形式的仕様を用いた再利用モデル

川北 誠[†] 酒井正彦[††]
山本晋一郎[†] 阿草清滋[†]

ソフトウェアの需要が高まるにつれて，ソフトウェアの信頼性や生産性の向上が要求されている．生産性を向上させる手段の一つとしてプログラミングの自動化がある．これは形式的な仕様からプログラムを自動生成する方法である．しかし，プログラムの効率を考えたときに必ずしも最良のプログラムが得られるわけではない．そこで既存の効率の良いソフトウェアを再利用する方法が考えられる．これは既存のソフトウェアの中から再利用可能なソフトウェアを検索し，変更を加えて，望むソフトウェアを得る方法である．本論文では，部品の検索を仕様の類似性に基づいて行う方法について述べる．また要求に応じて変更するために既存の部品の変更点を求めることも，仕様に基づいて行う．これらのことを仕様に基づいて行うには，仕様を形式的に扱うことが必要となる．本論文では記述性，読解性に優れるという観点から，等式の集合で仕様を記述する代数的仕様を用いる．検索のために代数的仕様の類似性を定義する．類似性は等式を抽象化することによって得られる順序として与える．定義した類似性に基づく部品検索システムと検索結果について述べる．また，検索された部品を再利用する方法を具体例で検討する．新しく得られたプログラムのうち60%程度既存の部品を再利用した例を示す．

A Model for Reuse Based on Formal Specifications

MAKOTO KAWAKITA,[†] MASAHIKO SAKAI,[††] SHINICHIROU YAMAMOTO [†]
and KIYOSHI AGUSA [†]

As software demands increase, more efficient and reliable software development methods have been required. Automatic programming is one of the candidates. However it cannot always produce an expected program from the view point of performance. One of alternative approaches is software reusing. Reuse of a high performance software may provide a required software of high performance. There are two problems in reusing; retrieving and customization. Since reusing in upper stream increases reusability of software products, our reusing targets reusing of requirements specification. We adopt an algebraic specification because of its abstractness and formality. We define the similarity of algebraic specifications for retrieving. The similarity is expressed by orders with *abstract equations*. With several examples, we get a new program which is retrieved from the database and customized. The reusability is 60%, that is, about 60% part of derived programs is reused one.

1. はじめに

ソフトウェア開発の効率化のために，既存のソフトウェアを再利用する方法が研究されている[3]．再利用の対象は主としてソースコードであるが，抽象度の高い段階での再利用を考えるとソフトウェア開発の上流工程における再利用が必要となる[4]．形式的仕様に基づく再利用に関して，Gaudel らは代数的仕様の再利用性を定義した[2]．また，Wirsing らは再利用可能部品を仕様の木として表し[7]，再利用の手順を形式化した[8]．しかし，Gaudel, Wirsing らは既存の部品の検索については述べていない．既存の部品群からの検索は部品を熟知した技術者が行うため，検索機能は不必要とする考えもある[9]．しかし，部品は常に増加し，未知の部品に対する検索も必要となることを考慮すると，検索機能は必須である．再利用可能部品の検索に関して，Runciman らは要求する関数の型をインデックスとして用いて部品を検索する方法を提案した[5]．しかし，Runciman らは検索後の再利用については述べていない．

一方著者らは，既存の部品群から検索するための手段として代数的仕様の類似性を定義した[6),12]．文献6)，12）では，等式の特徴に着目する構文的な類似性と，

† 名古屋大学工学部
School of Engineering, Nagoya University
†† 北陸先端科学技術大学院大学情報科学研究科
School of Information Science, Japan Advanced Institute of Science and Technology

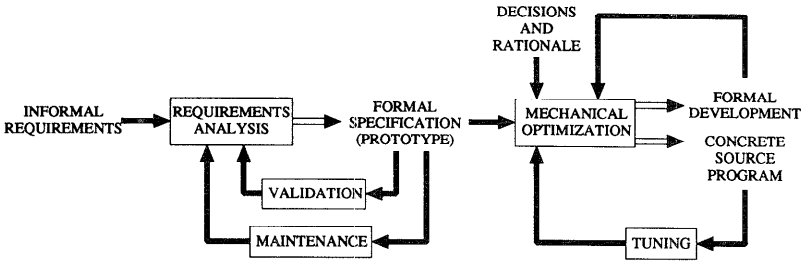


図1 Balzer のソフトウェア開発のモデル図
Fig. 1 Balzer's automation-based software paradigm.

等式部を項書換え系とみなし，意味的情報を用いる類似性を定義した．前者の類似性の定義における等式の分類法は形式性が低く，後者の類似性は正確な評価が困難なため，実行速度の点で問題があった．また，仕様とプログラムの関係，および類似性を利用したプログラムの再利用については詳細を述べなかった．

そこで本論文では，検索から差分抽出，プログラムの再利用までの一連の再利用の方法を提案する．そのために，差分抽出の容易な構文的アプローチを採用する．文献6)の構文的な類似性を形式化して，類似性を順序で与える方法について述べる．また，この方法に基づいて試作した部品検索システムの結果を示す．さらに，形式的仕様を用いて検索した後，プログラムを再利用する方法について，具体例を通して検討する．

以下，2章で本論文で提案する再利用モデルについて，3章で類似性の定義と類似性に基づく仕様の検索について述べる．また4章で部品検索システムと検索結果，および検索した部品を再利用する方法について述べる．

2. 再利用モデル

2.1 再利用モデル

Balzer らは図1のソフトウェア開発のモデルを提案した[1]．このモデルでは，非形式的な仕様から要求分析により形式的な仕様を得て，機械的な最適化によりプログラムを生成する．保守は形式的な仕様に対して行う．しかし，完全な機械化は困難であり，また保守はソースコードに対して行われているのが現状である．

そこで，本論文では仕様からプログラムを得る部分は自動化せず，プログラマがプログラムを作成する立場を取る．ただし，仕様に対する保守をプログラムに反映させるために，ある一定の書式でプログラムを作成し，仕様とプログラムの対応関係を取る．新しいプログラムを作成するときは，データベース中の効率の良い部品を再利用する．部品を仕様に基づき検索し，変更を加えることによって，効率の良いプログラムを作成する．そのために，仕様とプログラムを組としてデータベースに格納する．本論文で提案するモデルを図2に示す．仕様，および対応づけられたプログラムを組とし，再利用可能部品を仕様を用いて検索する．これらの点について，まず仕様とプログラムの対応づけを簡単に述べて，次章で仕様を用いた検索について述べる．

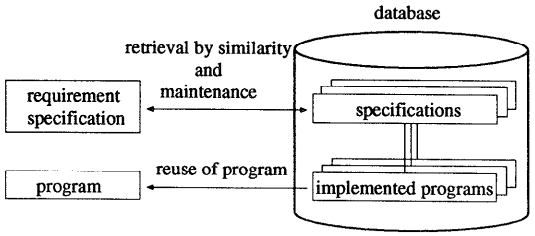


図2 仕様の類似性に基づく再利用モデル
Fig. 2 A model for reuse based on similarity of specifications.

2.2 代数的仕様

仕様とプログラムの対応づけについて述べる前に本論文で扱う形式的な仕様について簡単に説明する．本論文では記述性，読解性に優れるという観点から代数的仕様を用いる．代数的仕様では等式の集合で仕様を記述する．代数的仕様の例を図3に示す*．ソートの集合 S，構成子の集合 C，非構成子の集合 D，変数の集合 V と，機能を表す等式の集合 E からなる．構成子とは等式の左辺の最外に現れない関数記号であり，非構成子とはそれ以外の関数記号である．代数的仕様の詳細については文献10) を参照されたい．本論文で扱う

* 1行目の {DATA} は仕様 APPEND の定義で用いる他の仕様を示す．本論文では各仕様ごとの比較を行い，その仕様で用いている他の仕様は比較しないので，詳細は省略する．この表記法については文献6)を参照されたい．

```
APPEND = ⟨{DATA}, ⟨S, C ∪ D⟩, V, E⟩
S = { list },
C = { Nil :  → list;
      Cons :  data, list → list;},
D = { Append :  list × list → list;},
V = { d :  data; x, y :  list;},
E = { Append(Nil(),y)=y;
      Append(Cons(d,x),y)=Cons(d,Append(x,y));}
```

図3 リストのアペンドの仕様
Fig. 3 Specification of append on list.

仕様では,「等式の左辺には最外を除いて非構成子は現れない」と仮定する．この仮定により，仕様の非構成子とプログラムの関数を一対一に対応させることができる．等式 E を書換え規則と考えれば，これは項書換え系とみなすことができる．後に用いるため，ここで項の定義を述べる．

定義1（項） V を可算無限個の変数の集合，F を有限個の関数記号の集合とする．各関数記号はあらかじめ定められた項数をもつ．記号の集合 $\Sigma_v = V \cup F$ 上の項の集合 $T(\Sigma_v)$ を以下の条件を満たす最小の集合と定義する．

（1） $x \in V \Rightarrow x \in T(\Sigma_v)$.

（2） $f \in F$, f の引数の数が n, $t_1, \cdots, t_n \in T(\Sigma_v) \Rightarrow f(t_1, \cdots, t_n) \in T(\Sigma_v)$. □

2.3 仕様とプログラムの対応づけ

本論文で仕様からプログラムの自動生成の立場を取らない理由は，仕様に記述されていないエラー処理をプログラムに付加したり，効率を良くするためにプログラムに手を加えたりすることがあるためである．本論文では仕様に合わせてプログラマが効率良くプログラムを作成する立場を取る．ただし，仕様とプログラムの対応が分かるようにプログラムを作成する．

構成子 $c \in C$ に関して，関数 c，c の i 番目の引数を取る逆関数 c_inv_i，マッチングの判定部 is_c をプログラマが関数の意味を考えて効率良く作成する．また，仕様を構成する一つの等式に対して，それぞれ if のブロックを一つ作成する．このとき，等式の左辺から if 文の条件部を is_c を用いて作成し，等式の右辺から if 文の実行部を作成する．このとき，等式の右辺に現れる変数が左辺の構成子の引数となっている場合は，逆関数 c_inv_i を用いる．この方法により，図3の仕様から C 言語では**図4**のプログラムを作成する．

この方法でプログラムを作成することによって仕様とプログラムの対応が取れるので，仕様で求めた差分をプログラムに反映することができる．

```
typedef struct list {
    DATA  data;
    struct list *next_list;
} *LIST;
   ...
LIST append(LIST x, LIST y)
{
    if (is_nil(x))
        return(y);
    else if (is_cons(x))
        return(cons(cons_inv_1(x),
                append(cons_inv_2(x), y)));
    else {
        fprintf(stderr, "Unexpected argument");
        exit(1);
    }
}
    ...
LIST cons(DATA x, LIST y)
{
    LIST p;

    p = (LIST)malloc(sizeof(struct list));
    p->data = x;
    p->next_list = y;
    return(p);
}
```

図4 アペンドのプログラム例
Fig. 4 An implemented program of append.

3. 仕様の検索方法

本論文では文献5）と同様にデータベース中の部品を順序づけて整理する．抽象化した等式に順序を与え，この順序に基づいて等式の類似性を定義する．

3.1 諸 定 義

仕様の等式を抽象化するために，関数変数とコンテキスト変数が構成する抽象等式を定義する．

コンテキストとは，項のある部分項をホール(□)で置換したものである．項 $T(\Sigma_v)$ のコンテキストの集合を $T(\Sigma_v \cup \{\square\})$ で表す．$\xi[,]$ をコンテキスト $add(mult(\square_1, y), \square_2)$ とするとき，$\xi[S(0), 0]$ は $\square_1$ を $S(0)$ で，$\square_2$ を 0 で置換した項 $add(mult(S(0), y), 0)$ を表す．コンテキストの引数の数をホールの数と定義すると，引数のないコンテキストは項を表す．任意のコンテキストを表す変数をコンテキスト変数，任意の関数記号を表す変数を関数変数と呼ぶ．関数，関数変数，およびコンテキストに対して，*arity* #はそれぞれの引数の数を返す関数とする．

定義2（抽象項と抽象等式） $\tilde{\Sigma} = \Omega \cup \tilde{F}$ とする．ここで，Ω はコンテキスト変数の集合，$\tilde{F}$ は関数変数の集合であり，各関数変数とコンテキスト変数の引数の数は定められている．また，$\tilde{F} = \tilde{C} \cup \tilde{D}$ で，$\tilde{C}$ は構成子を表す変数の集合，$\tilde{D}$ は非構成子を表す変数の集合である．記号の集合 $\Sigma = \Sigma_v \cup \tilde{\Sigma}$ 上の抽象項の集合 $\tilde{T}(\Sigma)$ を以下の条件を満たす最小の集合と定義する．

（1） $x \in V \Rightarrow x \in \tilde{T}(\Sigma)$.

（2） $f \in F$, $arity\#(f) = n$, $\tilde{t}_1, \cdots, \tilde{t}_n \in$

$\tilde{T}(\Sigma) \Rightarrow f(\tilde{t}_1, \cdots, \tilde{t}_n) \in \tilde{T}(\Sigma)$.

(3) $\xi \in \Omega$, $arity\#(\xi)=n$, $\tilde{t}_1, \cdots, \tilde{t}_n \in \tilde{T}(\Sigma) \Rightarrow \xi[\tilde{t}_1, \cdots, \tilde{t}_n] \in \tilde{T}[\Sigma]$.

(4) $X \in \tilde{F}$, $arity\#(X)=n$, $\tilde{t}_1, \cdots, \tilde{t}_n \in \tilde{T}(\Sigma) \Rightarrow X(\tilde{t}_1, \cdots, \tilde{t}_n) \in \tilde{T}(\Sigma)$.

二つの抽象項を等号（=）で結んだ等式が抽象等式である。 □

$\Sigma_v=\phi$ のとき，$\tilde{T}(\Sigma)=\tilde{T}(\tilde{\Sigma})$ は関数変数とコンテキスト変数で構成され，これを特に**純抽象項**と呼ぶ．純抽象項で構成される抽象等式を**純抽象等式**と呼ぶ．また，$\tilde{\Sigma}=\phi$ のとき，$\tilde{T}(\Sigma)=\bar{T}(\Sigma_v)$ は項の集合 $T(\Sigma_v)$ に等しい．項のコンテキストの集合と同様に，抽象項のコンテキストの集合を $\tilde{T}(\Sigma \cup \square)$ で表す．

以下ではコンテキスト変数を $\xi, \eta, \zeta, \cdots$ で，非構成子を表す関数変数を $X, Y, \cdots$ で，構成子を表す関数変数を $A, B, \cdots$ で表す．また，抽象項は $\tilde{s}$, $\tilde{t}$ などで表す．

次に抽象項に対する代入を定義する．代入は抽象項の関数変数とコンテキスト変数に対して行う．

定義3（代入） 対象を抽象項とする代入 $\sigma: \tilde{T}(\Sigma) \to \tilde{T}(\Sigma)$ を以下のように定義する．

(1) 関数変数への代入 $\sigma_{\tilde{F}}: \tilde{F} \to F \cup \tilde{F}$ は，$A \in \tilde{C}$ に対して A と引数の数が等しい $A' \in \tilde{C}$ または $c \in C$ を割り当てる操作であり，$\sigma_{\tilde{F}}=\{\langle A, A' \rangle\}$，または $\sigma_{\tilde{F}}=\{\langle A, c \rangle\}$ と表す．あるいは，$X \in \tilde{D}$ に対して X と引数の数が等しい $X' \in \tilde{D}$ または $d \in D$ を割り当てる操作であり，$\sigma_{\tilde{F}}=\{\langle X, X' \rangle\}$，または $\sigma_{\tilde{F}}=\{\langle X, d \rangle\}$ と表す．$\sigma_{\tilde{F}}=\{\langle F, f \rangle\}$ のとき $\sigma_{\tilde{F}}(F)=f$ である．

(2) コンテキスト変数への代入 $\sigma_\Omega: \Omega \to \tilde{T}(\Sigma \cup \square)$ は，$\xi \in \Omega$ に対して ξ と引数の数が等しい抽象項のコンテキスト $\xi' \in \tilde{T}(\Sigma \cup \square)$ を割り当てる操作であり，$\sigma_\Omega=\{\langle \xi, \xi' \rangle\}$ と表す．このとき，$\sigma_\Omega(\xi)=\xi'$ である．

(3) $\sigma_{\tilde{F}}$ と σ_Ω を基にして，$\sigma=\sigma_{\tilde{F}} \cup \sigma_\Omega$ を定義する．

ここで $\tilde{t}_1, \cdots, \tilde{t}_n \in \tilde{T}(\Sigma)$ とする．

(a) $\forall x \in V, \sigma(x)=x$.

(b) $\forall f \in F, \sigma(f(\tilde{t}_1, \cdots, \tilde{t}_n)) = f(\sigma(\tilde{t}_1), \cdots, \sigma(\tilde{t}_n))$.

(c) $\forall \xi \in \Omega, \sigma(\xi[\tilde{t}_1, \cdots, \tilde{t}_n]) = \sigma_\Omega(\xi)[\sigma(\tilde{t}_1), \cdots, \sigma(\tilde{t}_n)]$.

(d) $\forall X \in \tilde{F}, \sigma(X(\tilde{t}_1, \cdots, \tilde{t}_n)) = \sigma_{\tilde{F}}(X)(\sigma(\tilde{t}_1), \cdots, \sigma(\tilde{t}_n))$. □

次に，抽象項と抽象等式の半順序を定義する．

定義4（抽象項の半順序） 抽象項 $\tilde{s}, \tilde{t} \in \tilde{T}(\Sigma)$ に対して $\tilde{t}=\sigma(\tilde{s})$ となる代入 σ が存在するとき，（σ に関して）$\tilde{s} \geq \tilde{t}$ であると定める。 □

$\tilde{s} \geq \tilde{t}$ のとき，$\tilde{s}$ は $\tilde{t}$ よりも抽象度が高い．言い替えれば，$\tilde{t}$ は $\tilde{s}$ よりも具体的である．例えば，$\tilde{s}=X(\xi[Y()])$, $\tilde{t}=X(A(Y(), B()))$ とするとき，$\sigma=\{\langle \xi[], A(\square, B()) \rangle\}$ ととると，$\tilde{t}=\sigma(\tilde{s})$ となるので $\tilde{s} \geq \tilde{t}$ である．

抽象項の半順序を定義したが，以下で用いるのは次の2通りの場合である．

・$\tilde{s} \in \tilde{T}(\tilde{\Sigma})$, $\tilde{t} \in \tilde{T}(\tilde{\Sigma})$，すなわち，純抽象項上の半順序．

・$\tilde{s} \in \tilde{T}(\tilde{\Sigma})$, $\tilde{t} \in \tilde{T}(\Sigma_v)$，すなわち，純抽象項 $\tilde{s}$ と項 $\tilde{t}$ のマッチング．この場合，$\tilde{s} \geq \tilde{t}$ が成り立つとき，$\tilde{s}$ は $\tilde{t}$ にマッチするという．また，純抽象等式 $\tilde{e}$ の左辺と右辺の純抽象項が，同一の代入に対して等式 e の左辺と右辺の項にそれぞれマッチするとき，$\tilde{e}$ は e にマッチするという．

定義5（抽象等式の半順序） $\tilde{t}_l, \tilde{t}_r, \tilde{s}_l, \tilde{s}_r \in \tilde{T}(\Sigma)$ に対して，抽象等式の半順序を以下のように定める．

$\tilde{t}_l \geq \tilde{s}_l$ かつ $\tilde{t}_r \geq \tilde{s}_r \Leftrightarrow (\tilde{t}_l=\tilde{t}_r) \geq (\tilde{s}_l=\tilde{s}_r)$. □

最後に，抽象等式の半順序に基づいた部品の整理方法と等式の類似性を定義する．

定義6（抽象等式部品群） 純抽象等式 $abst(N)$ と等式の集合 $Eqn(N)$ を節 N にもち，純抽象等式の半順序にしたがった順序木を，**抽象等式部品群** $\tilde{E}$ と呼ぶ．ただし，$Eqn(N)$ は次の条件を満たす等式の集合である．

$$\forall e \in Eqn(N), abst(N) \in \sqcap \geq \{abst(N') | N' \in \tilde{E}, abst(N') \geq e\}.$$

ここで，$\sqcap \geq \tilde{E}=\{\tilde{e} \in \tilde{E} | \forall \tilde{e}' \in \tilde{E}, \tilde{e} \not\geq \tilde{e}'\}$ である．

□

定義7（類似性） 抽象等式部品群 $\tilde{E}$ と一般の等式 e が与えられたとき，$abst(N) \geq e$ を満たす $\tilde{E}$ の節 N を根から順にたどったときの経路の一つを P とする．このとき，P 上の節の系列 $N_1 N_2 \cdots N_n$ から，$Eqn(N_i)=\phi$ なる N_i を除いた系列が $N_1' N_2' \cdots N_{n'}'$ のとき，等式 e に対する類似性を，

$$Eqn(N_1') \leq Eqn(N_2') \leq \cdots \leq Eqn(N_{n'}') \leq e.$$

と定める。 □

3.2 類似性に基づく仕様の検索

3.2.1 仕様の検索の概要

データベース中の仕様の等式を特徴づけ，分類するために抽象等式部品群を用いる．**図5**に抽象等式部品群の例を示す．理論的には抽象等式 $\xi=\eta$ が木の根になるはずであるが，実用的には根はもっと具体的な抽象等式を用いる．抽象等式部品群はデータベース中の仕様を構成する等式の特徴を表すことができるように用意する．例えば，図5は再帰定義を表している．

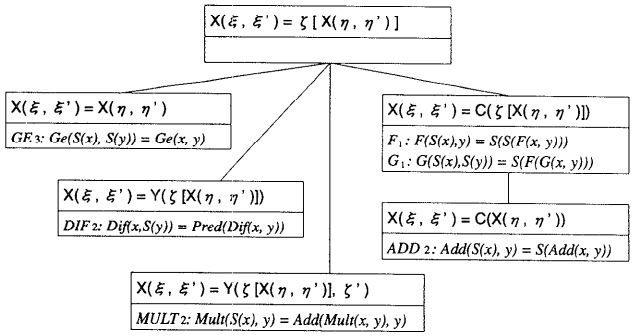


図5 再帰定義に基づく抽象等式部品群
Fig. 5 Abstract equations and equations matching them.

仕様は複数の等式から構成されるので，再利用可能な仕様の候補を検索するために，各等式と類似した等式を検索する．最も類似した等式を検索するために定義7を用いるが，検索は次の方法で行う．抽象等式部品群の根を出発点として，要求仕様の等式にマッチする抽象等式をもった節をたどる．その経路上にあるデータベース中の等式はすべて類似した等式である．ここで具体的な抽象等式がマッチする等式が，より類似しているので，候補の間には定義7にしたがって類似性が定まる．例えば，図5の抽象等式部品群に対して等式 S_1：$Sample(S(x), y)=S(Sample(x, y))$ にマッチする抽象等式をもった節をたどったときの経路を P_1 とすると，この等式と類似した候補として，

$\{F_1, G_1\}\leq_{P_1}\{ADD_2\}\leq_{P_1}\{S_1\}$,

が得られる．この式を経路 P_1 に基づく**類似式**と呼ぶ．

着目する特徴によって抽象等式部品群は異なるので，抽象等式部品群は複数個存在し，経路も複数存在し得る．最終的には何らかの基準によって最も類似した等式を得たいので，異なる経路から導かれる複数の類似式を一つにまとめる必要がある．これを類似式の合成と呼ぶ．

3.2.2 類似式の合成

定義8 EQ を等式の集合，OE を類似式 $EQ_1\leq\cdots\leq EQ_n$ とする．このとき OE 中の $\leq$ の数を与える関数 Len を，

$Len(EQ)-0$

$Len(EQ\leq OE)=1+Len(OE)$

と定義する． □

要求仕様の等式 eq に対して複数の抽象等式部品群上の経路 $1-n$ に対する類似式，

OE_1：$EQ_{11}\leq_1\cdots\leq_1 EQ_{1n_1}\leq_1\{eq\}$,

…

OE_n：$EQ_{n1}\leq_n\cdots\leq_n EQ_{nn_n}\leq_n\{eq\}$,

を仮定する．これらを合成して新たな類似式を得る方法を以下に示す．ただし，$N=\max(Len(OE_i))$，$\max(n)$ は n の最大値とする．

各等式に対して，

EQ_{11}：$N-(Len(OE_1)-1)$,

…, EQ_{1n_1-1}：$N-1$, EQ_{1n_1}：N,

…

EQ_{n1}：$N-(Len(OE_n)-1)$,

…, EQ_{nn_n-1}：$N-1$, EQ_{nn_n}：N,

と重みを与える．ただし EQ_{ij}：n は，EQ_{ij} のすべての要素に重み n を与えることを示す．各 EQ_{ij} の要素には同じ等式が含まれることがあるので，同じ等式の重みの和をその等式の新たな重みとする．そして，新たな重みの順に新しい類似式を得る．

例えば要求仕様の等式 s とデータベース中の等式 a, b, c, d に対して，以下の順序を仮定する．同時に重みを示す．

$\{a, b\}$：$1\leq_1\{c\}$：$2\leq_1\{d\}$：$3\leq_1\{s\}$,

$\{a\}$：$2\leq_2\{c\}$：$3\leq_2\{s\}$.

各等式の重みの和は a：$1+2$, b：1, c：$2+3$, d：3 となる．よって新しい順序は，

$\{b\}\leq\{a, d\}\leq\{c\}\leq\{s\}$,

となる．

ここで，仕様は複数の等式から構成されるので，各等式に対する類似式を合成して仕様の類似式を導かなければならない．各等式に対する類似式を合成して仕様に対する類似式を得る方法は，基本的には等式に対する類似式の合成と同様である．ただし，同一の類似式に同じ仕様を構成する等式が複数現れる場合は必要以上に重みが加算されるので，最高の重みのものだけを用いる．例えば要求仕様 S とデータベース中の仕様 A, B に対して以下の順序を仮定する．同時に重みを示す．ここで a_n は仕様 A の n 番目の式を意味する．

```
function retrieve(SP: 仕様; T: 抽象等式部品群の集合) : 類似式;
  var e : 等式;
      P : 抽象等式部品群中の一つの経路;
      order[ ], sim[ ], sim'[ ], SIM : 類似式;
  begin
    N := (SPの等式の数);
    for n := 1 to N do
      begin
        e := (SPのn番目の等式);
        N_P := (Tにおいてeがたどることができる
                経路の数);
        for i := 1 to N_P do
          begin
            P := (Tにおいてeがたどることができる
                 i番目の経路);
            order[i] := (Pに基づくeに対する類似式);
          end
        sim[n]:=(order[ ]を合成した類似式);
        sim'[n]:=(sim[n]に現れる同じ仕様の等式のうち,
                 最も類似した等式以外を削除した類似式);
      end
    SIM := (sim'[ ]を合成した類似式);
  end
```

図6 仕様検索アルゴリズム
Fig. 6 Algorithm of retrieving specifications.

$\{a_2\} : 1 \leq_1 \{a_1\} : 2 \leq_1 \{b_1\} : 3 \leq_1 \{s_1\}$,
$\{a_2\} : 2 \leq_2 \{b_1\} : 3 \leq_2 \{s_2\}$.

これらを合成すると,

$\{A\} : 4 \leq \{B\} : 6 \leq \{S\}$,

となる. この場合, A は二つの式が S と類似しており, B は一つの式だけが S と類似している. 仕様を等式集合として捉えると奇妙に思われるが, B は一つの等式で S 全体をカバーしていると解釈する. 例えば, B が IF 文で記述され, S の二つの等式が IF 文の真のときの値と偽のときの値に対応する場合などがそれである.

3.2.3 検索アルゴリズム

仕様の検索アルゴリズムを図6に示す. 概略は次のとおりである. 要求仕様 SP の等式 $e \in SP$ に対して抽象等式部品群を用いて類似式を求める. e に複数の抽象等式部品群の根の抽象等式がマッチし, 経路が複数存在する場合, その数だけ類似式が得られる. 得られた複数の類似式を合成して, e と類似したデータベース中の等式を類似式で表す. このとき同じ仕様の等式が一つの類似式に現れる場合, 最も類似した等式を残してそれ以外の等式を削除する. これらをすべての $e \in SP$ に対して行い, 各等式に対する類似式を合成して仕様の類似式を導く.

3.2.4 検索例

類似式の合成による検索の例を示す. 要求仕様を図7の自然数の減算 MINUS とする. 検索対象のデータベースには数値演算を行う関数が格納されており, 仕様数は 10, うち 2 引数関数が 6, 1 引数関数が 4 で, 1 関数当たりの平均等式数は 1.8 である. 用意した抽象等式部品群は, 左辺の特徴, 右辺の特徴, 再帰定義を表したものの 3 種類である. 再帰定義に基づく抽象等式部品群は図5から F_1 と G_1 を除いたものである. 検索の結果を以下に示す. ここで SP_n は仕様 SP の n 番目の等式を意味する.

```
MINUS = (⟨{φ}, ⟨S, C ∪ D⟩, V, E⟩
S = { nat },
C = { Z : → nat; S : nat → nat;
      ERR : → nat;},
D = { Minus : nat × nat → nat;},
V = { x, y : nat;},
E = { Minus(x,Z())=x;
      Minus(Z(),S(x))=ERR();
      Minus(S(x),S(y))=Minus(x,y);}
```

図7 自然数の減算の仕様
Fig. 7 Specification of subtraction on natural number.

$NINUS_1$ について次の類似式が得られた.

$order[1]=\{ABS_1, EQ_1\} \leq_{11} \{DIF_1, GE_1\} \leq_{11} \{MINUS_1\}$,
$order[2]=\{ADD_1, DIF_1\} \leq_{11} \{MINUS_1\}$.

順序 $order[\,]$ を合成して,

$sim[1]=\{ABS_1, EQ_1\} \leq_1 \{ADD_1, GE_1\} \leq_1 \{DIF_1\} \leq_1 \{MINUS_1\}$. (3.1)

(3.1) 式に同じ仕様の等式はないので,

$sim'[1]=\{ABS_1, EQ_1\} \leq_1 \{ADD_1, GE_1\} \leq_1 \{DIF_1\} \leq_1 \{MINUS_1\}$. (3.2)

以下同様にして, $MINUS_2$ に対して,

$order[1]=\{ABS_1, EQ_1\} \leq_{21} \{ADD_1, MULT_1, GE_2\} \leq_{21} \{MINUS_2\}$,
$order[2]=\{ADD_1, DIF_1\} \leq_{22} \{MULT_1, GE_1, GE_2\} \leq_{22} \{MINUS_2\}$.

これらの順序を合成すると,

$sim[2]=\{DIF_1, ABS_1, EQ_1\} \leq_2 \{GE_1\} \leq_2 \{ADD_1\} \leq_2 \{MULT_1, GE_2\} \leq_2 \{MINUS_2\}$. (3.3)

(3.3) 式より同じ仕様の等式を削除すると,

$sim'[2]=\{DIF_1, ABS_1, EQ_1\} \leq_2 \{ADD_1\} \leq_2 \{MULT_1, GE_2\} \leq_2 \{MINUS_2\}$. (3.4)

$MINUS_3$ に対して,

$order[1]=\{ABS_1, EQ_1\} \leq_{31} \{ADD_2, MULT_2\} \leq_{31} \{GE_3\} \leq_{31} \{MINUS_3\}$,
$order[2]=\{GE_3\} \leq_{32} \{MINUS_3\}$,
$order[3]=\{ADD_1, DIF_1\} \leq_{33} \{GE_3\} \leq_{33} \{MINUS_3\}$.

これらの順序を合成すると,

$sim[3]=\{ABS_1, EQ_1\} \leq_3 \{ADD_1, ADD_2, MULT_2, DFF_1\} \leq_3 \{GE_3\} \leq_3 \{MINUS_3\}$ (3.5)

(3.5) 式より同じ仕様の等式を削除すると,

$sim'[3]=\{ABS_1, EQ_1\} \leq_3 \{ADD_2, MULT_2, DIF_1\}$

```
GE = ⟨{φ},⟨S,C∪D⟩,V,E⟩
S = { bool, nat },
C = { T : → bool; F : → bool;
      Z : → nat ; S : nat → nat;},
D = { Ge :  nat × nat → bool;},
V = { x, y :  nat;},
E = { Ge(x,Z())=T();
      Ge(Z(),S(x))=F();
      Ge(S(x),S(y))=Ge(x,y);}
```

図8 減算の仕様と類似した仕様
Fig. 8 Specification similar to MINUS.

$$\leq_3\{GE_3\}\leq_3\{MINUS_3\}. \quad (3.6)$$

(3.2), (3.4), (3.6) 式を合成して,

$$SIM=\{ABS, EQ\}\leq\{MULT\}\leq\{DIF, ADD\}$$
$$\leq\{GE\}<\{MINUS\}. \quad (3.7)$$

最も類似した仕様 GE を図8に示す. ソートは異なるが, 左辺の場合分けや3番目の等式の再帰定義などが類似している. GE は差が0以上か否かを判定する関数であり, 差を求める MINUS と類似している点で直観にも合致する.

4. 評　価

以上の方法に基づいた部品検索システムのプロトタイプと, システムによって検索された部品の再利用例について述べる.

4.1 部品検索システム

4.1.1 システムの概要

ユーザの入力は要求仕様であり, システムはデータベース中の仕様から要求仕様と最も類似した仕様を求め, その仕様とプログラムを出力する. データベースと抽象等式部品群はあらかじめ用意されている.

システムは, あらかじめデータベース中の仕様を構成する等式を, 定義6にしたがって抽象等式部品群の節に登録する. 登録後の抽象等式部品群を用いて, 3.2.3項のアルゴリズムにより, ユーザが入力した要求仕様と類似した仕様を類似式で表す. 最後に, 仕様とプログラムを出力する.

4.1.2 検索結果

検索の対象としたデータベースには文字列処理関数が格納されており, 仕様数は49で, うち3引数関数が2, 2引数関数が14, 1引数関数が29, 引数のない関数が4, 1関数当りの平均等式数は1.4である. このデータベースはラインエディタの仕様[11]を基に作成した. 用意した抽象等式部品群は, 左辺の特徴, 右辺の特徴, 再帰定義を表したもの, および IF 文に関するものの4種類である.

文字列中の特定の一文字をすべて置換する関数 SUBST を要求仕様とする. この仕様を図9に示す.

```
SUBST = ⟨{IF, EQ_CHAR},⟨S,C∪D⟩,V,E⟩
S = { char, line },
C = { I_LINE : → line;
      M_LINE : char × line → line;},
D = { SUBST : line × char × char → line;},
V = { c0,c1,c2 :  char; l0 :  line;},
E = { SUBST(I_LINE(),c1,c2)=I_LINE();
      SUBST(M_LINE(c0,l0),c1,c2)
      =IF(EQ_CHAR(c0,c1),
          M_LINE(c2,SUBST(l0,c1,c2)),
          M_LINE(c0,SUBST(l0,c1,c2)));}
```

図9 要求仕様 (一文字置換)
Fig. 9 Requirement specification (substituting a character).

```
Specification most similar to (SUBST) is this:

Name(exch_lin)
  Other spec. -> (if)(ishere)
  Constructor
    M_LINE : char, line -> line ;
  Nonconstructor
    EXCH_LIN : line, char, int -> line ;
  Equation
    EXCH_LIN(M_LINE(c0, l0), c1, S(n0))
     == IF(ISHERE(S(n0)),
           M_LINE(c1, l0),
           M_LINE(c0, EXCH_LIN(l0, c1, n0)))

  Source File => exch_lin.c
```

図10 一文字置換の仕様の検索結果
Fig. 10 Result of retrieval of SUBST by prototype system.

例えば SUBST ("abcabc", 'a', 'A')='AbcAbc' である. この要求仕様と類似したデータベース中の仕様を, 作成したシステムを用いて検索したところ, 最も類似した仕様として図10のうわがきの仕様が得られた. 要求仕様の1番目の等式と対応する等式は図10の仕様にはないが, 要求仕様の2番目の等式が図10の等式と類似している. 本システムを用いて検索した部品が再利用可能であることを, この例を用いて次節で述べる.

4.2 部品の再利用例

図10から図9の仕様を得るには, 次のような変更を行った後, 関数名を置換すればよい. この変更点はあくまで人手によって求めたものである.

(1) EXCH_LIN の3番目のソート int を char に変更する. これに応じて, 左辺の EXCH_LIN の第3項および右辺の EXCH_LIN の第3項を変更する.
(2) SUBST の第1式を追加する.
(3) IF の条件部である第1項 ISHERE (S (n0)) を EQ_CHAR (c0, c1) に変更する.
(4) IF の第2項 M_LINE の第1引数 c1 を c2 に変更する.

```
LINE subst(l, c1, c2(1))
    LINE l;
    char c1;
    char c2;(1)
{
    LINE rslt, l0, l1, m_line(), m_line_inv_2();
    char c0, m_line_inv_1();

    if (is_i_line(l)) {(2)
        rslt = i_line();(2)
        return(rslt);(2)
    } else(2) if (is_m_line(l)) {
        c0 = m_line_inv_1(l);
        l0 = m_line_inv_2(l);
        if (eq_char(c0, c1)(3)) {
            rslt = m_line(c2(4), subst(l0,c1,c2)(5));
            return(rslt);
        } else {
            l1 = subst(l0, c1, c2(1));
            rslt = m_line(c0, l1);
            return(rslt);
        }
    } else {
        fprintf(stderr,
                "subst:Unexpected argument");
        exit(1);
    }
}
```

図11 変更前のプログラム
Fig. 11 Program before customization.

```
LINE exch_lin(l, c1, n(1))
    LINE l;
    char c1;
    int n;(1)
{
    LINE rslt, l0, l1, m_line(), m_line_inv_2();
    char c0, m_line_inv_1();
    int n0, s_inv();(1)

    if (is_m_line(l) && is_s(n)(1)) {
        c0 = m_line_inv_1(l);
        l0 = m_line_inv_2(l);
        if (ishere(n)(3)) {
            rslt = m_line(c1(4), l0(5));
            return(rslt);
        } else {
            n0 = s_inv(n);(1)
            l1 = exch_lin(l0, c1, n0(1));
            rslt = m_line(c0, l1);
            return(rslt);
        }
    } else {
        fprintf(stderr,
                "exch_lin:Unexpected argument");
        exit(1);
    }
}
```

図12 変更後のプログラム
Fig. 12 Program after customization.

(5) IFの第2項M_LINEの第2引数l0をSUBST (l0, c1, c2) に変更する.

上記の仕様の差分を仕様と対応の取れたプログラムに適用することによって，関数substを得た．その結果を**図11**，**図12**に示す．(1)によって関数宣言の引数を変更した．また，int型の変数や関数の宣言文を削除した．(2)については，対応するif文を追加した．(3)～(5)については，対応する文を変更した．それぞれ図中に下線で示した．

この結果，総数25行の関数exch_linから2行を削除し，3行を追加し，7行を変更することによって，総数26行の関数substを得た．ただし，空行は数えていない．再利用の有効性を示す数値としてプログラムの**再利用率**を2種類定義する．

既存部品再利用率

$$= \frac{\text{既存プログラムの総行数} - (\text{削除分} + \text{変更分})}{\text{既存プログラムの総行数}} \times 100(\%). \quad (4.1)$$

新規部品再利用率

$$= \frac{\text{新規プログラムの総行数} - (\text{追加分} + \text{変更分})}{\text{新規プログラムの総行数}} \times 100(\%). \quad (4.2)$$

(4.1)式は既存のプログラムにおいて再利用した割合を示す値であり，検索された部品がいかに再利用に適しているかを示す．(4.2)式は新しく得られたプログラムにおいて再利用した割合を示す値であり，いかに効率良く新しいプログラムを生成できたかを示す．

この例では既存部品再利用率＝((25－(2＋7))/25)×100～64%であり，新規部品再利用率＝((26－(3＋7))/26)×100≃62%である．新規部品再利用率が約60%であるので，得られたプログラムの半分以上は再利用によって得たものである．すなわち，再利用の成果が認められる．

5. おわりに

形式的仕様を用いた再利用モデルとして，仕様の類似性に基づく部品の検索について述べ，検索システムのプロトタイプと検索結果を示した．また，仕様と対応づけられたプログラムの再利用を具体例を通して検討した．

本論文で示した新規部品再利用率は，文献9)で述べられている再利用率＝再利用規模/出来高＝約20%に比べて大きな値となった．形式的な仕様を用いた今回の例がソフトウェア開発現場での大規模なプログラムにそのまま当てはまるわけではないが，検索から再利用までの手順を形式的に表現する手段が有効であることを示していると考えられる．

本論文で提案した類似性では，引数の数が異なる場合類似しているとみなすことができない．そのような場合でも類似した仕様であり，類似性の定義の改良が必要である．また，部品の整理に適した抽象等式を見つける方法を確立することも今後の課題である．さらに，例を用いて示したプログラムの再利用の形式的表現についての検討も必要である．

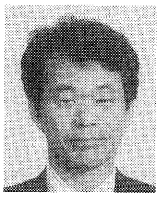
参考文献

1) Balzer, R., Cheatham, T. E., Jr. and Green, C.: Software Technology in the 1990's: Using a New Paradigm, *Computer*, pp. 39-45 (1983).
2) Gaudel, M. C. and Moineau, T.: A Theory of Software Reusability, *2nd European Symposium on Programming, LNCS No. 300*, pp. 115-130, Springer-Verlag (1988).
3) Krueger, C. W.: Software Reuse, *Comput. Surv.*, Vol. 24, No. 2, pp. 131-183 (1992).
4) Maiden, N. A. and Sutcliffe, A. G.: Exploiting Reusable Specifications through Analogy, *Comm. ACM*, Vol. 35, No. 4, pp. 55-64 (1992).
5) Runciman, C. and Toyn, I.: Retricving Reusable Software Components by Polymorphic Type, *Journal of Functional Programming*, Vol. 1, No. 2, pp. 191-211 (1991).
6) Sakai, M., Matsui, S., Kawakita, M. and Agusa, K.: Similarity on Algebraic Specifications toward Specification Databases, Research Report, JAIST, IS_RR_99-0012S (1994).
7) Wirsing, M., Hennicker, R. and Breu, R.: Reusable Specification Components, *Mathematical Foundations of Computer Science, LNCS No. 324*, pp. 121-137, Springer-Verlag (1988).
8) Wirsing, M., Hennicker, R. and Stabl, R.: MENU—An Example for the Systematic Reuse of Specifications, *2nd European Software Engineering Conference, LNCS No. 387*, pp. 20-41, Springer-Verlag (Sep. 1989).
9) 磯田定宏：ソフトウェア再利用の管理的側面，情報処理学会論文誌，Vol. 34, No. 5, pp. 1117-1124 (1993).
10) 稲垣康善，坂部俊樹：抽象データタイプの代数的仕様記述法の基礎 (1) (2) (3) (4)，情報処理，Vol. 25, No. 1, 5, 7, 9, pp. 47-53, 491-591, 708-716, 971-986 (1984).
11) 久保幸弘：対話型ソフトウェアの代数的仕様記述法とプロトタイピング，修士論文，名古屋大学大学院工学研究科電気工学，電気工学第二及び電子工学専攻(1987).
12) 松井聡一：仕様の再利用のための代数的仕様の類似性，修士論文，名古屋大学大学院工学研究科電気工学，電気工学第二及び電子工学専攻(1992).

(平成6年7月29日受付)
(平成7年2月10日採録)

川北　誠

昭和45年生．平成5年名古屋大学工学部電気学科卒業．平成7年同大学大学院博士課程前期課程修了後，オムロン株式会社に入社．在学中，代数的仕様に基づくソフトウェア再利用に関する研究に従事．電子情報通信学会会員．



酒井　正彦（正会員）

昭和36年生．昭和59年名古屋大学工学部電気学科卒業．平成元年同大学大学院博士課程修了．同年同大学工学部助手．平成5年より北陸先端科学技術大学院大学助教授．形式的仕様記述，仕様の検証，項書換え系に関する研究に従事．工学博士．平成3年度電子情報通信学会論文賞受賞．電子情報通信学会会員．

山本晋一郎（正会員）

昭和37年生．昭和61年名古屋大学工学部電気学科卒業．平成3年同大学大学院博士課程修了．同年同大学工学部電子情報工学科助手．項書換え系，関数型言語処理系に関する研究に従事．電子情報通信学会，日本ソフトウェア科学会各会員．

阿草　清滋（正会員）

昭和22年生．昭和45年京都大学工学部電気工学第二学科卒業．昭和49年より同情報工学科に勤務．平成元年より名古屋大学教授．現在，情報工学科に勤務．工学博士．ソフトウェア工学，特に仕様化技術，再利用技術に関する研究に従事．電子情報通信学会，日本ソフトウェア科学会各会員．