広島大学学術情報リポジトリ
Hiroshima University Institutional Repository

Title	Prologを対象としたソースプログラムからのプログラム仕様自動生成 : 構造と引数操作パターンに基づく変形解析法の提案
Auther(s)	永井, 隆弘; 今中, 武; 永澤, 勇一; 平嶋, 宗; 上原, 邦昭; 豊田, 順一
Citation	電子情報通信学会論文誌 D , J76-D2 (12) : 2586 - 2596
Issue Date	1993-12-25
DOI	
Self DOI	
URL	http://ir.lib.hiroshima-u.ac.jp/00045768
Right	Copyright (c) 1993 IEICE
Relation	

論 文

Prologを対象としたソースプログラムからのプログラム仕様自動生成 ——構造と引数操作パターンに基づく変形解析法の提案——

正員 永井 隆弘[†*] 正員 今中 武[†*] 非会員 永澤 勇一[†**]
正員 平嶋 宗[†] 正員 上原 邦昭[†***] 正員 豊田 順一[†]

A Method for Generating Program Specification from Source Program —— Analysis by Transforming Program Structure and Argument Manipulation ——

Takahiro NAGAI[†*], Takeshi IMANAKA[†*], *Members*, Yuichi NAGASAWA[†**], *Nonmember*, Tsukasa HIRASHIMA[†], Kuniaki UEHARA[†***]
and Jun'ichi TOYODA[†], *Members*

あらまし 本論文では，ソースプログラムからプログラム仕様を自動生成するための変形解析法を提案し，この手法を実装したシステムAPSG/I (Automatic Program Specification Generator I) について述べている．変形解析法では，Prologのリスト処理プログラムの類似性に注目し，類推の枠組みを利用することによって，典型的なリスト処理プログラムとその仕様から与えられたプログラムに対する仕様の生成を行っている．まず，プログラムが入力されると，あらかじめ定義した典型的なプログラムと比較して，①命令の有無や実行順序などで規定される構造，②データの受け渡しなどで規定される引数の組合せ，の2点について差異を求める．次に，求めた差異によって，あらかじめ定義したプログラム仕様を変換し，入力されたプログラムの仕様を生成する．変形解析法で生成される仕様は，①プログラム仕様と，仕様を変換する規則を自然語で定義しているため自然言語文になる，②形式が統一され，かつ細部に関する情報についても十分に含んでいるなどの特徴を有している．

キーワード プログラム仕様生成，Prolog，プログラム解析，リスト処理プログラム

1. まえがき

近年，ソフトウェア生産性の向上を目指して，プログラムの部品化，再利用の研究が盛んに行われている[(2),(8),(11)]．特に，部品を組み合わせて新たなプログラムを合成する手法については，多くの研究成果が報告されている．しかしながら，部品の蓄積に関しては，文献(12)で準備するべき部品とその完備性について述べられているのみで，詳細な議論はあまり行われていない．これは，多くの研究が事務処理，データベース検索など処理のパターンが予測しやすい分野を対象としており，あらかじめシステム作成者がすべての部品を準備するのが一般的になっているためである．

一方，大学の研究室，企業の研究所などにおいては，開発されるプログラムが多種にわたっており，あらかじめ部品を準備することは困難である．しかしながら，研究におけるプログラム開発においても，①ライブラリ，組込み関数といった既存のプログラムを多用する場合がある，②別のプログラマが同じ仕様のサブルーチンを既に作成していたことに気づくことがあるなど，プログラムの規模は小さいが複数の利用者で共有，再利用できるプログラムが数多く存在すると考えられる．実際に，筆者らは数個の代表的なリスト処理プログラム，簡単な英文入力を処理するプログラムなど，小規模プログラムをC-Prologインタプリタ内に新たに組み

† 大阪大学産業科学研究所，茨木市
The Institute of Scientific and Industrial Research, Osaka University, Ibaraki-shi, 567 Japan
* 現在，松下電器産業株式会社
** 現在，三菱電機株式会社
*** 現在，神戸大学工学部情報知能工学科

込み，多数の利用者で共有，再利用している．以上のことから，ソフトウェアの生産性を向上させるために，再利用可能プログラムをデータベース化し(以降では，再利用プログラムベースと呼ぶ)，プログラマが自由にプログラムの蓄積，利用を行えるようにすることが有効であると考えられる[6]．

再利用プログラムベースを構築する場合，新たな問題としてプログラムを蓄積する際に，プログラム作成者が独自で仕様記述しなければならないといった点が挙げられる．更に，記述する仕様は他のプログラマが再利用することを想定し，(1)理解容易である，(2)形式が統一されている，(3)細部についても記述もれがないなどの性質を備えていなければならない．従って，プログラム作成者にとっては，仕様を記述すること自体が非常に困難な作業になる．このような問題点を解決するための一つの方法として，ソースプログラムから上記の三つの性質を備えた仕様を自動生成することが考えられる．

本論文では，プログラム仕様自動生成の一つの試みである変形解析法について述べる．変形解析法では，与えられたソースプログラムと類似したソースプログラムのもつ仕様を利用して，与えられたソースプログラムの仕様を生成する．この方法はCarbonellの提案した変形類推の枠組み[1]を利用したものとなっており，プログラムの記述から仕様記述を生成するといった，プログラム自動生成の逆操作に相当する困難な変換を行うのではなく，プログラム間，仕様間の変換のみで必要な仕様を生成できるようになっている．しかしながら，一般に前述の三つの性質を備えた仕様の生成を保証することは必ずしも容易ではない．本論文では限定された領域において本手法の具体化を行い，本手法によって生成された仕様を実験によって評価することにより，本手法の有効性を示している．

現在のところ本手法で対象としているプログラムは，

- 言語：論理型言語Prolog
- データ構造：リスト
- 制御構造：再帰構造

である(以降では，このようなプログラムをリスト処理プログラムと呼ぶ)．Prologを対象に選んだ理由は，①モジュラリティが高く，他人の作成したプログラムを再利用できる可能性が高いこと，②Prologプログラムが宣言的に記述されるため，制御に関する情報を含まないプログラム仕様の生成と整合性がよいこと，などを考慮したためである．上記のデータ構造と制御構造を対象に選んだ理由は，Prologプログラムで非常によく用いられていることを考慮したためである．また，本手法ではバックトラックにより別解を生成するようなプログラムは含めていない．

生成されるプログラム仕様は，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言語文で表した仕様であり，プログラムの実行効率などの内部仕様は，①Prologのモジュラリティの高さから再利用プログラムを小規模にすることができ，実行効率が検索時に重要となることは少ないと考えられる，②実行効率などの扱いは入力データに依存したり，高度な数学的知識が必要になるなど変形解析法の枠組みでは困難である，などの理由からプログラム仕様には含めていない．現在，変形解析法を実装したプログラム仕様自動生成システムAPSG/I (Automatic Program Specification Generator I)を作成しており，APSG/Iを用いて変形解析法の有効性を確認するために，変形解析法の①適用可能な範囲を示すこと，②生成したプログラム仕様が細部に関する情報を十分に含んでいることの確認，③生成したプログラム仕様が人間にとって理解容易であることの確認の3点についての実験を行った．その結果，少なくとも教科書レベルのPrologのリスト処理プログラムでは，変形解析法が適用可能であり，生成したプログラム仕様は形式が統一されているだけでなく，理解が容易で，かつ細部に至るまで十分に情報を含んでいることが示された．

2. 類推を用いた仕様生成の枠組み

Prologのリスト処理プログラムは，文献(10)で報告されているように再帰処理に伴う入力リストの取扱いが類似しているため，多くのプログラム間で多種の類似部分を共有している．特に，末尾再帰といった構造は，多数のリスト処理プログラムに共通する構造としてとらえることができる．変形解析法は，このようなプログラム構造をもっているリスト処理プログラムを対象とし，Carbonellによって提案された変形類推の枠組み[1]を利用することによって，自然言語仕様を生成する手法である．Carbonellの変形類推は，過去に行った問題解決の解を，現在の問題解決に利用しようとするもので，現在の問題の仕様を満たすように過去の問題解決に対する解に修正を加えることにより，所望の解を生成するものである．変形解析法は，「プログラムが似ているなら，仕様も似ている．」という考えをもとにしており，類似したプログラムの仕様に修正を加え

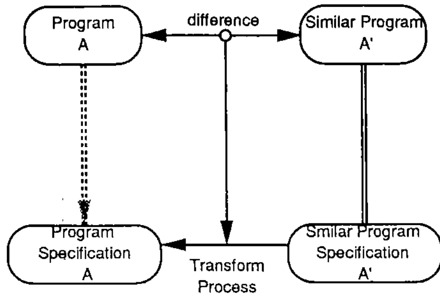


図1 変形解析法による仕様生成過程
Fig. 1 The process of generating specification.

ることによって，所望のプログラムに対する仕様を生成する手法である．図1が変形解析法による仕様生成の過程である．

リスト処理プログラムでは，再帰処理に伴う入力リストの取扱いに類似性が見られるため，リスト中の要素の取扱いに関していくつかのパターンに分類することができる．変形解析法では，このようなパターンごとに典型的なプログラム(以降では，基本プログラムと呼ぶ)を類似プログラムとして用意している．更にこれらの基本プログラムに対して仕様を与えておく．

仕様生成に利用する類似プログラムの選択は，与えられたプログラムがどのパターンに属するかを求めることによって行っている．ここでは，与えられたプログラムを基本プログラムのいずれかに対応づけるために変形オペレータによるプログラム変形を用いる．変形オペレータはあるプログラムの一部分に変更を加えることによって，他のプログラムを生成するオペレータであり，これらはプログラムを分析した結果により，なるべく多くのプログラムに利用可能な基本的なものを用意している．これによって与えられたプログラムとあらかじめ用意した基本プログラムとが対応づけられる．またこの際に用いられた変形オペレータは類似プログラムとの差を表現したものとなっている．

類似プログラムの仕様を与えられたプログラムの仕様として利用するためには，プログラム間の差に着目して仕様を修正する必要がある．仕様の変換には仕様変換規則を用いて行われる．仕様変換規則は，各変形オペレータによってプログラムの変形を行った場合に，仕様に対して行うべき変換を規則化したものである．仕様変換規則の選択は，プログラムの差を表現している変形オペレータと仕様変換規則を1対1に用意しているため，プログラムに適用された変形オペレータに対応づけられた仕様変換規則を用いることに対応している．

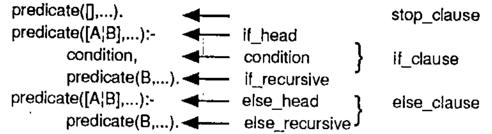


図2 プログラムの基本構造
Fig. 2 Basic program structure.

3. 変形解析法によるプログラムの解析と仕様生成

変形解析法による仕様の生成では，まず与えられたプログラムを既存の基本プログラムと対応づけるためにプログラムの変形を行う．変形操作には変形オペレータが用いられ，与えられたプログラムが基本プログラムと一致するまで繰り返し適用される．更に，対応づけられた基本プログラムに用意された仕様に対して修正を行う．修正にあたっては，プログラムの変形の際に用いられた変形オペレータに対応した仕様変換規則が用いられる．この一連の処理によって与えられたプログラムに対する仕様が生成される．本章では，まず変形解析法で用意した部品に関して述べる．次にAPSG/Iによる仕様生成例について述べる．

3.1 基本プログラムと仕様の表現

Prologのリスト処理プログラムは，大多数が末尾再帰と呼ばれる手法を実現する類似構造で構成されている(9),(10),(13)．変形解析法では，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一つ定義し，基本構造と呼んでいる(図2)．図2に示すように，基本構造は階層的な部分構造から構成されている．実際には，プログラムの述語名はプログラマが自由に設定するものであるために，基本構造では仮の述語名を "predicate" としている．

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，リスト中の要素の取扱いを表現した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(9),(10)．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数個の典型的なパターンであり，リスト要素の削除，取出しなど，リストの要素に対する処理操作の種類を規定する(表1)†．引数操作パターンは，図2の基本構造における stop_clause, if_head,

† 表1，2で示す引数操作パターンおよび変形オペレータはすべて文献(9)，(10)で調査した結果に基づき定義したものである．

表1 引数操作パターン

<table>
<tr><th rowspan="2">パターン名</th><th rowspan="2">パターンの持つ意味内容</th><th colspan="5">引数の組み合わせ</th></tr>
<tr><th>stop_ clause</th><th>if_ head</th><th>if_ recursive</th><th>else_ head</th><th>else_ recursive</th></tr>
<tr><td>check_pattern</td><td>入力リストに対して条件を満足するか調査する。</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td>count_pattern</td><td>入力リストの条件を満足させる要素の数を数え、結果を出力リストに与える。</td><td>0</td><td>N1 (N1 ← N+1)</td><td>N</td><td>N</td><td>N</td></tr>
<tr><td>delete_pattern</td><td>条件に従って、入力リストから要素を削除して、結果を出力リストに与える。</td><td>[]</td><td>C</td><td>C</td><td>[A|C]</td><td>C</td></tr>
<tr><td>fetch_pattern</td><td>入力リストから条件を満足する要素を取り出し、取り出した結果を出力リストに格納する。</td><td>[]</td><td>[A|C]</td><td>C</td><td>C</td><td>C</td></tr>
<tr><td>insert_pattern</td><td>条件に従って、入力リストに要素を挿入して、結果を出力リストに与える。</td><td>[]</td><td>[A,H|C]</td><td>C</td><td>[A|C]</td><td>C</td></tr>
<tr><td>rewrite_pattern</td><td>条件に従って、入力リストの条件を満足させる要素を書き換え、結果を出力リストに与える。</td><td>[]</td><td>[P|C]</td><td>C</td><td>[A|C]</td><td>C</td></tr>
<tr><td>sum_pattern</td><td>入力リストの条件を満足させる要素の合計を求め、結果を出力リストに与える。</td><td>0</td><td>N1 (N1 ← N + A)</td><td>N</td><td>N</td><td>N</td></tr>
<tr><td></td><td>入力引数</td><td>[]</td><td>[A|B]</td><td>B</td><td>[A|B]</td><td>B</td></tr>
</table>

if_recursive, else_head, else_recursive部分の引数のみから定義される．あらかじめ用意している基本プログラムは，ここで定義した基本構造と引数操作パターンで表現している．ここでは理解を容易にするために，引数操作パターンである削除パターンを基本構造と組み合わせることによって表現される基本プログラム（プログラム1）を具体例として用いる．

（プログラム1）

```
predicate([ ],[ ]).
predicate([A|B],C) :-
    condition,
    predicate(B,C).
predicate([A|B],[A|C]) :-
    predicate(B,C).
```

プログラム1では，下線を付けている引数部分の組合せが削除パターンである．このように，基本構造と引数操作パターンを組み合わせたものは，入出力の区別と，conditionの部分に代入されるべき条件を与えれば，一つの完全なプログラムとなる．

基本プログラムの仕様は，先の基本構造および引数操作パターンに仕様部品として与えている．基本構造に対する仕様部品は以下のようになっている[†]．

```
basic_structure([input_list (P),condition(R)],
  [[入力リスト，P，の，条件，R，を，満足する，
    要素，すべてを],
  [ ],
  [入力リスト，P，の最後まで，条件を満たす要素，
    がなければ，プログラムは，実行を終了する],
  [入力リスト，P，が，空リスト，の場合は，プロ
    グラムは，実行を終了する]]).
```

第1引数は，変数Pの属性が入力リストであること，変数Rの属性が条件であることを示している．第2引数中で空リストが含まれているのは，基本構造のみで部分仕様が決定できないことを表し，引数操作パターンの仕様部品と組み合わせたときに部分仕様が格納される．また，引数操作パターンに与えられている仕様部品の例として，削除パターンに与えられている仕様部品を次に示す．

```
argument_manipulation (delete_pattern,
  [input_list(P),output_list(Q)],
  [[削除して，結果を，出力リスト，Q，に，格納す
    る],
  [出力リスト，Q，中に格納される，要素の，順序
    は，入力リスト，P，と，同様，である],
  [入力リスト，P，を，そのまま出力し],
  [入力リスト，P，を，そのまま出力し]]).
```

引数操作パターンの仕様部品は，第1引数にパターン名，第2引数に仕様中の各変数の属性が格納されている．従って，要素の削除を表す基本プログラムの仕様は，引数操作パターンの仕様部品を基本構造の仕様部品と組み合わせたものであり，以下のようになる．

（仕様1）

```
spec([input_list(P),output_list(Q),condition(R)],
  [[入力リスト，P，の，条件，R，を，満足する，
    要素，すべてを，削除して，結果を，出力リ
    スト，Q，に，格納する],
  [出力リスト，Q，中に格納される，要素の，順序
    は，入力リスト，P，と，同様，である],
  [入力リスト，P，の最後まで，条件を満たす要素，
```

† 実際には，仕様部品，仕様変換規則ともに，表現形式は，更に複雑である．本論文では，説明に必要な部分のみを示している．

表2　変形オペレータ

変形オペレータ	変形操作	変形前と異なる部分仕様の意味
add_argument	引数を増やす。	引数に与えられている変数を条件の中で用いる。
add_call	if_clauseで別のプログラムを呼び出す。	入力リストの要素を用いた条件チェックや変換をする別のリスト処理プログラムを用いる。
add_if	if _clauseを増やす。	条件が選言的になる。
add_if_recursive	if_recursiveを増やす。	分割したそれぞれの要素に対して処理を行う。
add_stop	停止節を増やす。	空リスト以外の条件でも停止する。
backward_recursive	if_head ,else_head部で行われている引数操作を if_recursive,else_recursive部で行う。	入力リストの要素の順序を逆にして出力リストに与える。
change_criteria	再帰呼び出しを行う際に、比較基準を変更する。	出力がリストの場合はすべての要素が条件を満たすように処理を行い、出力がアトムの場合は、最も条件を満たす要素を処理する。
change_operation_atom_to_list	処理対象の引数をアトムからリストに変更する。	アトムの処理からリストの処理に換える。
continuous_element_operation	if_head 部において入力リストの連続する要素を取り出すようにする。	入力されたリストの連続する要素に対して処理を行う。
cut_condition	conditionを取り除く。	入力リストの要素すべてに対して、同一の処理を行う。
cut_else	else _clauseを取り除く。	1つでも if 節の条件を満たさない要素があれば、実行は失敗する。
cut_else_recursive	else_recursive部を取り除く。	先頭から走査して、要素が条件を満たしている間のみ処理を行う。
cut_if_recusive	if _recursive部を取り除く。	最初に条件を満たす要素1つだけを処理する。
cut_stop	stop_clauseを取り除く。	条件を満足する要素がない時、空リストが入力された時にプログラムの実行は失敗 (fail) する。
get_initial_value	stop_clauseで出力引数と入力引数を同じ変数する。	入力される値を出力引数の初期値とする。
move_recursive_position	if_recursive部の位置を変更する。	変化なし
set_initial_value	stop_clauseで出力引数に初期値を設定する。	空リスト（あるいは0）以外が出力引数の初期値となる。

がなければ, プログラムは, 入力リスト, P, を, そのまま出力し, 実行を終了する],
[入力リスト, P, が, 空リスト, の場合は, プログラムは, 入力リスト, P, を, そのまま出力し, 実行を終了する]]).

3.2 変形オペレータと仕様変換規則

変形解析法では, まず基本プログラムを変形することによって与えられたプログラムを導く. 変形操作は, プログラムのある部分に対する削除, 追加, 変更等を定義した変形オペレータによって行われる. このような変形オペレータによって, 基本プログラムと与えられたプログラムとの対応付けがなされる. この際に適用されたオペレータ系列は, 基本プログラムと与えられたプログラムとの差を表現したものとなっており, 基本プログラムに対する仕様の修正に用いられる. 本手法で用意した変形オペレータは, 表2の17種である[†].

変形オペレータには, 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とがある. 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. 例えば, 基本構造の if_recursive 部分を削除する cut_if_recursive オペレータを前節のプログラム1に適用すると

(プログラム2)

```
predicate([ ],[ ]).
predicate([A | B],B) :-
  condition.
predicate([A | B],[A | C]) :-
  predicate(B,C).
```

のように変形される, プログラム2は, プログラム1が条件 condition を満たす要素すべてを削除するのに対し, リストの先頭から調べて条件を満たす要素を一つだけ削除するものである. 但し, cut_if_recursive オペレータは, if_head 部分の引数も同時に書き換えるようになっている. 引数の書換えは, まず if_head 部分の出力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. プログラム1では, if_head 部分の出力リスト中にある変数で if_recursive 部分の出力リストと等しいのは, 変数Cである. 従って, プログラム2では, プログラム1の if_head 部分の変数Cは, if_recursive 部分の入力リストとなっていた変数Bに書

† 現在のところ基本構造から与えられたプログラムへの変形を行うものとして命名, 定義しているが, 実際には逆に, つまり与えられたプログラムから基本構造を生成する方向に適用可能となっている. 3.3で述べる APSG/I における仕様生成では, 変形オペレータは与えられたプログラムから基本プログラムを生成する方向に用いられる.

き換えられている．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影響されずに適用できる．このため，既に削除されている構造や引数部分を削除することはできないなどの制約を除けば，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．先のプログラム 2 では，条件 condition が固定的に記述できる場合のみを扱っている．例えば，値が 60 未満の要素といったように，60 といった比較の対象となる値があらかじめ固定的に決まっている場合である．これに対し，比較の対象が入力として引数に与えられる場合がある．例えば，第 3 引数に入力された値よりも小さな値をもつ要素を先頭から走査して一つだけ削除するプログラムなどがある．この場合，上記のプログラム 2 の削除プログラムでは引数が不足しており，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．上記のプログラム 2 に add_argument オペレータを適用した例を以下に示す．

(プログラム 3)

```
predicate([ ],[ ],D).
predicate([A|B],B,D) :-
    condition (D を含んでいる).
predicate([A|B],[A|D],D) :-
    predicate(B,C,D).
```

プログラム 3 では，比較の対象が第 3 引数に割り当てられている．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．

また，各変形オペレータには仕様を修正するための仕様変換規則が付加されており，例えば，cut_if_recursive オペレータには，以下のような仕様変換規則が与えられる．

```
changing_operator (cut_if_recursive,
    [[delete_spec (すべてを), add_spec (after (要素),
      を先頭から走査して 1 個だけ)],
    [ ],[ ],[ ]]).
```

仕様変換規則の第 1 引数は規則が対応づけられている変形オペレータ名を表し，第 2 引数のリストには，プログラム仕様変更のための規則が格納されている．仕様変換規則の第 2 引数のリスト中にある項 delete_spec は引数にある単語を仕様中から削除することを表し，リスト中の別の項 add_spec は，第 1 引数が指定する場所に第 2 引数の単語を追加することを表す．従って，上記の仕様変換規則は仕様中の単語「すべてを」を削除し，更に単語「要素」の後ろに単語「を先頭から走査して 1 個だけ」を挿入することを表している．また，仕様変換規則の第 2 引数にある空リストは，プログラム仕様を表すリスト中の第 2，3，4 番目のリストには変換操作を施さないことを表している．先の仕様 1 に cut_if_recursive オペレータの仕様変換規則を適用すると，以下の仕様 2 が得られる．

(仕様 2)

```
spec([input_list(P),output_list(Q),
      condition(R)],
   [[入力リスト，P，の，条件，R，を，満足する，
     要素，　を先頭から走査して 1 個だけ，　削除して，結果を，出力リスト，Q，に，格納する],
    [出力リスト，Q，中に格納される，要素の，順序は，入力リスト，と，同様，である],
    [入力リスト，P，の最後まで，条件を満たす要素，がなければ，プログラムは，入力リスト，P，を，そのまま出力し，実行を終了する],
    [入力リスト，P，が，空リスト，の場合は，プログラムは，入力リスト，P，を，そのまま出力し，実行を終了する]]).
```

3.3 APSG/I によるプログラム仕様の生成

APSG/I は，変形解析法の実現可能性および有効性を評価するために Sun ワークステーション上の C-Prolog を用いて作成した試作システムである．次章で APSG/I を用いて行った評価実験について述べる．APSG/I に与えられる情報は，Prolog のソースプログラムと共にそこで用いられている引数の入出力の区別を表すファクトである．これは，Prolog プログラムは宣言的に記述されるため，引数の入出力が決まらないとプログラムの機能が定まらないからである．APSG/I による仕様の生成には，①与えられたプログラムから，基本プログラムと用いられている変形オペレータを取り出す段階 (Analyzing Phase)，②変形に用いた変形オペレータの組合せからプログラム仕様を生成する段階 (Generating Phase) の二つの段階がある．APSG/I では，①の段階を与えられたプログラムに変形オペレータを適用し，基本プログラムを導くといった処理方法で実現している．これは，基本プログラムから与えられたプログラムを生成するよりも，与えられたプログラムから基本プログラムの生成を試みる場合の方が，探索空間が狭くて済むからである．また，大小比較，

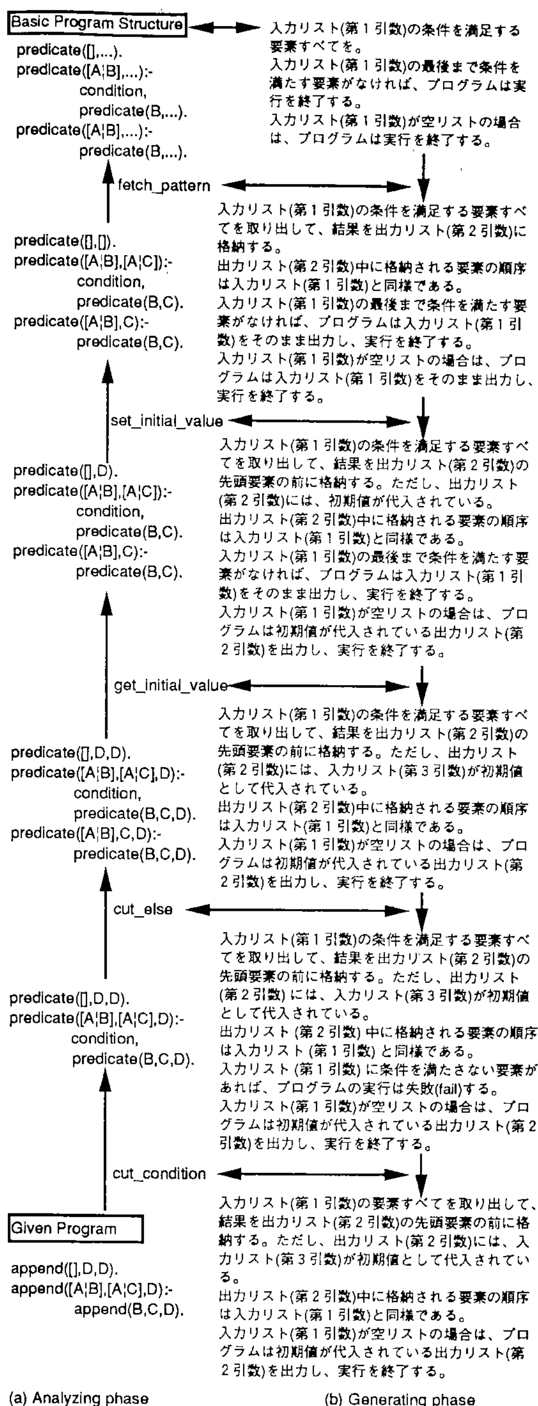


図3 APSG/I の実行列
Fig. 3 Example of execution flow of APSG/I.

入出力命令などの組込み関数については，「より大きい」，「出力する」などの言葉を対応づけて，APSG/I 内に格納している．

APSG/I を用いた変形解析の例として，二つの入力リストを結合する append プログラム（第1，3引数が入力リスト，第2引数が出力リストとなっている†）に対する処理の概要を図3(a)に示す．図3(a)のようにAPSG/I では，まず与えられた append プログラムに，変形オペレータ cut_condition，cut_else，get_initial_value，set_initial_value を逆向きに適用して，基本構造と引数操作パターン（fetch_pattern）をもった基本プログラムに変形される．また，オペレータの適用順序は，APSG/I が変形解析法の有効性を示すための実験的システムであることから，総当り的な方法を用いている．次に，②の段階では，①の段階で用いた基本プログラムの引数操作パターンと基本構造の仕様部品を組み合わせ，更に①の段階で適用した変形オペレータの仕様変換規則を適用して，プログラム仕様を生成する．図3(b)は，変形オペレータ列に対応する仕様変換規則を適用することによって，append プログラムのプログラム仕様を生成している様子を示している††．

4. APSG/I の評価実験

4.1 実験方法

APSG/I の有効性を示すために以下のような3点について評価実験を行った．

① APSG/I の適用範囲を調べる

② APSG/I の出力するプログラム仕様が細かな部分についても十分に情報を含んでいるかどうかを調べる

③ APSG/I の生成したプログラム仕様が人間にとって理解容易かどうかを調べる

①については，Prolog プログラムの代表的な文献5冊(3)～(5)，(7)，(15)で実際に紹介されているリスト処理プログラムをすべて抽出して実験対象とし，APSG/I を適用することによって，文献に記述されている仕様と同等のプログラム仕様が生成できるかどうかを調べた．同等かどうかは，筆者らを含めて Prolog

† APSG/I ではこのような入出力の区別を append (input, output, input) というファクト形式で与えている．

†† ここで生成される仕様には自然言語文として不自然なものが含まれているが，プログラム仕様としての有用性については，4.の評価実験において確認している．

表3　適用範囲に関する実験の結果

本の名前	リスト処理プログラムの個数（個）	仕様生成可能なプログラム（個）	仕様生成可能なプログラムの割合（%）
Prolog	27	23	85
Prologの技芸	34	28	82
Prologプログラミング入門	20	17	85
Programming in Prolog	29	22	76
Prologのソフトウェア作法	18	14	78
合計	128	104	81

```
predicate(input,output,input).

predicate([],A,A).
predicate([A|B],[A|C],D):-
        predicate(B,C,D).
```

No.	○×	入力	出力	入力	出力
		第一引数	第二引数	第三引数	T/F
1		[6,3,7,35,73]	[6,3,7,35,73,6,1,7,43,8]	[6,1,7,43,8]	true
2		[]	[n,o,l,d,b]	[b,d,l,o,n]	false

(a) Test example for group A

入力リスト(第1引数)の要素すべてを取り出して、結果を出力リスト(第2引数)の先頭要素の前に格納する。ただし、出力リスト(第2引数)には、入力リスト(第3引数)が初期値として代入されている。
出力リスト(第2引数)中に格納される要素の順序は入力リスト(第1引数)と同様である。
入力リスト(第1引数)が空リストの場合は、プログラムは初期値が代入されている出力リスト(第2引数)を出力し、実行を終了する。

No.	○×	入力	出力	入力	出力
		第一引数	第二引数	第三引数	T/F
1		[6,3,7,35,73]	[6,3,7,35,73,6,1,7,43,8]	[6,1,7,43,8]	true
2		[]	[n,o,l,d,b]	[b,d,l,o,n]	false

(b) Test example for group B

図4　実験に用いた入出力例
Fig. 4　Test example.

プログラムについての豊富な知識を有する数名による主観的な判断である．②については，20名のプログラマに協力してもらい，APSG/Iによって生成したプログラム仕様をもとにして，再びプログラムを作成してもらった．このプログラムと，もとのプログラムとの間で細部にいたる機能の等価性を比較することにより，仕様の欠落部分の有無について確認を行った．この実験では，各プログラマに1人当り10種類のプログラム仕様についてプログラムを作成してもらった．

③については，考慮しなければならない点として，Prologプログラマは十分に時間を費やせば，ソースプログラムを直接，理解する能力があるという事実がある．すなわち，Prologのソースプログラムは複雑ではあるが，理解可能であり，一種の仕様とみなすことができる．従って，仕様生成システムを考える場合，生成したプログラム仕様がソースプログラムと比較し，極めて理解容易であることが重要なポイントとなる．

この考え方に従い，実験③ではPrologプログラマ14名を7名ずつの二つのグループA，Bに分け，グループAには，ソースプログラム[†]のみを，グループBには，APSG/Iの生成したプログラム仕様のみを示し，あらかじめ用意した入出力例の分類を行わせた．分類はあらかじめ準備した入出力例が，示したソースプログラム，あるいはプログラム仕様の正しい入出力例になっているかどうかを求める図4のような試験形式で行わせた．準備した入出力例は1プログラム当り10例で，分類は1人当り約25個のプログラムに対して行わせた．この分類を正しく行うためには，グループAのプログラマはソースプログラム，グループBのプログラマはプログラム仕様を理解する必要がある．このため，正しく分類できるまでに要する時間には，入出力例を読み，正誤を判断して記述する分類作業時間に加えてソースプログラム，あるいはプログラム仕様の理解に要する時間が含まれている．実験では，同一プログラムに対しては，いずれのグループに属するプログラマにも同じ入出力例の分類を行わせた．従って，仕様が理解できていれば，入出力例を読み正誤を判断して記述するのみであり，分類作業時間はグループに関係なく，ほとんど同じであると考えている．以上のことから，グループによって正誤の分類に費やした時間が大きく異なるとすれば，ソースプログラムとAPSG/Iのプログラム仕様で，理解に要した時間が異なるためであると考えられる．本実験では，以上のような考え方に基づき，グループごとで分類を正しく行うまでに費やした時間を計測し，APSG/Iによるプログラム仕様の理解容易性を評価した．

4.2　実験結果と考察

APSG/Iの適用範囲に関する実験結果を表3に示す．表3より80%以上のプログラムについて，プログラム仕様を生成することができ，得られた仕様については

† 述語名からプログラムを推測できないように，“predicate”という述語名を用いている．また，図4(a)のプログラムは“append”プログラムであるが，引数の順序も一般的に用いられているものと変えてある．

すべて正しいものであった．これは，APSG/I の基本構造が1個で，引数操作パターン，変形オペレータが合計24個であることなどを考慮すれば，少数のオペレータで多数のプログラムを扱えたことになる．すなわち，教科書レベルのプログラムでは，APSG/I で定義したオペレータに十分な一般性があり，組合せが有効に行えているものと考えられる．また，APSG/I でプログラム仕様が生成不可能であった残りの約20％のプログラムは，大きく分けて，入力リストの要素に前提条件があるプログラムと，リストのもつ構造に対して操作を施しているプログラムの2種類であった．前者の代表例として，あらかじめソートされている二つのリストを結合するマージソートプログラムが挙げられる．この場合，プログラム仕様を生成するためには，入力リストがあらかじめソートされているといった前提が必要となる．これに対して，変形解析法では前提条件を扱う能力がなく，このようなプログラムの仕様生成はできない．また，後者の代表例としてはリスト中に要素として存在するリスト構造をすべて分解するプログラムが挙げられる．このようなプログラムについては，引数操作パターンとして構造を分解するパターンを追加するなどの対処の方法が考えられる．

次に，APSG/I で生成した仕様が細部について十分に情報を含んでいるかどうかに関する実験の結果を示す．実験の結果，すべてのプログラマが APSG/I の生成したプログラム仕様のみから，細部についても入出力の等価なプログラムを作成できた．また，プログラマには，プログラム作成時に必要な情報が欠けている場合，欠落部分を指摘するように指示したが，すべてのプログラマが欠落部分を指摘することなく，プログラムを正しく作成した．すなわち，APSG/I のプログラム仕様中には，入出力の等価なプログラムが作成できる程度に必要な情報がすべて含まれており，その情報がプログラマに正確に伝わっていることが示されたと考えられる．

最後に，理解容易性に関する実験③の結果を表4に示す．表4は，ソースプログラムと APSG/I のプログラム仕様のいずれかにより，プログラムの仕様を理解し，与えられた入出力例に対して正しく正誤を分類するまでの時間を示している．表4に示すように，APSG/I の仕様を用いたグループBの解答時間が，ソースプログラムを用いたグループAの解答時間に対し，約1/3と非常に短時間で行われている．これは，APSG/I の生成したプログラム仕様が理解容易であることを示唆している．また，実験の結果，グループBのプログラマは，どのプログラムに対しても約2分以内に入出力例に対する正誤の分類を完了していることが判明した．これに対し，グループAでは，プログラムによっては分類を完了するのに約15分もかかる場合がいくつかあった．このことは，ソースプログラムからの理解では非常に時間のかかった複雑なプログラムに対しても，プログラム仕様を用いればほとんど一定の短い時間で理解可能であることを示している．従って，理解困難なプログラムの方が，ソースプログラムと APSG/I のプログラム仕様との間での理解容易性の差は，大きくなるものと思われる．実際に，理解するのに時間のかかっているプログラムに対する実験結果のみを集計したところ，グループBのプログラマは，グループAのプログラマに比べ，1/4以下の時間で正しく分類を行えていた．その理由として，Prolog のプログラムではそのプログラムで何を行うのかといった機能が，プログラムの構造や引数の組合せといった形で複数箇所に分けて表現されている．このため，プログラムそのものを理解するには，これらの分散された情報を見つけ出し，まとめあげて機能を考える必要がある．これに対して，APSG/I では基本構造，引数操作パターンという形でプログラムの大まかな機能を取り出され，その機能が仕様に反映されている．このため機能の理解についてこのような大きな時間の差になったと思われる．

表4 理解容易性に関する実験の結果

	被検者数	1問当りにかかった平均時間	時間の比
ソースプログラム（グループA）	7名	257秒	3.06
プログラム仕様（グループB）	7名	84秒	1

また，APSG/I によって生成される仕様には不自然な記述も含まれてはいるが，これらの実験の結果から教科書レベルのプログラムに関しては，適用可能である．

5. むすび

本論文では，プログラム仕様自動生成の手法として変形解析法を提案し，変形解析法の有効性を示すために行った実験について述べた．本研究は，再利用プログラムベース構築を目指し，ソースプログラムからプログラム仕様を自動生成するといった目的をもっている．リバースエンジニアリングやリエンジニアリングといった分野では，保守作業の向上を目指して，同様

の目的をもった研究が行われている[14],[16]．文献(16)では，アセンブラプログラムを対象に，よく用いられている命令パターンを抽出し，それらと直接マッチングを行うことによって，パターンに対応した仕様記述（英文）に置換し理解容易な仕様記述の系列を得る方法が述べられている．文献(14)では，Cobolプログラムを対象に，プログラムの制御の構造化を行い，条件を伴った代入式にプログラムを変換し，組み合わせる（関数の合成）ことによって，ソースプログラムから理解容易で厳密な表現形式（数学的な記述）をボトムアップに生成する方法が述べられている．両者ともプログラムを解析するために，あらかじめ抽象度の低いひな型を用意し，それとの単純なマッチングを行っている．これらの方法では，置換する仕様に対する保証を与えやすく，システムの作成も容易となる．しかしながら，類似はしているが細部の構造の異なったプログラムが多く存在するようなPrologプログラムを対象とした場合，類似したひな型を非常に多く用意しなければならなくなる．一方，変形解析法ではソースプログラムをひな型で表現するのみでなく，変形するといった新しい視点を導入することによって，抽象度の高いひな型とマッチングを実現している．これによって，少ないひな型とオペレータで，このようなプログラムの仕様生成を行っている．

また，本手法のような変形を繰り返すことによって解を得るようなシステムでは変形の停止性と合流性が問題となるが，ここでは実験的な行った検証について述べる．変形の停止性については，適用範囲に関する実験で取り扱ったプログラムに対して変形オペレータを逆向きに適用した結果，基本構造をもったプログラム，または適用可能なオペレータが存在しないようなプログラムに変形され，解析処理が停止することを確認した．また，変形の合流性については，理解容易性に関する実験で用いた25個のプログラムについて別解の生成を試みたところ，オペレータに適用順序が異なる場合においても，同様の仕様を生成することができた．しかしながら，停止性と合流性に対しての厳密な保証を与えることはできておらず今後の検討が必要である．

また，変形解析法では，Prologのリスト処理のみを対象としているが，リスト処理に限らず多くのプログラムを対象とすることが必要である．実際に，Prologプログラムにおいて，リスト処理以外のプログラムでも類似した構造やパターンがあり，いくつかの基本構造が存在する[9]．従って，基本構造，引数操作パターン，変形オペレータをリスト処理以外のプログラムにも適用可能にするための検討が必要である．また，マージソートプログラムのように入力データに前提条件があるプログラムにも対処できるように，変形解析法の枠組みを拡張することも検討する予定である．

謝辞　本研究における評価実験に協力して頂いた大阪大学産業科学研究所音響機器部門，電子機器部門，音響材料部門の諸氏，ならびに関西大学工学部管理工学科4年生の諸氏に感謝致します．本研究の一部は，文部省科学研究費（重点領域研(2)02249204）による．

文　献

(1) Carbonell J. G.: "Learning by Analogy: Formulating and Generalizing Plans from Past Experience", in Machine Learning: An Artificial Intelligence Approach, R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (Eds.), Tioga, Palo Alto, Calif. (1983).

(2) 原田　実，篠原靖志："部品合成によるプログラム自動生成システムARIES/I", 情処学論, **27**, 4, pp. 417-424 (1986).

(3) 木原茂之："Prologプログラミング入門", オーム社 (1988).

(4) 黒川利明："Prologのソフトウェア作法", 岩波書店 (1985).

(5) Sterling L. and Shapiro E.: "Prologの技芸", 松田利夫訳, 共立出版 (1988).

(6) 永澤勇一，今中　武，萩野貴之，永井隆弘，江嶋義典，三根　久，豊田順一："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—プログラムの変形を利用した支援手法", 情処学ソフトウェア工学研報, 91-SE-77, 77-13 (1991).

(7) 中島秀之："Prolog",産業図書 (1983).

(8) Imanaka T. Uehara K. and Toyoda J.: "Analogical Program Synthesis from Program Components", in Lecture Note in Computer Science: Logic Programming '87. K. Furukawa, H. Tanaka and T. Fujisaki (Eds.), Springer-Verlag (1988).

(9) 今中　武，上原邦昭，豊田順一："類推，帰納の概念を導入したプログラミング知識の学習メカニズム", 情処学知識工学と人工知能研報, 88-AI-59, 59-13, pp. 113-122 (1988).

(10) 今中　武，上原邦昭，豊田順一："プログラムの類似性定義のためのネットワーク表現", 情処学第38回全大, 7L-3, pp. 1201-1202 (1989).

(11) 古宮誠一："部品合成によるプログラム自動合成システムPAPS—知識工学的アプローチを用いたその実現方式について", 信学技報, SS87-2 (1987).

(12) 古宮誠一："自動プログラミングシステムの構築に必要な部品とその完備性の保証について", 情処学ソフトウェア工学研報, 89-SE-68, 68-2 (1989).

(13) 元木　誠，永田守男："類似プログラムの抽出の一技法", 情処学第36回全大, 6P-1, pp. 1477-1478 (1988).

(14) Hausler P. A., Pleszkoch M. G., Linger R. C. and

Hevner A. R. : "Using Function Abstraction to Understanding Program Behavior", IEEE Software, **7**, 1, pp. 55-63 (1990).

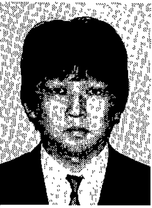
(15) Clocksin W. F., Mellish C. S. : "Programming in Prolog", 中村克彦訳, MICROSOFTWARE. INC. (1983).

(16) Kozaczynski W., Lingosari E. S. and Ning J. Q. : "BAL/SRW : Assembler reengineering workbench", Information and Software Technology, **33**, 9, Butterworth-Heinemann Ltd., pp. 675-684 (1991).

(平成4年11月16日受付, 5年7月20日再受付)

永井 隆弘

平3阪大・基礎工・情報卒. 平5同大大学院前期課程了. 同年松下電器産業(株)入社. 人工知能, 特に知的プログラミング環境に関する研究に従事. 情報処理学会, 人工知能学会各会員.

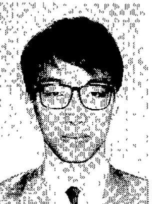


今中 武

昭60阪大・基礎工・情報卒. 平3同大大学院博士課程了. 工博. 同年松下電器産業(株)入社. 自動プログラム合成, ニューラルネットワークの研究に従事. 情報処理学会, 人工知能学会各会員.

永澤 勇一

平1関西大・工・管理工卒. 平3同大大学院修士課程了. 同年三菱電機(株)入社. プロトタイピング技術, オブジェクト指向技術に関する研究に従事. 情報処理学会会員.



平嶋 宗

昭61阪大・工・応物卒. 平3同大大学院博士課程了. 同年阪大産業科学研究所助手. 工博. 現在, 人工知能, 特にITSおよびプログラミング環境の研究に従事. 情報処理学会, 人工知能学会, 教育工学会各会員.

上原 邦昭

昭53阪大・基礎工・情報卒. 昭58同大大学院博士課程単位取得退学. 阪大産業科学研究所助手, 講師を経て, 平2神大工学部情報知能工学科助教授. 工博. 人工知能, 特に機械学習, 自然言語理解, プログラム合成の研究に従事. 1990年度人工知能学会研究奨励賞受賞. 人工知能学会, 情報処理学会, 計量国語学会, 日本ソフトウェア科学会, システム制御情報学会各会員.

豊田 順一

昭36阪大・工・通信卒. 昭41同大大学院博士課程単位取得退学. 同年同大基礎工助手. 昭44助教授. 昭57阪大産業科学研究所教授. 工博. 現在, 概念の形成, Visual-fidelity, マニュアルのわかりにくさに関する研究に従事. 情報処理学会, 人工知能学会, 日本認知科学会各会員.