

形式的記述に基づくモジュールシグニチャを用いたソフトウェア部品検索

情報システム学講座 0231002 新井 宏文

指導教官： 渡辺 成良 教授 西野 哲朗 助教授

1 はじめに

ソフトウェア資源の有効利用の為にソフトウェアリポジトリにはソフトウェア部品検索機構が必要となる。一般的に広く用いられている検索手法は、キーワードを用いて自然言語で記述されたソフトウェアのドキュメント等と字面上の一致をとるものである。Robert らは検索する際に、要求するソフトウェアのドメインの指定や、キーワード間に論理演算子を用いる枠組みを設けている [1]。しかし、キーワード単体で持っている意味が文脈に依存する上、複数入力されたキーワード自体の意味的關係が判断することが出来ない為、取得する部品が要求を満たす可能性が低い点がある。そこで、健全性を保証する検索手法として、形式的仕様を用いる方法も提案されている [2]。これらの手法では、形式的仕様記述言語を検索キーとして用いるため、記述量が非常に多くなってしまふ点や部品検索を行う段階が、設計が完了した段階で作成された仕様書を基に検索が行われる点があり、検索に柔軟性がないという点がある。

そこで本研究では、入出力のデータ型に着目し、形式的記法を導入し定義したデータ型を用いて、部品単位でシグニチャを記述することにより機能を表現する。さらに、シグニチャによる機能の表現だけでは抽象度が非常に高いため、部品の入出力のデータ型に対して型だけではなく性質を記述する要素を付加したデータ型定義方法を示す。また、比較する対象として形式的仕様記述言語を用いることにより、健全性が高く柔軟な検索手法を提案する。

2 本研究の方針

要求工学が注目されている現在、より堅牢なシステムを作るため、開発行程の様々な部分で、形式手法が用いられるようになった。それにともない、各開発行程で有用なプロダクトが生成される。しかし、これまでの開発で部品検索が行われるシーンは一般的に実装段階であり、再利用対象はソースコードとなる。そこで、部品に付随する各開発行程で生成される、形式的な仕様書、設計書等の再利用も視野にいれた柔軟な検索手法を提案する事が重要となる。

本手法では形式的な手法を用い、その上で抽象的な表現が可能な検索キーの記法を提案する。入出力のデータ型による検索が有効であると考え、データ型定義とそれを用いた機能のインタフェースに当るシグニチャを記述することによって、抽象的に部品を表現する。より要求にあった部品を取得するため、前提としてリポジトリにある部品は、形式的仕様記述言語が付随している事とし、検索キーの比較対象として形式的仕様記述言語を用いる。また、形式的

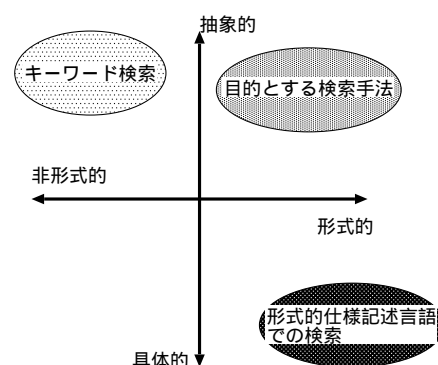


図 1: 目的とする検索手法

仕様記述言語を用いた検索は数学的証明を行い一致の判定を行うが、数学的証明を用いることは時間的コストがかかるため、本手法では一致の判定を文字列の一致問題で行える枠組みをもうける。

本手法では図 1 に示すように、要求の論理的構造の記述が可能な形式的手法を用いて抽象度の高い検索キーを記述し部品を検索する枠組みを提供する事を目的とする。

3 部品の抽象的記述法

本手法では、部品の入出力を機能単位で記述する。シグニチャの記述によって部品のインタフェースを表現する事により抽象度の高い検索キーを実現する。しかしシグニチャによる検索の場合要求の型とリポジトリ内の型の名前が等しくないと検索する事が困難となる。そのためデータ型の定義と複数のデータの間になり立つ関係の式を記述するフレームワークを設けた。上記の方針を実現するために、本手法での書式は以下のように定める。

```
type <データ型の定義>
signature <目的の機能の入出力>
constraint <type で定義したデータ型間の関係>
```

3.1 データ型

本手法では、要求する機能を表現する際にアルゴリズムやモジュール構造を記述しない。これらに代りデータ型に性質の情報を付加することで、機能の抽象化を行う [3]。

3.2 データ型の定義と記法

ここではシグニチャを用いて検索するため、上記で述べたデータ型の定義方法を示す。

```
type
  N = {T; V : D, ..., expression }
```

ここで N はデータ型の識別子、T は後に型構成子や既定義の型識別子、V はデータ型 D の変数、D は N または N を定義するために用いられた型識別子、expression は V を

表 1: 基本データ型と演算子

型名	使用可能な演算子
int	$+, -, *, div, \leq, \geq, =, <, >$
real	整数の演算子, /
char, string	$\leq, \geq, =, <, >$

用いてデータ型の性質を示す式を記述したものである。但しこの中で記述しなければならないものは N のみであり、 $=$ の右辺が記述されていない場合、 N は任意のデータ型を表す。同一の型識別子が検索キーで複数用いられた場合それらは同一の型である事を示す。

また、型宣言において複数列挙する場合は、ブランクや改行によって区切る事とする。

3.2.1 基本型

データ型の高階定義に用いるプリミティブな型として `int`(整数)、`real`(実数)、`string`(文字列)、`char`(文字) を用意する。またそれらに付随する演算子も与える(表 1)。これらの演算子は要求する型の性質をより明確に表すための型の中で閉じた性質を表す式の記述に用いる。さらに演算子に対して適用可能な型を与えることにより、検索キーで用いられた演算子をもとに、型を推論することが可能となる。また、複数の型で使用可能な演算子を設けることによって型を完全に推論せずに基本型から 1 段抽象度の高い検索を提供することが可能となる。

3.2.2 型構成子

データ構造を記述するため、`array`(配列)、`list`(リスト)、`record`(レコード) の各データ型構成子を用意する。

`list[T]` リスト
`array[T]` 配列
`record[T1, T2, ...]` レコード

ここで T, T_1, T_2 は基本型や型識別子である。

これらのデータ構造の性質を記述するため、次の記法を導入する。

`leng(V)` リスト、配列の長さを示す。
`{V}` リストの要素に対し順序関係をとりはらい要素のバッグに落したものを示す。
 V_n, V_j, \dots リスト、配列の要素を示す。但し n, j は任意の自然数である
 $V1.V2$ レコードの要素を示す

ここで、 V はデータ型に当てた変数で、その変数に添字をつけたものをリスト、配列の要素とする。また、 $V1$ はレコード型で $V2$ はレコード内の要素を示している。

レコード型構成子内で定義される要素のデータ型が定義順に関わらず同一ならば等価であるとする。例えば、日付を扱うデータ型の定義は以下ようになる。

```
REC={record[int, int, int]; r : REC, day : int, month :
int; 1≤r.day≤31, 1≤r.month≤12 }
```

またここでは `REC` というレコード型の要素の `int` のどれか 1 つが 1 から 12 までの値をとり、別の `int` が 1 から 31

までの値を取ることを意味する。

3.3 シグニチャの記述

シグニチャとは機能を抽象的にとらえ、データ型で捕えた機能の振舞を示す為のものである。本手法では、上記で示した記述法で定義したデータ型を用いて、それを入出力の型として要求するモジュール単位で以下のように記述する。

```
signature
input, input, ... → output, output, ...
  入力 of の並び                      出力 of の並び
```

このように関数が入出力として扱う型を記述することで、アルゴリズムや部品構造を考慮する必要がなくなる。

3.4 制約式の記述

データ型に対して、他のデータ型から、何らかの制約を受ける場合に記述する式を本稿では制約式と呼ぶ。制約式を記述することにより、シグニチャによる検索からさらに絞り込んだ検索を可能とする。制約式の記述方法は、データ型定義での性質の記法に準ずる。ただし、データ型定義では型の中で閉じた性質を記述するが制約式ではある機能を用いたとき、データ型間で成り立っていることが要求される関係を記述する。

図 2 に本手法で記述した例を示す。ここで x は任意の型を示し、入力リストと出力リストの要素の型が同一であることを表している。また、複数の式を列記するときは “,” を用いて区切る事とする。

```
type
x
LIST = {list[x]}
SLIST = {list[x]; sl : SLIST; sln ≤ sln+1 }
signature
LIST → SLIST
constraint
l : LIST,
sl : SLIST;
leng(l)= leng(sl)
```

図 2: 何らかの型のリストをソートする機能

4 代数仕様 CafeOBJ

本手法で検索キーと比較する対象は形式的仕様記述言語の CafeOBJ[4] とする。CafeOBJ は代数論理を用いており、演算のインターフェースを示したシグニチャによる代数の定義とそれらの関数の公理を示した公理系の記述による代数の限定により機能を表現する(図 3 参照)。

まず `module` の後に名前を記述し、`protecting()` で他の仕様を導入する。[] の中でソート(集合の名前)の宣言を行い、`signature{ }` で演算の定義を行う。`axioms{ }` の中で `signature` で定義された演算の公理の定義を記述する。また、ここで条件付きの等式を記述することによって、演算の振舞に制約を設ける事ができる。これらの記述によって機能を表現する言語である。

汎用な機能を定義しておき、ビューという記述法(図 4)を用いることによりその汎用な機能が入力としてとる型を

```

1. module LIST [ X :: TRIV ]{
2.   protecting(NAT)
3.   [ Elt < NeList < List ]
4.   signature {
5.     op _ _ : NeList List -> NeList
6.     op leng : List -> Nat
7.     op car : NeList -> Elt
8.     op cdr : NeList -> List
9.   }
10.  axioms {
11.    var L : List
12.    var E : Elt
13.    eq leng( E L ) = leng( L ) + 1 .
14.    eq car( E L ) = E .
15.    eq cdr( E L ) = L .
16.  }
17.}

```

図 3: CafeOBJ によるリストの定義

```

1. view INT-as-TRIV from TRIV to INT{
2.   sort Elt -> Int
3. }
4. module INT-LIST{
5.   protecting (LIST[X <= INT-as-TRIV])
6. }

```

図 4: 整数を要素に持つリストの仕様

束縛することも可能である。

5 検索キーからの形式的仕様の生成

利用者によって記述された検索キーを仕様と比較するため CafeOBJ へ変換する必要がある。その仕様の生成方法について述べる。

本稿では検索キーから生成された仕様を生成仕様と呼び、リポジトリ内にある仕様を DB 仕様と呼ぶ。

5.1 データ型定義からの生成

まず、データ型定義から生成仕様を生成する前に、任意の型として定義されたデータ型を性質を表す式等に用いられている演算子を見ることにより、型の判定を行う。型判定を行った検索キーのデータ型に対してそれぞれの基本型、型構成子に当る CafeOBJ へと変換する。また階層構造を持つデータ型(図 6)を定義した場合、ビューを用いた CafeOBJ へと変換する。これらを用いて、生成仕様のデータ型定義に当るモジュールを生成する。

5.2 インタフェース定義からの生成

検索キーの signature 部で記述された部品機能のインタフェースに当たる部分を CafeOBJ 仕様へと変換していかなくてはならない。そこで、5.1 節の方法で生成されたデータ型の CafeOBJ 仕様を輸入するモジュールを定義する。さらに、輸入したモジュールのソートを用いて演算子定義を行うモジュールを生成する。

5.3 データ型の性質と機能の制約

検索キー内に記述した性質に当る等式を axioms 部に人の手を介して記述することは可能であるが、一つの式から複数の等式が生成される可能性が高い。その為、性質、制約の部分は CafeOBJ 化をせずに逆に部品登録時に CafeOBJ 仕様の条件付き等式の記述を、左辺を処理前右辺を処理後と見做すことによって条件付き等式の中から、入力型の性質、出力型の性質、機能の制約情報を抽出する。

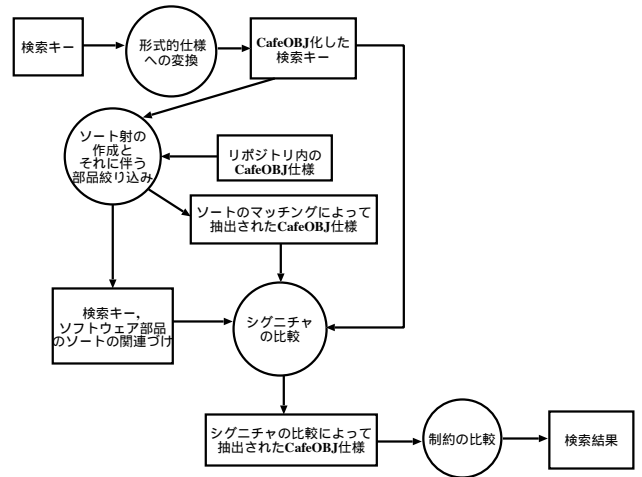


図 5: 照合手順

6 部品照合手法

これまで示してきた記述法を検索キーとして用いリポジトリ内のソフトウェア部品と照合する方法を示す。

本手法では前提として基本型、型構成子に対する代数仕様は既に定義されており、開発者がプロダクトを作成時に定義されている基本型等の代数仕様を輸入して仕様を記述したものがプロダクトと共にリポジトリへ登録されているものとする。

また、本手法において検索キーの抽象度が高いため生成される仕様は不完全である。そこで、CafeOBJ 仕様での演算の定義や、等式は振舞の規則を定義しているものと考え、検索キー内から生成した仕様の記述を全て網羅する仕様を一致したものと判定する。

照合手順(図 5)は以下の通りである。

1. 生成仕様と DB 仕様でソートのマッチングを行い、ソートの関連づけを行う
2. 生成仕様のソート名を関連づけにしたがって変換する
3. 生成仕様の CafeOBJ シグニチャと DB 仕様のシグニチャで文字列一致を見る
4. 登録時に抽出された性質と、制約を用いて検索キーに記述された制約にマッチするかを比較する
5. 検出結果を出力する

6.1 データ型の比較

データ型に関する生成仕様のデータ型を表すモジュールのソート $\{s^k\}$ と、DB 仕様中のソート $\{s^d\}$ に対してソートの射 $s^k \rightarrow s^d$ を可能な生成する。それに従って各 s^k を置き換え生成仕様のシグニチャ Σ^k から $\Sigma^{k'}$ を作る。そして、DB 仕様のシグニチャ Σ^d に対して $\Sigma^{k'} \subseteq \Sigma^d$ が文字列レベルで成り立つかを判定する。成り立たない場合は生成仕様の演算を比較した DB 仕様が含まれていない場合、輸入関係を辿り、一つ上の仕様を用いて再び比較を行う(図 6)。成立する場合は、ソートの射の集合 $s^k \rightarrow s^d$ を用いてインタフェースの比較を行う。

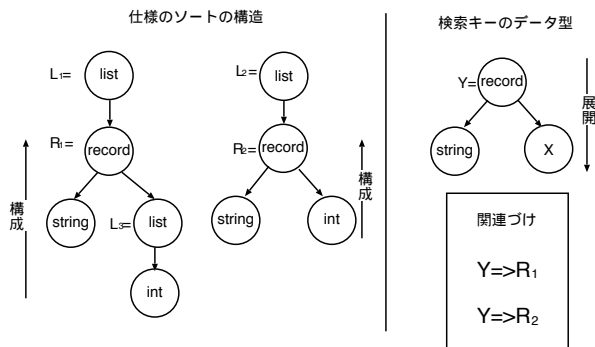


図 6: データ型のマッチングの概要

6.2 インタフェースの比較

データ型の比較により、作られたソートの射を用いて、生成仕様の全てのシグニチャに対して代入を行う。同じ機能のインタフェースが等しい場合、ソートの置換を行えば以下のように CafeOBJ の記述の : の右側が同じ記述となる。

```
op sorting : List -> List --DB 仕様
op a : List -> List --生成仕様
```

6.3 性質、制約の比較

一般的に、CafeOBJ モジュールの等価性を測るとき 2 つのモジュールの間で、書き換え規則に法則が同じであるかを計算していく詳細化検証という手法を用いる。しかし、本手法では証明という作業を極力削減するため、入力された検索キーにデータ型の性質や機能のインタフェースに関する制約が記述されている場合、登録時に抽出しておいた入出力の型の性質の式と検索キーとして与えられた制約の式の形 (式の形は式中に用いられている演算子や変数の型によって決まるものとする) を比較することによって一致しているかを判断する。その結果を検索結果とする。

7 考察

性質を付加したデータ型を用いてシグニチャを記述する事、型定義時に任意の型という記述を許すことによって機能を形式的かつ抽象的に示す方法を提案した。これにより、上流工程での部品の検索が可能となり、ソースコードだけでなくソフトウェア仕様書等の各工程のプロダクトの再利用が期待できる。しかし、本稿で用意した演算子は抽象度が低いため高階なデータ型間の関係を記述する事ができない。その為、現段階では性質を記述するには基本型まで具体化しなければならない。

7.1 記述方法

データ型に特化した記述方法を用いたため、部品のアルゴリズムやモジュール構造を排除することにより、記述量を削減する事が可能となった。また、任意の型の定義をすることができるにより記述可能な抽象度に大きく幅を持たせることが可能となった。

しかし、型構成子間の関係を記述する演算子が用意されていないため、複雑なデータ型を定義した場合、更にタイトな要求を記述することが出来ないという点がある。

7.2 検出方法

比較を行う際に数学的な部分を出来るだけ排除し文字列一致問題へ変換することによって、これまで数学的証明を行っていた検索に比べて時間的コストを削減できると考えられる。また、性質の一致を見る際に登録時の計算により、CafeOBJ 仕様から部品の入出力の型に関する性質や機能にかかる制約を抽出しておくことにより、全体的に証明機に依存する部分が軽減される。

本手法は抽象的な検索を提供していくため、取得部品が検索キーの要求を満たすが、完全に要求を満たした部品が取得できるとは限らないため、よりの確な検索を行えるようにするためには、システム側に、何らかの付加機能を持たせる必要があると考えられる。

8 まとめ

本研究では部品仕様の再利用も視野に入れ、これまで提案されていた形式手法を用いる検索とは異なる部品検索機構を構築することを最終目標としている。そこでより抽象度の高い検索キーを記述するために、入出力のデータ型を定義し、それを用いてシグニチャを記述する。また、細かい要求も記述できるよう、これまで機能として記述してきた部分をデータ型の性質という捉え方をすることにより、同じデータ型でも異なる要素を持つ型であるという部分を記述できる枠組みをもうけた。これにより、部品に対するモジュール構造等を記述しない要求の表現が可能となった。

検索キーと比較する対象として、形式的仕様記述言語 CafeOBJ を用いることとし、登録されている部品には、CafeOBJ 仕様が付随しているものとした。一致を判定する際に数学的証明をなるべく排除し、文字列の一致による判定をすることにより従来の形式手法を用いた検索より時間的コストを削減できる可能性もある。また形式的記法の採用により文字列による判定を用いた上で、健全性の高い検索手法を提案した。

謝辞

本研究を進めるに当たり終始温かいご指導を頂きました渡辺成良教授、並びに西野哲朗助教授、織田健助手に厚く御礼申し上げます。

参考文献

- [1] Robert C. Seacord, Scott A. Hissam, and Kurt C. Wallnau. Agora: A search engine for software components. *IEEE INTERNET COMPUTING*, pp. 62–70, 1998.
- [2] 栗野俊一, 松澤裕史, 深澤良彰. ソフトウェア部品検索における形式的仕様の活用法. *ソフトウェア工学の基礎 I*, pp. 129–136, 1994.
- [3] 新井宏文, 織田健. 形式的記述に基づくモジュールシグニチャを用いたソフトウェア部品検索. *FIT 2003*.
- [4] 中川中, 谷津弘一, 本間毅寛. CafeOBJ への誘い. *CafeOBJ HomePage*, 1996.