を適用した場合でも，平均縮退率の評価結果は表 1 と同じになることを確認した．

6. アプリケーション開発への応用

開発したいアプリケーションの仕様がパターン q で記述されているとする．このとき，PL において q の同型パターン検索に失敗しても，2.5 節で示したパターンの分解法を用いることで，別の解決策を考えることができる．**図 14** に，ある問合せパターン q の $\beta(q)$ の一般化階層の例を示す．図 14 では，各頂点にパターングラフを示している．破線の矩形は，パターングラフ $G(q)$ に存在する拡張関節点を用いて q を分解して得られる q のサブパターンを表す．パターン q を持つ部品が PL に存在しなくても，$\{q_1, q_5\}$ を実現する部品が存在すれば，q を満たす合成部品を組み立てることができる．$\{q_2, q_4\}$，$\{q_3, q_4, q_5\}$ のサブパターンへの分解の仕方においても同様である．

q を満たすサブパターンへの分解の仕方の選択と，各サブパターンを実現する部品の選択は，ユーザが行う．部品ライブラリの中から，各サブパターン q' を実現する部品を選択するには，q' と同型パターンを持つ部品だけでなく，部品ライブラリの PL において，$ancestors(q', \beta)$ や $descendants(q', \beta)$ の空間を探索すればよい．見つからない場合，サブパターンへの分解の仕方を変更して，再度，探索すればよい．パターン記述法では，単一部品と合成部品のパターンが同じ形式で記述されるため，q' を実現する部品は，単一部品と合成部品の両方の場合がある．

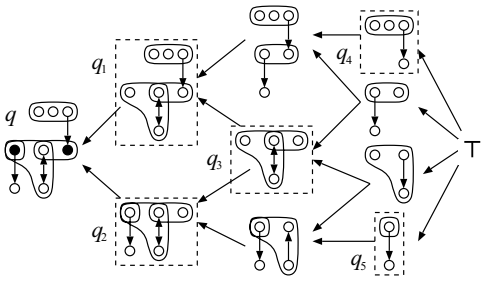

図 14 一般化階層とサブパターンへの分解の例
Fig. 14 An example generalization hierarchy for a pattern, and its decomposition into subpatterns.

7. おわりに

本論では，IntelligentPad アーキテクチャに基づいて著者らが提案した部品のパターン記述法に対して，この記述法に基づく類似部品検索の実現方式と，ハッシュ法を用いた部品検索の高速化について述べた．また，コーディング法を用いた同型パターン判定について，評価を行い，その実用性を示した．

本論が示す部品検索機構の応用に，ある部品 X と頻繁に組み合わされて再利用される部品を発見することが考えられる．部品 X と X を部分部品とする合成部品 $Y = X[Z]$ では，そのパターンの間に $ptrn(X) \subseteq ptrn(Y)$ の関係が成り立つ．このとき，PL では $ptrn(X)$ から $ptrn(Y)$ へのパスが存在し，$ptrn(X)$ から，$descendants(ptrn(X), \beta)$ の空間を探索することで，部品 Y を探し出し，X と組み合わされて再利用される部品 Z を見つけることができる．文献 7)，8) では，クラスライブラリやフレームワークの再利用における，クラスの継承やメンバ関数の再定義の関係を連想規則で記述し，統計的尺度から連想規則のサポートと信頼性を評価することで，クラスの再利用パターンを示している．このように，部品 X と組み合わされて頻繁に再利用される部品の発見について，統計的尺度から評価することも考えられる．

5 章で提案したパターンのコーディング法は，スロット型を考慮していないため，厳密な意味での同型判定ではない．スロットの型を考慮したコーディング法の改善は今後の課題である．

参考文献

1) Ellis, G.: Compiling Conceptual Graphs, *IEEE Trans. Knowledge and Data Engineering*, Vol.7, No.1 (1995).
2) Ellis, G. and Lehmann, F.: Exploiting the Induced Order on Type-Labeled Graphs for Fast Knowledge Retrieval, *Proc. 2nd International Conference on Conceptual Structures*, pp.293–310 (1994).
3) Falouts, C. and Christodoulakis, S.: Signature

Files: An Access Method for Documents and Its Analytical Performance Evaluation, *ACM Trans. Office Information Systems*, Vol.2, No.4 (1984).
4) Isakowitz, T. and Kauffman, R.J.: Supporting Search for Reusable Software Objects, *IEEE Trans. SE.*, Vol.22, No.6, pp.407–423 (1996).
5) Jeng, J.-J. and Cheng, B.H.: Specification Matching for Software Reuse: A Foundation, *Proc. ACM Symposium on Software Reuse*, pp.97–105 (1995).
6) Sowa, J.F.: *Conceptual Structures: Information Processing in Mind and Machine*, Addison-Wesley (1984).
7) Michail, A.: Data Mining Library Reuse Patterns in User-Selected Applications, *14th IEEE International Conference on Automated Software Engineering* (1999).
8) Michail, A. and Notkin, D.: Assessing Software Libraries by Browsing Similar Classes, Functions, and Relationships, *21st International Conference on Software Engineering* (1999).
9) Prieto-Diaz, R. and Freeman, P.: Classifying Software for Reusability, *IEEE Software*, Vol.4, No.1, pp.6–16 (1987).
10) Rivest, R.: The MD5 message Digest Algorithm, RFC 1321, MIT and RSA Data Security, Inc. (1992).
11) Udell, J.: Componentware, *Byte*, Vol.19, No.5, pp.46–56 (1994).
12) Zaremski, A.M. and Wing, J.M.: Signature Matching, a Tool for Using Software Libraries, *ACM Trans. Software Engineering and Methodology*, Vol.4, No.2 (1995).
13) Zaremski, A.M. and Wing, J.M.: Specification Matching of Software Components, *ACM Trans. Software Engineering and Methodology*, Vol.6, No.4, pp.333–369 (1997).
14) 青山幹雄：コンポーネントウェア：部品組み立て型ソフトウェア開発技術，情報処理，Vol.37, No.1, pp.71–79 (1996).
15) 田中 譲：ミームメディア・アーキテクチャIntelligentPadとその応用，情報処理学会誌，Vol.38, No.3, pp.222–231 (1997).
16) 長崎 祥，田中 譲：シンセティック・メディアシステム：IntelligentPad，コンピュータソフトウェア，Vol.11, No.1, pp.36–48 (1994).
17) 平野亮太，田中 譲：IntelligentPadにおける部品の抽象的仕様記述と開発への応用，情報処理学会論文誌，Vol.40, No.6, pp.2799–2809 (1999).

(平成 12 年 11 月 13 日受付)
(平成 13 年 3 月 9 日採録)

平野 亮太（正会員）

1971 年生．1994 年北海道大学工学部電気工学科卒業．1996 年同大学院電気工学専攻修士課程修了．現在，同大学院工学研究科電子情報工学専攻博士後期課程に在学中．ソフトウェア開発手法，ソフトウェア再利用技術等の研究に従事．

田中 譲（正会員）

1972 年京都大学工学部電気工学科卒業．1974 年同大学院電子工学専攻修士課程修了．工学博士．1974 年北海道大学工学部助手．1977 年同講師．1985 年同助教授を経て，1990 年同教授，現在に至る．1996 年北海道大学知識メディアラボラトリー長．この間，1985 年 10 月より 1 年間，IBM 社 T.J. ワトソン研究所客員研究員．ソフトウェア科学会，人工知能学会，米国 IEEE 各会員．データベース理論，データベース・マシン，並列処理アーキテクチャ，メディア・アーキテクチャ等の研究に従事．「コンピュータ・アーキテクチャ」（雨宮氏との共著，オーム社）等の著書あり．1994 年に IntelligentPad の開発に関して日経 BP 技術賞大賞受賞．