TR-116

一階述語論理式を用いた
ソフトウェアモジュールの機能検索

吉田裕之，加藤英樹，杉本正勝
(富士通)

June, 1985

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

一階述語論理式を用いた
ソフトウェアモジュールの機能検索

吉田裕之，加藤英樹，杉本正勝
富士通株式会社

ABSTRACT

This paper introduces a method to retrieve software modules from the module library in order to reuse them as parts of new software. It will be effective to the programming environment in which specifications of software modules are formalized using first-order predicate logical formulae. This method uses the resolution and heuristics to decide reusability of existing modules. A prototype system has been developed using C-Prolog on VAX11/780.

1．はじめに

Logic Programming におけるプログラム言語は，一階述語論理式のサブセットであるホーン節を理論的基礎とするものである．したがって，そうした言語で記述される各ソフトウェアの仕様を規定する言語が，論理式と遠くかけ離れたものになることは考えがたい．その意味で，一階述語論理式そのものを仕様記述言語の一つの候補として考えることができる．しかし，一階述語論理式は人間にとって書き易いものでも読み易いものでもなく，人間向きのインタフェースが必要になってくる．現在，東工大が中心となって研究を進めているＴＥＬＬシステム[1]は，自然言語によるインタフェースと一階述語論理式による形式的な仕様記述を融合させようという試みである．すなわち，ＴＥＬＬシステムの中核となる仕様記述言語ＴＥＬＬ／ＮＳＬは構文を制限された自然言語（英語）であって，それを用いて記述したソフトウェアの仕様を構文解析することによって一意に一階述語論理式に翻訳することができる．

ソフトウェアの仕様をきちんと書いておくことのメリットは，設計者の意図をプログラムそのものよりも正確に読み取れるために，保守が容易になることだけでなく，既存のソフトウェアの再利用を容易にすることにもある．多くの有用なソフトウェアがライブラリとして整備されその仕様を簡単に検索できれば，それらの既存ソフトウェアの再利用を促進し，生産性を大きく向上させることができる．その場合，検索をある程度自動的に行えることが望ましいが，単なる字面上のパターンマッチングでは実用的にはなりがたく，ソフトウェアの機能に注目した検索手法が求められている．すなわち，設計者の意図である「仕様の意味」を何んらかの形で捉え，意味のマッチングを行うことである．本稿で論ずるのは，ＴＥＬＬシステムのように仕様の意味を一階述語論理式で定式化できるようなプログラミング環境におけるソフトウェアモジュールの機能検索手法である．本手法では，一階述語論理式のマッチング手段として導出法とヒューリスティクスを用いている．以下本稿では，２節で仕様記述言語ＴＥＬＬ／ＮＳＬを簡単に紹介し，３，４節で本手法の概

情処 32 - 116

要とアルゴリズムを述べ，５節でVAX11/780上にC-Prologを用いて開発されたプロトタイプについて報告する．

２．仕様記述言語ＴＥＬＬ／ＮＳＬ

ＴＥＬＬ／ＮＳＬを用いて記述できるモジュールは，述語や関数に相当する機能定義，抽象データ型に相当するクラス定義，並列プロセスに相当する動作定義，プロセス間の共有データに相当する動的クラス定義の４種があるが，本検索手法で対象にしているのは現在のところ機能定義のみである．

Fig. 1にＴＥＬＬ／ＮＳＬを用いて記述したエイトクイーンパズルを解く場合のトップレベルのモジュールの定義を例示する．この仕様は次の(1)式に翻訳される．

```
Arrangement X is an eight queens' solution
  means that
    1) Eight queens are placed in X,
    2) No queen is checking against any other queen in X,
end eight queens' solution;
```

Fig. 1　A Sample Specification (by TELL/NSL)

$$\forall c\ [\mathrm{eight\_queens'\_solution}(c) \equiv \exists S\ [\,|S|=8 \wedge \forall q\ [q \in S \Rightarrow \mathrm{placed}(q,c)]\,] \wedge \sim \exists q1\, \exists q2\ [q1 \neq q2 \wedge \mathrm{checking}(q1,q2,c)\ ]\ ] \quad \cdots(1)$$

ＴＥＬＬ／ＮＳＬは，擬似自然言語であって一階述語論理式に翻訳できることに加えて，語義分割法によって段階的詳細化を自然に行えるという特徴を持っている．つまり各モジュールに一つの単語が対応付けられ，モジュールの仕様記述はその単語の意味を自然言語を用いて説明することに相当する．その説明文中に現れる単語は，もとのモジュールの下請けとなる補助モジュールとしてさらに説明することになる．こうして，ソフトウェア全体のモジュール構造やモジュール間のインタフェースが自然言語による説明文に基づいて自然に定まるのである．各モジュールの仕様は補助モジュールとの論理的な関連を定義することになり，比較的簡単な論理式に翻訳される．したがって，それらの論理式に対して導出法を用いても十分実用的な反応速度を得られることが期待できる．また，小さなモジュールが数多く作られる傾向があり，機能検索を行って再利用を促進することの効果は大きい．

３．機能検索

本手法を用いた機能検索システムは，例えば以下の手順で利用されるものである．

① ライブラリには，モジュールごとにＴＥＬＬ／ＮＳＬを用いた仕様，それを翻訳した論理式，その論理式を満足することが検証されているプログラムの三つ組が格納されている．

② 新規のモジュールの開発者は，そのモジュールの仕様をＴＥＬＬ／ＮＳＬを用いて記述する．

③ 新規のモジュールの仕様は構文解析され，一階述語論理式に翻訳されて，機能検索システムに入力される．

④　機能検索システムは，入力された論理式とライブラリ中の各モジュールの論理式を比較して再利用可能か判定する．

⑤　開発者は，再利用可能と判定されたモジュールのプログラムを加工して新規のモジュールのプログラムを得る．

本機能検索手法は論理式の同値性判定に導出法を用いているので，字面や言回しの相違とは無関係に，二つの仕様が論理的に等価であれば再利用可能と判断する．例えば次の(2)式のように翻訳される仕様を持つモジュール eight_queen_puzzle は，Fig. 1 に例示したモジュール eight_queens'_solution として再利用可能である．

```
∀ c [eight_queens_puzzle(c) ⇔
        ∀q1∀q2 [checking(q1,q2,c) ⇒ q1=q2]
      ∧ ∃ S [ | S | =8 ∧ ∀ q [ q ∈ S ⇔ placed(q,c)] ] ]          …(2)
```

しかしながら現実には，等価な機能を持つモジュールがライブラリに既に登録されていることは稀であって，機能検索システムが有効に利用できるためにはある程度「類似」したモジュールも検索できなければならない．その場合，どういう時に二つの仕様が類似しているとみなすかが問題である．機能検索の主たる目的はプログラムの再利用を促進することにあり，検索された既存モジュールのプログラムを加工するだけで容易に望みのプログラムを得ることができなければならない．したがって，機能検索システムは単に「類似している」と答えるだけでなく，プログラムテキストのどこをどう加工すればよいかの指針を与える必要がある．こうした点から本手法では現在のところ以下に述べる 3 種，およびそれらを組み合せた場合に「類似」と判定している．

(a)　引数の順序の相違

仮引数の順序を入れ換えれば等価になるような場合である．例えば，それぞれ以下の(3)(4)式のように翻訳される仕様を持つモジュール successor，predecessor は，二つの仮引数を入れ換えれば論理的に等価である．

```
∀ x ∀ y [successor(x,y) ⇔ x=increment(y) ]                    …(3)
∀ z ∀ w [predecessor(z,w) ⇔ increment(z)=w ]                  …(4)
```

したがって，機能検索システムは(3)式を入力すると，ライブラリに predecessorが登録されていれば(5)式を返す．

```
∀ x ∀ y [successor(x,y) ⇔ predecessor(y,x)]                   …(5)
```

利用者は，ライブラリからモジュールpredecessor のプログラムテキストを取り出し，第一引数の z を y に，第二引数の w を x にすべて置き換えれば新しいモジュール successor のプログラムを得ることができる．

(b)　引数の定数化

何んらかの定数を一部の実引数として与えると等価になるような場合である．例えば，次の(6)式のように翻訳される仕様を持つモジュール n_queens'_solution は，第一引数

として定数８を与えれば，Fig. 1で例示した eight_queens'_solution として利用可能である．

```
∀ n ∀ c [n_queens'_solution(n, c) ≡
        ∃ S [ | S | =n ^ ∀ q [ q ∈ S ≡ placed(q, c)] ]
      ^ ～∃q1 ∃q2 [q1≠q2 ^ checking(q1, q2, c) ] ]                    …(6)
```

したがって，機能検索システムは(1)式を入力すると，(7)式を返す．

```
∀ c [eight_queens'_solution(c) ≡ n_queens'_solution(8, c)]          …(7)
```

利用者は，ライブラリからモジュール n_queens'_solution のプログラムテキストを取り出し第一引数のｎをすべて定数８に置き換えれば，新しいモジュール eight_queens'_solutionのプログラムを得ることができる．

(c)　補助モジュールの相違

下請けの補助モジュールを同じものに変えれば等価になるような場合である．例えば(8)(9)式のように翻訳される仕様を持つモジュール sort と generate_testにおいて，それらの補助モジュールである permutationと generatedおよび sorted と tested がそれぞれ論理的に等価であると仮定すると，もとのモジュールどうしも等価になる．

```
∀ x ∀ y [sort(x, y) ≡ permutation(x, y) ^ sorted(y)]               …(8)
∀ z ∀ w [generate_test(z, w) ≡ generated(z, w) ^ tested(w)]        …(9)
```

そこで，機能検索システムは(8)式を入力すると(10)式，および補助モジュールの同値性の仮定である(11)(12)式を返す．

```
∀ x ∀ y [sort(x, y) ≡ generate_test(x, y) ]                        …(10)

∀ x ∀ y [permutation(x, y) ≡ generated(x, y)]                      …(11)
∀ x [sorted(x) ≡ tested(x)]                                        …(12)
```

利用者は，ライブラリからモジュールgenerate_test のプログラムテキストを取り出し，その中に現れる補助モジュール名 generatedを permutationに，testedを sorted にすべて置き換えれば新しいモジュール sort のプログラムを得ることができる．ここで注意したいのは，本手法が(11)(12)式の仮定をあえて検証しないことである．したがって，本手法では場合によっては機能がまったく異なるモジュールを検索することがありうる．しかし，実際には(11)(12)式が成立しているか否かに関わらず，上記のようにgenerate_test のプログラムは容易に再利用できるのである．新モジュール sort の補助モジュール permutation，sortedに関しては，それらが既存のものであるならばそれをそのまま利用すればよいし，新しいモジュールであればまた sort と同様に仕様を定義してライブラリから再利用可能なモジュールを機能検索すればよい．その場合でも，検索されたモジュールが generatedや tested である必要はないのである．

4．検索の原理

各モジュールの引数は，仕様を翻訳した論理式の中では全称的に束縛された論理変数で表現されている．したがって，前節(a)(b)の類似性は論理式の同値性を導出法で検証する過程で，それらの論理変数どうしあるいは論理変数と定数が単一化（unify ）されることから自然に判定できる．一方(c)の類似性判定は言わば高階の単一化を行うことになり，本手法ではヒューリスティクスとして実現されている．

以下に本手法が用いている手順とヒューリスティクスを述べる．

(a) 用語と記法

述語名はｐ，ｆ，ｇ等の英小文字で表記し，一般の論理式はＦ，Ｇ等の英大文字で代表する．述語ｐの引数 X1, X2, ..., Xn の組をタップルとして「$\underline{X}$」と表記する．したがって，ｐの呼出しは「p($\underline{X}$) 」と表記される．タップルの要素の順序を適当に並び換える操作を置換と言う．タップル$\underline{X}$を置換πで並び換えたタップルを「$\underline{X}\pi$」と表記する．二つのタップル$\underline{X}$＝<X1, X2, ..., Xi>，$\underline{Y}$＝<Y1, Y2, ..., Yj>に対して，タップル <X1, X2, ..., Xi, Y1, Y2, ..., Yj> を$\underline{X}$と$\underline{Y}$の連結と言い，「$\underline{X}+\underline{Y}$」と表記する．

述語の呼出し，およびその否定をリテラルと呼ぶ．特に，述語ｐの呼出しp($\underline{X}$)，およびその否定～p($\underline{X}$) をそれぞれｐの正のリテラル，ｐの負のリテラルと呼ぶ．

節はリテラルの集合である．唯一つのリテラルからなる節を単位節と呼び，特に述語ｐのリテラル一つのみからなる節をｐの単位節と呼ぶ．

その他一般的な用語，表記法は文献〔2〕による．

(b) 手順

以下では，検索対象である新規のモジュールの名前をｆとし，それと等価であることを検証する既存のモジュールの名前をｇとする．モジュールｆ，ｇの定義をそれぞれ(13)(14)式とする．

$$\forall \underline{X}\ [f(\underline{X}) \equiv F(\underline{X})\ ] \qquad \cdots(13)$$

$$\forall \underline{Y}\ [g(\underline{Y}) \equiv G(\underline{Y})\ ] \qquad \cdots(14)$$

第一フェーズでは，モジュールｇの機能がモジュールｆの機能を満たしているか，検証する．つまり，モジュールｆの引数タップル$\underline{X}$に対して(15)式を満たすようなモジュールｇの引数タップル$\underline{Z}$が発見できればよい．

$$\forall \underline{X}\ [g(\underline{Z}) \supset f(\underline{X})\ ] \qquad \cdots(15)$$

この時，検証すべき式(15)があらかじめ分っていないので，通常の導出法をそのまま適用することができない．そこで，本手法ではモジュールｆの引数タップル$\underline{X}$を固定し，(16)(17)式を公理としてモジュールｇの単位節(18)式を導こうとする．

$$\sim F(\underline{X}) \qquad \cdots(16)$$

$$\forall \underline{Y}\ [G(\underline{Y}) \Rightarrow g(\underline{Y})\ ] \qquad \cdots(17)$$

$\sim g(\underline{Z})$ …(18)

(18)式が導ければ(19)式が成立し，これと(13)式とから(15)式が検証できたことになる．

$\sim F(\underline{X}) \Rightarrow \sim g(\underline{Z})$ …(19)

この時タップル$\underline{Z}$は，通常タップル$\underline{X}$（あるいはそれにいくつかの定数からなるあるタップルを連結したもの）の置換になっている．

(15)式だけでは，モジュールｇがモジュールｆの部分解にすぎない場合もあるので，第二フェーズでは逆の(20)式も検証する．

$\forall \underline{X} [f(\underline{X}) \Rightarrow g(\underline{Z})]$ …(20)

(15)(20)式より(21)式が成立し，ｆとして再利用可能な既存モジュールｇが発見できたことになる．

$\forall \underline{X} [f(\underline{X}) = g(\underline{Z})]$ …(21)

(c) ヒューリスティクス

① ｇのリテラルの強制簡約化

導出法の効率向上のために用いられ，本手法の第一フェーズに特有なヒューリスティクスである．すなわち，導出された節がｇの負のリテラルを複数個含む時，もしそれらが簡約化（factoring ）可能ならば強制的に簡約してｇの負のリテラルを常にたった一つのみ含むようにする．簡約化できないならばその節を削除する．これは，第一フェーズでは最終的に導きたい節が空節ではなく，(18)式のようなｇの単位節だからである．

② 補助モジュールに関する類似性判定

ｆの補助モジュールｈと，ｇの補助モジュールｋで，引数の数が等しくかつそれらが等価であると仮定すると推論がさらに進むような組を発見するヒューリスティクスである．

第一フェーズでは，それまでに導出されている節の中から，以下のような二つの節Ｃ1，Ｃ2 を捜す．

- Ｃ1 がｈの正のリテラル$h(\underline{V})$ を含み，かつＣ2 がｋの負のリテラル$\sim k(\underline{W})$ を含む．あるいはＣ1 がｈの負のリテラル$\sim h(\underline{V})$ を含み，かつＣ2 がｋの正のリテラル$k(\underline{W})$ を含む．
- タップルの置換πが存在し，$\underline{V}\pi$と$\underline{W}$のそれぞれ対応する要素のクラスが等しく，また$\underline{V}\pi$と$\underline{W}$が最汎単一化作用素（most general unifier）σを持つ．
- もしも，Ｃ1 とＣ2 がそれぞれｇの負のリテラル，$\sim g(\underline{Y})$，$\sim g(\underline{Z})$ を含むならば，$\underline{Y}\sigma$と$\underline{Z}\sigma$は単一化可能（unifiable ）である．

以上の条件を満たす時，ｈとｋが（引数を置換πで並び換えると）等価になるもの

と仮定すればC1, C2 から新しい節が導出でき，たとえその節にgの負のリテラルが複数個現れても上記①のヒューリスティクスによって排除されることがない．そこで以降は，(22)式を公理に加えて検証を進める．

$$\forall \underline{V}\ [h(\underline{V}) \equiv k(\underline{V}\pi)\ ] \qquad \cdots(22)$$

第二フェーズでも，この(22)式を公理の一つとして用いる．

③ 再帰的定義の扱い

再帰的に定義されている二つのモジュールの同値性を判定するためには，厳密にはそれらが操作するデータ構造に関する帰納法を用いなければならないが，これを計算機で自動的に行うことは非常に困難である．しかし，そういう厳密な方法によらなくとも，見た目に明らかに等しいと判断できる場合がしばしばある．その際に注目しているのは，それらのモジュールの本体中での再帰呼出しの現れ方である．下請けの呼出し方から判断するという方針は，上に述べたヒューリスティクス②のそれと同じものであり，本手法では②と同様の方法で再帰的なモジュールの同値性を判定することにした．

第一フェーズでは，それまでに導出されている節の中から，以下のような二つの節C1, C2 を捜す．

- C1 はfの負のリテラル$\sim f(\underline{V})$ と，gの負のリテラル$\sim g(\underline{Z})$ のみからなる．
- C2 はgの正のリテラル$g(\underline{W})$ と負のリテラル$\sim g(\underline{Z})$ のみからなる．
- 定数のみからなるタップル$\underline{A}$（長さ0の場合を含む）と置換πが存在して，
  $(\underline{V}+\underline{A})\pi$ と$\underline{W}$が最汎単一化作用素σを持ち，$(\underline{X}+\underline{A})\pi$ と$\underline{Z}\sigma$が等しい．

以上の条件を満たす時，(23)式が満たされたものとして第一フェーズを終了するのである．

$$\forall \underline{X}\ [g(\underline{Z}\sigma) \Rightarrow f(\underline{X})\ ] \qquad \cdots(23)$$

第二フェーズでも同様にして再帰呼出しを処理する．

さて，このヒューリスティクスがリスト構造に関して再帰的に定義されたモジュールの場合に正しいことを以下に示す．簡単のために，f，gとも二引数で第一引数が入力，第二引数が出力に使用される述語とし，定義をそれぞれ(24)(25)式とする．

$$\forall x \forall y\ [f(x,y) \equiv F(x,y)] \qquad \cdots(24)$$

$$\forall x \forall y\ [g(x,y) \equiv G(x,y)] \qquad \cdots(25)$$

第一フェーズでは，モジュールfの引数x，yを固定し，(26)(27)式を公理としてモジュールgの単位節を導こうとし，ヒューリスティクス③は(28)(29)式のような節が導出されることを監視する．

$$\sim F(x,y) \qquad \cdots(26)$$

$$\forall v \forall w\ [G(v,w) \Rightarrow g(v,w)] \qquad \cdots(27)$$

$$\sim f(cdr(x),y) \vee \sim g(x,y) \quad \cdots(28)$$

$$g(cdr(x),y) \vee \sim g(x,y) \quad \cdots(29)$$

(28)(29)式の節が導出されるならば，(24)(25)(26)(27)式から次の(30)(31)式が導ける．

$$\forall x \forall y \ [g(x,y) \cdot f(cdr(x),y) \Rightarrow f(x,y)] \quad \cdots(30)$$

$$\forall x \forall y \ [g(x,y) \Rightarrow g(cdr(x),y) \vee f(x,y)] \quad \cdots(31)$$

ここで，(32)式が成立していることをリスト構造に関する帰納法で証明する．

$$\forall x \forall y \ [g(x,y) \Rightarrow f(x,y)] \quad \cdots(32)$$

(1) xがnil の場合

リテラルg(cdr(x),y) は「$\exists z$ [cdr(x,z)・g(z,y)]」のことであるからこの場合は偽であり，(31)式より(33)式が導ける．

$$\forall y \ [g(nil,y) \Rightarrow f(nil,y)] \quad \cdots(33)$$

(2) xがcdr(a)の場合

(34)式を仮定すると，(30)(31)式より(35)式が導ける．

$$\forall y \ [g(cdr(a),y) \Rightarrow f(cdr(a),y)] \quad \cdots(34)$$

$$\forall y \ [g(a,y) \Rightarrow f(a,y)] \quad \cdots(35)$$

5．プロトタイプシステムと再利用例

本手法を用いた機能検索システムのプロトタイプが，VAX11/780 上のC-Prologを用いて実現されている．これは，およそ2000行のプログラムであり，順序付き線型導出法［2］を用いて論理式の同値性を判定している．また，4節で述べたヒューリスティクスの内，①③に関しては適用可能ならば随時適用し，②は現在のところ導出法が行き詰った時のみ考慮している．

このプロトタイプは，ＴＥＬＬ／ＮＳＬを仕様記述言語とするプログラム開発環境［3］を構成するモジュールの一つとして動作し，半自動合成システムと呼ばれるもう一つのモジュールの下請けとしても用いられている．半自動合成システムは，ソフトウェアの設計者が会話的に使用し，ＴＥＬＬ／ＮＳＬによる仕様記述，パーザによる論理式の翻訳，機能検索システムによる再利用可能モジュールの検索，検索されたモジュールの参照および加工，ライブラリへの登録等の機能を持っている．

Fig. 2は，既存モジュール on_the_same_row を再利用して，on_the_same_column のプログラムを合成する例である．以下順を追って説明する．

① ユーザは求めるプログラムの仕様(a)を入力する．

② 半自動合成システムはパーザを用い，(a)の仕様を論理式(b)に変換する．

③ 半自動合成システムは続いて，機能検索システムを呼出し，類似モジュールを検索する．

④ 機能検索システムは，類似したモジュールの名前(c)，その論理式(d)，両者が一致する条件の論理式(e)，プログラム(f)を返す．

⑤ ユーザは(f)を見て利用可能と判断し，(e)をもとに(f)のプログラムを加工して求めるプログラム(g)を得る．

⑥ 半自動合成システムはこの作成されたモジュールをライブラリに登録する．

この後ユーザは（必要ならば）さらに詳細化をすすめる（補助モジュールを作成する）べく，このサイクルを繰り返す．

(a) New Specification (by TELL/NSL)

```
Queen Q1 and queen q2 are
                on the same column of arrangement x
 means that
 1) The x_coordinate of the position of q1 in x is
    the x_coordinate of the position of q2 in x.
end
```

(b) Translated Logical Formula of (a)

```
∀q1,q2, x [on_the_same_column [q1,q2, x ] ⇔
            x_coordinate [position [q1,x] ]
              = x_coordinate [position [q2,x] ] ]
```

(c) Retrieved Module Name

```
on_the_same_row
```

(d) Translated Logical Formula of (c)

```
∀q1,q2, x [on_the_same_row  [q1,q2, x ] ⇔
             y_coordinate [position [q1,x] ]
               = y_coordinate [position [q2,x] ] ]
```

(e) Assumption of the Equivalence

```
∀ x [x_coordinate [ x ] =y_coordinate [ x ] ]
```

(f) Program of (c) (by Prolog)

```
on_the_same_row(Q1,Q2,X) :-
      position(Q1,X,P1),
      y_coordinate(P1,X),
      position(Q2,X,P2),
      y_coordinate(P2,X).
```

(g) Modified Program of (a)

```
on_the_same_column(Q1,Q2,X) :-
      position(Q1,X,P1),
      x_coordinate(P1,X),
      position(Q2,X,P2),
      x_coordinate(P2,X).
```

Fig. 2 A sample of program synthesis

6．おわりに

仕様として与えられた一階述語論理式の同値性を証明することで，再利用可能な既存のソフトウェアモジュールを検索する手法について述べた．この手法は，仕様が一階述語論理式で規定されるようなプログラミング環境であれば，インプリメンテーション言語が何んであれ適用できる．

本手法を用いても，利用者が考えている仕様と論理的に等しい仕様を持つ既存のモジュールすべてを検索できるわけではない．また逆に，検索したモジュールの仕様が新規の仕様と必ずしも等しいわけではない．それは主に，二つのモジュールの等しさを判定する際にそれらの補助モジュールやそれらが操作するデータ構造の仕様を参照しないためである．しかし，ソフトウェア開発において，既存のモジュールの再利用を促進するためには，できる限り早い段階で流用可能なモジュールを判定できることが望ましく，詳細なレベルまで完全に記述しなくとも検索が行なえる必要がある．そこで本手法では，参照しているモジュール（補助述語やデータクラス）の仕様まで含めた論理的な同値性を証明するのでは

なく，言わばその「参照形式」が似ていれば良いものとしている．これは，仕様における参照の形式が似ていれば，そのインプリメンテーションであるプログラム上でのインタフェースも似ていて，容易に流用可能であろうという考え方に基づいている．本手法では，補助述語の仕様を用いずに，その呼出し方，引数の与え方のみからそれらの等しさを判断し，またデータ構造に関する帰納法を用いずに，再帰呼出しの形式だけから再帰的な等しさを判断する．ソフトウェアの設計者は段階的詳細化の各ステップにおいて，本手法を用いて再利用可能な部分は既存のものを流用し，そうでない部分のみさらに詳細化してゆくことができる．

［謝辞］

本研究は，第５世代コンピュータプロジェクトの一環として行われた．本研究の機会を与えて下さった，新世代コンピュータ技術開発機構の方々に深謝いたします．

また，ＴＥＬＬについて御指導いただいた東工大榎本教授（現国際情報社会研究所長），米崎助教授，佐伯助手に深謝いたします．

［参考文献］

［１］ Enomoto,H., et.al., NATURAL LANGUAGE BASED SOFTWARE DEVELOPMENT SYSTEM TELL, ICOT TR-067, 1984.

［２］ Chang,C., et.al., Symbolic Logic and Mechanical Theorem Proving, Academic Press, 1973. （邦訳，長尾　他，コンピュータによる定理の証明，日本コンピュータ協会，1983.）

［３］ Sugimoto,M., et.al., Design Concept for Software Development Consultation System, ICOT TR-071, 1984.