ソフトウェア工学 77－13
（１９９１．２．７）

プログラムの再利用を支援するプログラミング環境の作成について
－プログラムの変形を利用した支援手法－

永澤勇一，今中　武†，荻野貴之，永井隆弘†，
江澤義典，三根　久，豊田順一†
（関西大学，†大阪大学産業科学研究所）

プログラムの再利用を支援するプログラミング環境PERC (Programming Environment to Reuse program Components) の作成について述べる．PERCは，Prologを対象言語として，利用者自身が再利用可能なプログラムモジュールを蓄積し，少人数のグループ等で再利用するための支援を行う．PERCは，現在９種類のツールを結合することにより実現しているが，本報告ではこのうちプログラムの蓄積を支援するツール，および検索されたプログラムの修正を支援するツールを中心に説明を行う．これらのツールでは，プログラムの一部分に対するの書き換え（プログラムの変形）がプログラムの仕様に対してどのような影響を与えるかを定義した変形オペレータを利用しており，変形オペレータについても説明する．

A Design of Programming Environment to Reuse Program Components

Yuichi NAGASAWA, Takeshi IMANAKA†, Takayuki OGINO,
Takahiro NAGAI†, Yoshinori EZAWA, Hisashi MINE,
and Jun'ichi TOYODA†

KANSAI University
†The Institute of Scientific and Industrial Research, OSAKA University

This paper describes our newly proposed programming environment PERC(Programming Environment to Reuse program Components). PERC consists of nine tools, and provides an environment to share reusable program components among users belonging to same project, group, etc. In this paper, we firstly describe outlines of PERC. Secondly, we define transformation of programs assist users with to modification of reusable program components. Finally, we descuss implementation of some tools based on the transformation.

1．はじめに

ソフトウェアの生産性を向上させるために，モジュール化したプログラムを部品として蓄積し，再利用することは非常に有効である．しかしながら，プログラムの再利用を効率よく行うためには，再利用を支援するためのプログラミング環境を整備する必要がある[1]．本報告ではPrologを対象言語として，プログラムの再利用を支援するプログラミング環境PERC(Programming Environment to Reuse program Components)について述べる．本報告で扱うプログラムの再利用は，大学や企業の研究室において利用者自身が再利用可能なプログラムモジュールを蓄積し，各研究グループ等で共有するといった考え方に基づいている．PERCでは，1)プログラム蓄積に対する支援，2)蓄積されるプログラムの統一的管理，3)プログラム検索に対する支援，4)検索したプログラムの修正に対する支援などの実現を目指している．環境の作成については，これらの支援機能を持った複数個のツールを開発し，ネットワーク環境下で結合させることにより実現する．本報告では，2),3)については概要を述べるにとどめ，1),4)を実現するツールについて中心的に説明する．

一般に，再利用プログラムを部品として利用する場合，組み込むプログラムに合わせて修正を施さなければならない．ただし，プログラム中に記述されている処理の中心的な部分に対して修正する場合，プログラムを根本的に書き直すこととほとんど同等であり，筆者らは再利用の対象から除外する．したがって，再利用時に行う修正の対象は処理の細部についてのみと考える．PERCの作成においては，プログラムに対する変形操作を定義し，プログラムの細部に対する修正を支援するツールの開発を行っている．また，PERCではプログラムを蓄積する際に，修正の対象とならない中心的な処理を記述した部分のみを蓄積し，利用する際に細部の修正を変形操作により行わせるようにしている．このため，1)プログラム蓄積時の仕様記述は，処理の中心的部分に対してのみ行えばよく，蓄積を行う際の負担を軽減することができる，2)蓄積されるプログラムの量を減少させることができる，3)再利用の範囲を拡大することができるなどの利点が生じる．以下，2章ではPERCを構成するツール群の概要を示す．3章では，プログラムの変形操作について具体的に述べ，4章ではプログラムの変形を利用した支援ツールについて述べる．5章においては，まとめと今後の課題を示す．

2．プログラム再利用を支援する環境

筆者らが現在作成しているプログラミング環境PERCを図1に示す．本環境は，表1に示すツール群を結合することにより構成される．

データベース管理ツール，データベースサーバは，共に再利用可能プログラムの管理を行うためのツールである．データベース管理ツールは，既存のデータベースシステムUNIFYを用いて実現しており，UNIFYの持つデータベース管理機能を用いてプログラムの管理を行っている．データベースサーバは，1つのデータベースを複数のワークステーションからアクセスできるようにするためのものであり，ローカルエリアネットワークを介して検索ツール[2]との通信を行う．

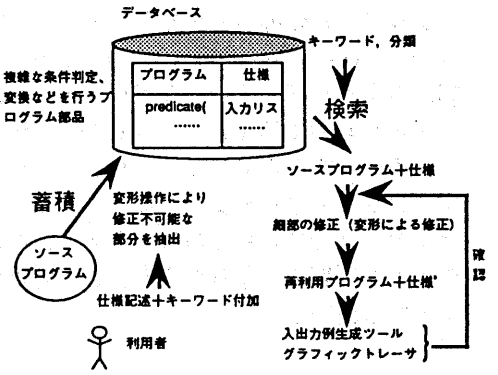

図1　プログラムの再利用を支援する環境

表1　PERCを構成するツール群

ツール	ツールの行う処理
プログラム編集時の検索ツール	プログラム編集中にデータベースの検索を行うためのツール（検索用クライアント含む）
ウィンドウ環境の検索ツール	PSI IIからデータベースの検索を行うツール（検索用クライアント含む）
データベースサーバ	データベースの検索結果をネットワークを通じて返送するツール
データベース管理ツール	プログラムの統一的管理を行うためのツール
蓄積用ツール	蓄積するプログラムを基本構造へ変換する。定義したプログラム変形操作により変形不可能な部分を抽出し、その部分のキーワードおよび仕様の入力を求めるツール
修正支援ツール	利用者の要求に合わせるためのプログラムの修正を支援するツール
入出力例生成ツール	修正を行ったことによる特徴的な振舞を示す入出力例を生成し、利用者に提示するツール
グラフィックトレーサ	プログラムの動作を視覚的に確認するためのツール
プログラム仕様生成ツール	変形操作などによる仕様の変化に基づきプログラム仕様の一部分を自動生成するツール

再利用プログラムを検索するためのツールはSunワークステーション上で稼動するものと，逐次推論型マシンPSIII上で稼動するものがある．これらのツールは，共に再利用プログラムの検索を支援するツールである．たとえば，利用者が再利用プログラムを必要とした場合，これらのツールを起動すれば，必要とするプログラムの処理に関していくつかの質問が行われる．利用者が質問に答えると，データベースサーバに検索要求が送られ，要求を満たすプログラムを提供するために必要なプログラム部品が検索される．プログラム部品が検索されると，部品を典型的と思われるプログラムの構造（以降では，基本構造と呼ぶ）と組み合わせる．そのプログラムの仕様をプログラム仕様生成ツールが生成し，プログラムと共に提示する．基本構造は，筆者らが処理の種類に合わせて定義したものであり，たとえば，リスト処理のテールリカージョン，グローバルなデータを扱う処理のfailループなどがある．また，このように基本構造にするのは，特殊な処理をできる限り少なくし，利用者の理解を容易にするためである．

利用者は，検索ツールにより提示された仕様に対し，細部に至るまで要求を満たすものかどうかチェックする．この時，要求に合わなければ，自分の要求に合うようにプログラムの修正を行う．プログラムの修正は，修正支援ツールを用いて行う．このツールは，変形操作を行うためのオペレータ（以降では簡単に変形オペレータと呼ぶ）を利用者に提供することによりプログラムの修正を支援する．変形オペレータとは，プログラムの細部を変更することにより，処理内容（仕様）がどのように変化するかを対応づけルール化したものである．

さらに，利用者が修正したプログラムが実際に要求通りであるかを確かめるために，PERCでは入出力例生成ツールとグラフィックトレーサを提供している．入出力例生成ツールは，プログラムの処理内容（仕様）を顕著に表す入出力例を生成し，利用者に提示するツールである．また，グラフィックトレーサは，視覚的にプログラムの動作確認を行うためのツールである．この3つのツールを組み合わせて用いることにより，プログラムの修正をより効率よく行えるように支援する．

蓄積用ツールは，プログラムの蓄積を支援するツールである．利用者が蓄積しようとしているプログラムに対して，変形オペレータを適用し，基本構造に変形する．プログラムを基本構造に変形した後は，再利用時の修正を行わない中心的な処理部分に対して，利用者に仕様記述，処理の分類，キーワードの入力を求める．処理の分類は，条件判定，データ検索処理，変換処理（入力に対して，何らかの出力を求める），入力処理，出力処理などである．キーワードと処理の分類は，検索時のキーとして用いる．また，利用者が処理の種類を条件判定と入力した場合，条件を満足する例，満足しない例をそれぞれ入力させる．これらの例は，入出力例生成ツールで用いる．

3．プログラムの変形

変形オペレータは現在，テールリカージョンによるリスト処理プログラム，生成検査法によるプログラム，failループによるプログラムに対してそれぞれ定義している．本章では，リスト処理プログラムにおける変形操作について述べる．ここでリスト処理プログラムとは，入力がリスト形式のPrologプログラムを指す．

○基本構造と引数操作パターン

Prologのリスト処理プログラムは，大多数がテールリカージョンと呼ばれる手法を実現する類似構造で構成されている[3]．筆者らは，これらの類似構造から共通部分を抽出し，最も典型的と思われる構造を1つ定義し，「リスト処理の基本構造」（以降では，簡単のために単純に基本構造と呼ぶ）とした．基本構造は図2に示すように階層的な部分構造から構成されている．

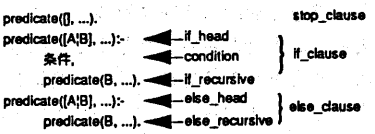

図2　プログラムの基本構造

また，Prologのリスト処理プログラムにおいては，構造のみでなく，引数部分についても類似したパターンがよく用いられる[3]．引数操作パターンは，これらの類似パターンから取り出した典型的なパターンであり，リスト要素の削除，取り出しなど，処理操作の種類を規定する．引数操作パターンは，リスト処理の基本構造における stop_clause，if_head，if_recursive，else_head，else_recursive部分の引数のみから定義される．

○変形オペレータ

変形オペレータは，適用する対象に応じて構造に関する変形オペレータと引数操作に関する変形オペ

レータに分けられる．構造に関する変形オペレータは，基本構造の各部分を削除したり，追加したりするものである．たとえば，基本構造に対してif_recursive部分を削除するcut_if_recursiveオペレータを適用すると

（プログラム１）

```
predicate([], ...).
predicate([A¦B], ...):-
        条件.
predicate([A¦B], ...):-
        predicate(B, ...).
```

のように変形される．プログラム１は，基本構造が条件を満たすすべての要素に何らかの処理を行うのに対し，リストの先頭から走査して条件を満たす要素を１つだけ処理するものである．ただし，変形オペレータを適用した場合，引数が書き換えられる場合がある．たとえば，cut_if_recursiveの適用によって行われる引数の書き換えは，まずif_head部分に出力リストがあれば，そのリスト中にある変数で，if_recursive部分の出力リストと同じ変数を探し出す．次に，探し出した変数をif_recursive部分の入力リストに割り当てられていた変数に書き換える．このような，変形オペレータの変形操作，および変形に伴う引数の書き換えは，組み合わされる引数操作パターンや他の変形オペレータにほとんど影響されずに適用できる．このため，各変形オペレータは自由に組み合わせることが可能である．

次に，引数操作に関する変形オペレータについて示す．条件部分における比較の対象が入力として引数に与えられる場合がある．たとえば，引数に入力された値よりも小さな値を持つリスト要素を先頭から走査して１つだけ削除するプログラムなどがある．この場合，引数の数を増加させる変形オペレータadd_argumentを用いることができる．

以上のように，引数操作パターンと変形オペレータの適用によって，基本構造を持つプログラムを互いに仕様の異なる多種のプログラムに変形することができる．実際には，各引数操作パターンには，それぞれのパターンによって行われる処理を表した仕様部品が対応づけられている．また，変形オペレータは，変形を行うことによってプログラム仕様にどのような修正が加えられるのかを表した仕様変換規則を対応づけて定義している．たとえば，変形オペレータcut_if_recursiveを利用した場合，対応する仕様変換規則に従い仕様中にある「すべての要素に対して処理を行う」といった表現を「先頭から走査して条件を満たす要素１つだけに対して処理を行う」に変更を行う．現在，引数操作パターンを７つ，変形オペレータを17個定義し，４章で述べるツールに組み込んでいる．

４．プログラムの変形を利用した支援ツール

３章で述べたように，Prologのリスト処理プログラムにおいては，プログラムの構造や引数に対する変形と変形による仕様の変化を定義することが可能である．このため，典型的な構造を持つプログラムのみ蓄積しておけば，利用時に変形を行うことにより多くの場合に利用可能なプログラムを提供することができる．本章では，蓄積用ツールと修正支援ツールについて示す．特に修正支援ツールについては，別のツールである入出力例生成ツール，グラフィックトレーサとの関わりも述べる．

４．１　蓄積用ツール

以下のプログラムを蓄積する場合について述べる．プログラム２は，第１引数に動詞のリストが入力され，その動詞をすべて過去形に変換し，第２引数に出力するプログラムである．

（プログラム２）

```
predicate([], []).
predicate([A¦B], [P¦D]):-
        verb_inflection(A, P, PP),
        predicate(B, D).
```

プログラム２のif_clause内で呼び出されている述語verb_inflectionは辞書検索のプログラムモジュールである（プログラム２ではverb_inflectionの定義を省略している）．verb_inflectionに対しては，特殊な処理を多く含むため変形操作を行うことはできない．したがって，verb_inflectionについては，利用者に仕様記述[1]，処理の分類，キーワードの入力を求める．これに対し，verb_inflectionを除いた部分は，リスト処理の基本構造から変形操作により生成可能であり，生成に必要となる変形オペレータと仕様変

[1]本ツールの扱う仕様記述は，「入力，出力の組み合わせ」，「リスト，アトムなど，代入が許される変数のタイプの組み合わせ」，「述語内の変数が何を表したものかを示したものの組み合わせ」，「その述語がどのような処理で用いられるのかを表現したもの」の４つから構成される．

換規則を用いれば，「入力リストのすべての要素を述語verb_inflectionの出力（第2引数）で書き換えを行い出力引数に与える」[2]であることが判明する．

また，蓄積用ツールでは，利用者に対して仕様記述，処理の分類，キーワードの質問を行う．仕様記述の入力を効率的に行わせるために，仕様記述を構成する「入力，出力の区別」，「リスト，アトムなど，変数に代入が許されるタイプ」，「変数の内容」，「どのような処理を行うか」などに関する質問を順に行う．処理の分類は，出力処理，条件判定など，変形不可能な述語がプログラム内でどのような役割を担っているかを示したものである．蓄積用ツールでは，引数の入力，出力の組み合わせを根拠にして処理の分類を絞り込み，利用者に提示するようにしている．たとえば，verb_inflectionのように入力，出力の両方を含んだ述語呼び出しに対しては，データの変換，データの検索のいずれかであることを示す．利用者はこのような指示に従い，選択を行う．また，利用者が処理の分類を”条件判定”とした場合に限り，次節で述べる入出力例生成ツールで用いるために，条件を満たす要素，および満たさない要素をそれぞれ5例ずつ入力させる．キーワードの入力は利用者の属している研究グループなどで用いている用語が利用される．また，入力の際にはできる限りすでに使用されているキーワードを利用させるためにこれらの提示を行う．

プログラム2のverb_inflection(A,P,PP)を例に説明する．verb_inflection(A,P,PP)は，述語呼び出しであり，かつその定義は，本枠組みで変形不可能である．したがって，修正不可能な部分であると判断され，仕様記述，処理の分類，キーワードの入力が求められる．仕様記述は，”verb_inflection(入力，出力，出力)”，”verb_inflection(アトム，アトム，アトム)”，”verb_inflection(動詞の原形，動詞の過去形，動詞の過去分詞)”，”英単語の動詞を活用する”のように入力される．また，verb_inflection(A,P,PP)は，引数に入力と出力の両方を含んでおり，蓄積用ツールにより「verb_inflection(A,P,PP)の処理の分類は，”データの変換”または”データの検索”である」と表示され，利用者は，verb_inflectionの場合データ変換であると入力する．次にキーワードの入力が求められ，”動詞の活用”のように入力が行われる．

4．2　修正支援ツール

PERCでは，検索ツールを用いて利用者の入力したキーワード，あるいは処理の分類を検索キーとすることにより，プログラムを検索することができる．検索されるプログラムは複雑な条件判定，データ変換を行う部分であり，利用者には基本構造に埋め込んだプログラムを示し，そのプログラム仕様[3]も基本構造の持つ仕様部品と組み合わせて示す．たとえば，”リスト処理”，”動詞の活用”，”書き換え”を指定した場合は，動詞の活用というキーワードの付けられているverb_inflectionが検索され，さらに基本構造に埋め込まれて，以下のようにプログラムが表示される．

（プログラム3）　　プログラム仕様

```
predicate([], []).
predicate([A¦B], [P¦C]):-
        condition,
        verb_inflection(A, P, PP),
        predicate(B, C).
predicate([A¦B], [A¦C]):-
        predicate(B, C).
```

入力リストの条件 condition を満足する要素すべてを動詞の過去形に書き換え、結果を出力リストに与える。
入力リストが空リストの場合は、出力リストに空リストを与える。
出力リストの要素の順序は、入力リストの要素の順序に従う。

次に，修正支援ツールは利用者に対し，処理を行う際の条件の有無を確認する．これは，検索時の利用者の入力に条件に関する指示がないのに対し，リスト処理プログラムの基本構造には，condition部分があるためである．”条件あり”と利用者が入力した場合，本ツールは検索ツールを呼び出し，データベース中に蓄積されているすべての条件部品に付加された仕様を検索し，その結果を利用者に提示し，選択を求める．たとえば，「動詞のチェック」といった仕様を持ったverbという条件がデータベース中にあり，利用者がverbを選択した場合，プログラム3のcondition部分が書き換えられ，プログラム4のようになる．また，条件が単純な組み込み述語を

（プログラム4）　　プログラム仕様

```
predicate([], []).
predicate([A¦B], [P¦C]):-
        verb(A),
        verb_inflection(A, P, PP),
        predicate(B, C).
predicate([A¦B], [A¦C]):-
        predicate(B, C).
```

入力リストの条件1を満足する要素すべてを動詞の過去形に書き換え、結果を出力リストに与える。
（条件1）
入力が英単語の動詞であるかチェックする。
……

[2]実際に生成された仕様はさらに複雑であるが，説明のため本文中では重要な記述のみを示す．
[3]プログラム仕様とは，プログラムが入力データに対してどのような出力を行うのかを自然語の並びで表した仕様．

用いて定義できる場合，利用者にソースコードの入力を求める．たとえば，利用者が条件として「60未満」を希望していれば，利用者は" 60>要素" のコードを入力し，同様にcondition部分の書き換えが行われる．

また，利用者がverbなどの条件判定を処理する部品を指定した場合，蓄積時に条件を満足する例と満足しない例が付加されており，入出力例生成ツールを起動すれば，これらの例を用いて入力例が生成される．たとえば，このプログラム４では，条件verbを満たすリスト要素すべてを過去形に変換するため，条件を満たす動詞を複数個含む[tennis, play, ball, read, run, racket, work, walk]のような入力例が生成される[4]．生成した入力例を用いてプログラムを仮実行し出力[tennis, played, ball, read, ran, racket, worked, walked]を得る．その際に，利用者はグラフィックトレーサを利用することにより，プログラム動作の確認を容易に行うことができる．

修正支援ツールは，要求が完全に満足されているかどうか利用者に確認し，満足されてない場合，さらに変形オペレータの一覧をメニュー表示する．このメニューに対し，たとえば「条件を満たす最初の要素のみを処理する」といった項目が選択されると，修正支援ツールは変形オペレータを適用し，プログラムの修正を行う．プログラム４に対し上記の修正を行った例をプログラム５に示す．

（プログラム５）

```
predicate([], []).
predicate([A|B], [P|B]):-
    verb(A),
    verb_inflection(A, P, PP).
predicate([A|B], [A|C]):-
    predicate(B, C).
```

プログラム仕様

入力リストの条件１を満足する要素を先頭から走査して１つだけ動詞の過去形に書き換え、結果を出力リストに与える。
（条件１）
入力が英単語の動詞であるかチェックする。
.....

このプログラムに対しても，入出力例生成ツールを用いれば，入出力例が生成され，動作確認が行える．さらに，グラフィックトレーサを用いる場合，利用者自身も入力例を手入力でき，[he, want, to, play, tennis], [to, play, tennis, is, fun]などいろいろな入力例をトレーサ上で確認できる．しかしながら，後者の入力例に対しては，[to, played, tennis, is, fun]といったように望ましくない結果が出力される[5]．この場合，to不定詞の後ろに続く動詞は原形のままでよいといった条件を付加する必要があり，修正支援ツールを再び用いて変形オペレータの一覧から，" 条件を満たす要素の前（後ろ）に対する操作" という項目を選ぶ．さらに，リスト要素がtoと等しいかどうかを判断するための条件が必要となる．これに対しては，再利用可能な条件判定が蓄積されていれば検索を再び行い，なければ条件となるプログラムコードを手入力する．この場合，修正支援ツールに" 要素¥== to"，" 追加条件はand" のように入力すれば，プログラム５はプログラム６に変形される．

（プログラム６）

```
predicate([], []).
predicate([A, B|C], [A, P|C]):-
    verb(B),
    A ¥== to,
    verb_inflection(B, P, PP).
predicate([A|B], [A|C]):-
    predicate(B, C).
```

プログラム仕様

入力リストの条件１を満足し、条件２を満足する後ろの要素を先頭から走査して１つだけ動詞の過去形に書き換え、結果を出力リストに与える。
（条件１）
入力が英単語の動詞であるかチェックする。
（条件２）
入力がtoと等しくない。
.....

５．おわりに

Prologプログラムの再利用を支援する環境PERC (Programming Environment to Reuse program Components)の作成について述べた．特に，プログラムの蓄積用ツール，および修正支援ツールについて述べた．さらに，これらのツールで用いているプログラムの変形について具体的に説明した．

今後の課題として，1)引数操作パターン，変形オペレータの拡張，および検討，2)ツールの結合方法に対する検討および評価などがある．

[参考文献]

[1]宗近，小尾，蓮田，松村:"部品指向の設計支援環境50SM－部品化・再利用支援機能の強化－"，情報処理学会研究報告，SE64-14, pp.105-112 (1989).
[2]永澤，今中，上原，三根，豊田:"現実関係モデルを導入した自然語による検索キーの取り扱い"，情報処理学会，第41回全国大会，4G-2 (1990).
[3]今中，上原，豊田:"プログラムの類推定義のためのネットワーク表現"，情報処理学会，第38回全国大会，7L-3 (1989).

[4] verb(A)の蓄積時に，その条件を満たす例と満たさない例として，満たす例：" [play, read, run, work, walk]"，満たさない例：" [tennis, ball, recket, pen, book]" のように入力されたものとしている．
[5] 英文を過去形にするといった目的で作成している場合を想定している．