ソフトウェア工学 94－11
(1993. 9. 21)

個人の言語によるプログラム作成システム

小林　要，木村高久

株式会社富士通研究所情報社会科学研究所

410-03　沼津市宮本１４０番地

あらまし

計算機プログラムを、応用目的の事柄、用いられる計算機動作、およびこの両者の対応づけ、の要素に分けて記述するプログラミング手法である役割プログラミング法を提案する。役割プログラミングのために我々が開発したツールＲＰＳを紹介し、役割プログラミングによるプログラム作成手順について示す。本方法によって、応用目的の事柄を計算機言語とは独立に個人の言葉で表現できる。また、計算機動作を任意の計算機言語で表現できる。さらに、両者の対応づけの方法を指示する変換規則言語を導入して、対応づけを自動的に行わせることができる。本手法の本質的な効果は、保守改良時、およびプログラムの他への流用時であることを主張する。

和文キーワード　役割木、役割木変換、表現生成、利用者言語、プログラムソースコード

A System For Programming-In-Personal-Languages

Kaname Kobayashi, Takahisa Kimura

Institute For Social Information Science,
Fujitsu Laboratories LTD.

140 Miyamoto, Numazu-shi, Shizuoka, 410-03 Japan

Abstract

We propose a 'role-programming method', where programs are described as the triplet of (a) application objects, (b) computational functions used by the programs, and (c) their associations of (a) and (b). We present role-programming processes by using RPS system which is therole programming tool we have been developing. In our method, application objects are represented by using personal languages inherent to the application domain separetedly from programming languages. Any programming languages planed to use can be used. In addition, the association can be done automatically by introducing association description languages. We consider that this method works well in maintenance of programs and program reuse.

英文 key words Role tree, Role tree transformation, Representation generation, User-language, Source program

1．はじめに

計算機プログラムには、(a)応用目的の事柄、(b)用いられる計算機動作、(c) および(a)と(b)の間の対応づけ、の3つが同時に含まれていると考えられる。本研究では、プログラムをこれらの要素に分けて記述する方法を提案する。本方法によって、応用目的の事柄を計算機言語とは独立に自分の言葉で表現できる。また、使う計算機動作を任意の計算機言語で表現できる。さらに、両者の対応づけの方法を指示する変換規則言語を導入して対応づけを自動的に行わせることができる。

本方法では、対象の果たす役割を明示的に区別する必要がある。筆者らは、対象の果たす役割を明示的に区別するプログラミング方法のことを“役割プログラミング(role programming)”と呼んでいる。役割を区別するとは、例えば、引き算サブルーチンSUB(A,B,C)が使われているとする場合、どれが引く数で、どれが引かれる数か、どれが引き算の答えか、などの区別を行うことに相当する。

本手法は応用目的の事柄を自分の言葉で表現するという点で、literacy programming[1]に類似しているようにも見えるが、我々の狙いはプログラム作成に用いた情報を相互に独立なものに分解し、できるだけ再利用性の高いデータにすると同時に、自動化できる部分を次々と自動化することにある。

本研究では役割を識別した役割木と呼ぶデータ構造を用い、これを変換することによって応用側と計算機側とを結びつける。役割木は構文木や属性文法[2]で言う属性木に類似の構造のものであるが、属性が値への写像である点や構文木が文法的構造を表している点に対して、役割は対象への写像であり、対象の利用関係を表す点で異なる。また役割木の変換と従来のプログラム変換[3]との違いは、等価変換でない点や、役割の変換と対象の変換を区別する点などにある。

現在はパソコンのMS-DOS上で動作する基本ツールRPS 第一版を試作し、日本語で記述した初期文書からMASMプログラムを生成する実験を行っている。ツールは特定言語に依存しないように作られているため、今後はC言語など他の言語用の実験を行いデータを蓄積する予定である。本稿では、役割プログラミングの処理系を紹介しながら、役割プログラミングによるプログラムの作成手順について報告するが、本手法の本質的な効果は保守改良時および他への流用時に発揮されることを主張したい。

2．役割プログラミングの処理系

役割プログラミングの処理系は、図1に示す3段階からなる。makeTree、decVoc2 、trans 、showV の4本のバッチ・ツールと、初期文書、語彙変換規則集、役割木変換規則集、パターン集、語彙集の5種類のデータを使用する。

第1段階では、初期文書と語彙変換規則とを入力して開始役割木を生成する。初期文書は、個人言語により書かれた抽象プログラムである。第2段階では、役割木変換規則集を入力して開始役割木の内容を詳細化し、最終役割木を生成する。第3段階では、パターン集と語彙集とを入力して、最終役割木からソースプログラムを得る。

役割プログラミングにおけるプログラマの作業はこれら5種類のデータを必要に応じて新規に作成したり、既存データを再利用することに相当し、処理系の処理順序とプログラマのプログラミング作業の順序とは一致しない。

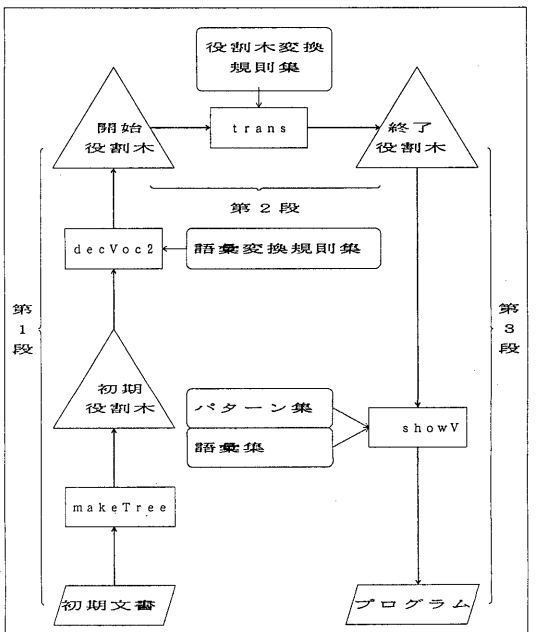

図１．役割プログラミング処理系

3．役割木の構成

3．1．役割木の形式と解釈

役割木は役割プログラミングを特長づけるデータ構造である。役割木は、視点定義と呼ばれる記述の集まりである。一つの視点定義は、表現したい事柄、表現したい事柄を見る見方、役割、役割を演じる対象の4種類の識別名間の関係を表す〔図2a〕。ここで、表現したい事柄を主視点と呼ぶ。表現したい事柄を見る見方を役割パターンあるいは単にパターンと呼ぶ。役割を演じるものを副視点と呼ぶ。

視点定義は主視点と視点内容の組で表され、一つの主視点には一つのパターンが組み合わされる。視点内容の形式は、図2bから図2dまでの3種類の形式が与えられている。図2bの形式〔固定長形式〕で、一つの視点内容における役割：副視点の組みの並びの全体を役割付与と呼ぶ。図2cの形式〔いれこ形式〕は、副視点を主視点に持つ別の視点定義の視点内容を、副視点の位置に代入したものである。処理系の内部表現では、代入によって消去された副視点を仮の副視点で補って、図2bの形式に還元される。

図2dの形式〔不定長形式〕は、役割付与部分が副視点の並びで、先頭からの相対順位によって順位役割を暗黙に表すために使用する。内部表現では、先頭からの相対位置を表す順位役割を内部的に補って、図2bの形式に還元される。いれこ形式も許される。

役割木は、VPRS表と呼ばれる数学的関係〔表1参照〕として解釈される。

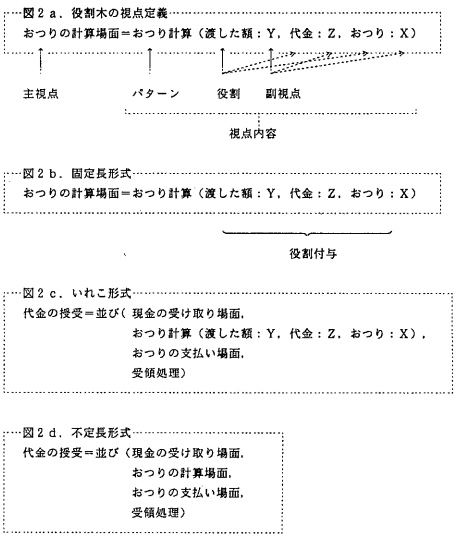

図2．視点定義とその形式

表1．役割木とそのVPRS表

役割木

例1＝条件実施（条件：渡した額が代金より多い，内容：おつりの計算場面）
渡した額が代金より多い＝大小比較（大：Y，小：Z）
おつりの計算場面＝おつり計算（渡した額：Y，代金：Z，おつり：X）

vprs表

視点（v）	パターン（p）	役割（r）	副視点（s）
例1	条件実施	条件	渡した額が代金より多い
例1	条件実施	内容	おつりの計算場面
渡した額が代金より多い	大小比較	大	Y
渡した額が代金より多い	大小比較	小	Z
おつりの計算場面	おつり計算	渡した額	Y
おつりの計算場面	おつり計算	代金	Z
おつりの計算場面	おつり計算	おつり	X

3．2．役割木の制約条件

主視点および副視点を総称して視点と呼ぶ。役割木の中で主視点としてのみ用いられる視点をトップ視点という。トップ視点は複数あってもよい。主視点としても副視点としても用いられる視点を中間視点という。副視点としてのみ用いられる視点を語彙視点という。

まず、役割パスの概念を定義する。

1）視点定義における主視点をxとするとき、その視点内容において副視点yが存在すれば、xからyへの役割パスがあると言う。

2）xからyへの役割パスがあり、yからzへ

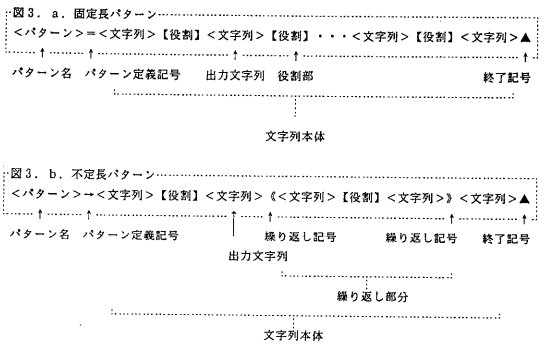

図3．役割パターン文字列とその形式

の役割パスがあるときに、xからzへの役割パスがあると言う。

3) 上記1)、2)のみによって、役割パスがあることが定まるものとする。

視点定義の集まりが役割木として満たすべき4つの条件を以下に示す。

① 中間視点を副視点として持つ視点定義は唯一でなければならない。（中間視点の唯一性）

② トップ視点を主視点とする視点定義は唯一でなければならない。（トップ視点の唯一性）

③ 一つの視点定義で一つの役割には一つの副視点のみを付与する。（vprs表における視点・パターン・役割から副視点への関数従属性）

④ 任意の中間視点には、自分自身への役割パスがない。（中間視点の非ループ性）

4．役割木から文書表現の生成

パターン集と語彙集を用いて役割木から文書表現を生成する。パターン集は、パターン文字列の集まりである。語彙集は、語彙文字列の集まりである。語彙文字列は、語彙視点と、それに対応する出力文字列との組で与えられる。省略すると副視点が出力文字列とされる。詳細は割愛する。パターン文字列は、パターン名、パターン定義記号、文字列本体からなり、固定長パターン（図3ａ）と不定長パターン（図3ｂ）とがある。

文書表現を生成する手続きは、役割木上のパターンと語彙視点とに文字列を割り当て、役割木の木構造に従って出力文字列を組み立てる。図4に役割木と出力文字列との関係を示す。

固定長パターンは、パターン名、固定長パターン定義記号、固定長文字列本体から構成される。固定長文字列本体は、出力文字列と役割部との任意の並びで構成される。表現生成は、視点定義におけるパターンに対応した固定長文字列本体をとり出し、本体の出力文字列を出力する。本体の中に役割部があらわれると、視点定義で同じ役割が付与された副視点を同様の方法で出力文字列におきかえて生成する。役割部は、役割木上で漏れなく含まれている必要がある。

不定長パターンは、パターン名、不定長定義記号、不定長文字列本体の組で構成する。不定長文字列本体には出力文字列と役割部とが任意

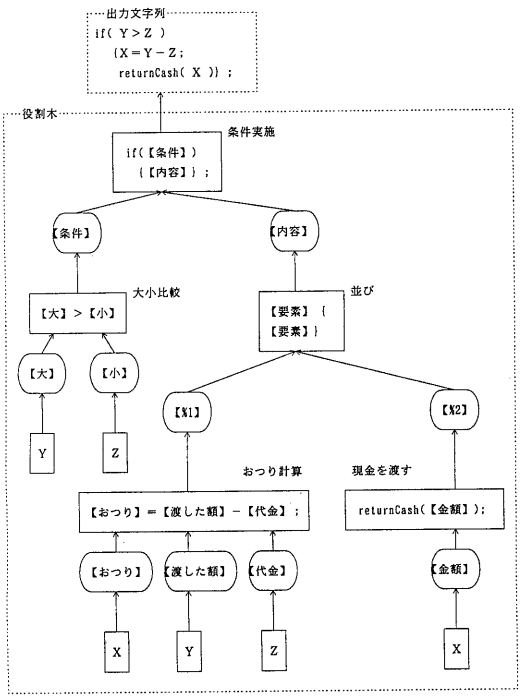

図４．表現生成手順

図5ａ．初期役割木ファイル

```
■=(:,)#■
トップ視点= 部分並び( 部分( 名前: 代金の授受,
          内容: 並び( コメント( 内容: メインプログラム),
                代金に対する現金を受け取る,
                条件実施( 条件: 失敗,
                      内容: 代金の授受に戻る),
                条件実施( 条件: 現金が代金より少ない,
                      内容: 並び( 代金不足請求を行う,
                            代金の授受に戻る)),
                条件実施( 条件: 現金が代金より多い,
                      内容: 並び( おつりの計算場面,
                            おつりの支払い場面)),
                受領処理)),
    部分( 名前: 受領処理,
          内容: 並び( 品物を包んで渡す)),
    部分( 名前: おつりの計算場面,
          内容: 並び( 受け取った額に対する代金のおつりを計算する)),
    部分( 名前: おつりの支払い場面,
          内容: 並び( おつりを渡す)))
```

図5ｂ．初期文書ファイル

```
■=(:,)#【】※。→ {} ■
【代金の授受】
  ※メインプログラム。
  代金に対する現金を受け取る。
  失敗→代金の授受に戻る。
  現金が代金より少ない→ {代金不足請求を行う。代金の授受に戻る。}
  現金が代金より多い→ {おつりの計算場面。おつりの支払い場面。}
  受領処理。
【受領処理】品物を包んで渡す。
【おつりの計算場面】
  受け取った額に対する代金のおつりを計算する。
【おつりの支払い場面】
  おつりを渡す。
```

図５．初期文書と初期役割木

に並べられると同時に、繰り返し記号対によって、繰り返し出力される部分が囲まれる。表現生成は、該当の不定長パターンを持つ視点定義に対して、不定長文字列本体の出力文字列を出力する。副視点があればそれを評価して得られる文字列を、役割部の登場順序に従って出力する。視点内容の副視点列がなくなるまで、繰り返し記号に囲まれた部分に沿って繰り返す。

パターン集や語彙集はファイルとして与えることができるので、出力文字列だけを変えたものを用意すれば、ファイルを交換することによって一つの役割木から複数言語に対するプログラムを生成できる。

5．個人言語から初期役割木の生成

初期役割木は部分の並びから構成される〔図5a〕。各部分は、部分の名前と文の並びから構成される。個人言語で書かれた初期文書は、初期役割木の構成に写像されるものであれば何でもよく、RPS第1版では、初期役割木にほぼ対応する構造とした〔図5b〕。

初期文書は、簡単な区切り記号を利用して、初期役割木に変換される。初期文書の構造は、処理系makeTreeを個人用に作成することでいくらでも拡張できる。これらの構造は、初期文書の先頭に宣言される区切り文字を手掛かりに表される。

個人言語の表現は、文を単位に、初期役割木の語彙視点に対応づけられる。初期役割木の語彙視点のうち、役割付与構造に展開できるものや、個人的表現から標準的表現に書き換えることが望ましいもの等を書き換えて、開始役割木を得る。語彙変換規則集は、語彙変換規則の並びであり、各々は前提部と帰結部から成る。

語彙変換規則の前提部と語彙視点との文字列照合に成功すれば、前提部の照合変数位置に文字列が得られる。これを対応する帰結部の、同じ照合変数をもつ位置に代入して、書き換え文字列を得る。得られた書き換え文字列で、語彙視点を書き換える。各々の語彙視点に対して語彙変換規則の並び順に照合が行われ、照合しなければ、書き換えられない。語彙変換規則の例を図6に示す。

```
…共用性の高い語彙変換規則…
■【】→▲■
【先】に戻る→進む( 先: 【先】) ▲
【対象】が【比較対象】より少ない→大小比較( 小: 【対象】, 大: 【比較対象】) ▲
【対象】が【比較対象】より多い→大小比較( 大: 【対象】, 小: 【比較対象】) ▲

…問題固有の語彙変換規則…
■【】→▲■
受け取った額に対する代金のおつりを計算する
    →おつり計算( 受け取った額: 現金, 代金: 代金, おつり: おつり) ▲
代金に対する現金を受け取る→現金受け取り( 代金: 代金, 現金: 現金) ▲
代金不足請求を行う→代金不足請求( 代金: 代金, 現金: 現金) ▲
品物を包んで渡す→品物渡し( 品物: 品物) ▲
おつりを渡す→おつり渡し( おつり: おつり) ▲
```

図6．語彙変換規則

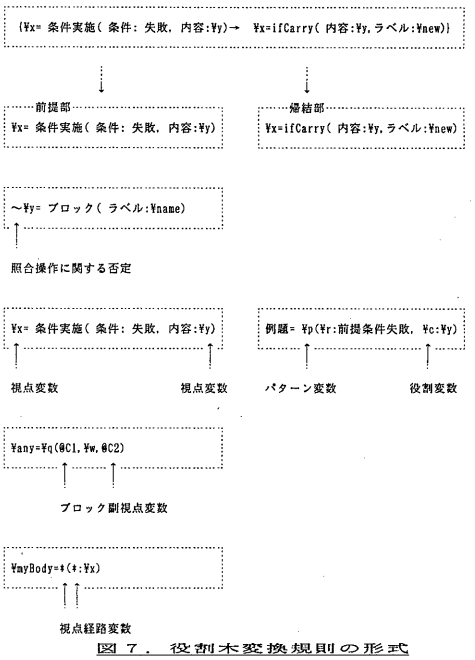

図7．役割木変換規則の形式

表2．役割木変換手順

役割木変換規則

{¥x= 条件実施(条件: ¥y, 内容:¥z)→¥x=ifThen(条件:¥y,内容:¥z,ラベル:¥new)}

役割木変換規則の内部表現

前帰	肯否	視点名	パターン名	役割名	副視点名	照合状態
前提部	肯定	¥x	条件実施	条件	¥y	照合状態
前提部	肯定	¥x	条件実施	内容	¥z	照合状態
帰結部	肯定	¥x	ifThen	条件	¥y	照合状態
帰結部	肯定	¥x	ifThen	内容	¥z	照合状態
帰結部	肯定	¥x	ifThen	ラベル	¥new	照合状態

〔続く〕

6．役割木から役割木への変換

6．1．視点条件の形式

役割木から役割木への変換には、役割木変換規則集を適用する。役割木変換規則集は、役割木変換規則を並べたものである。

一つの役割木変換規則は、図7に示すように、前提部と帰結部とからなる。前提部、帰結部はそれぞれ、複数個の視点定義形式の視点条件から構成される。各視点条件には存在否定記号～を前置することができる。

視点条件には、主視点、パターン、役割、副視点の各位置に変数を置くことができる。変数は3種類あり、それぞれ区切り記号によって区別される。￥記号で始まる変数は、主視点、パターン、役割、副視点の各位置に置くことができ、おのおの一つの名前を値にとる。@記号で始まる変数は、主視点、副視点の位置のみに置くことができ、副視点列を値にとる。記号＊は特別な変数を表す。v=*(*:s)は、役割木上で、視点vから視点sへの役割パスが存在する条件を表す。@b=*(*:s)は、副視点列@bの中に、副視点sが含まれる条件を表す。変換規則の形式を図7に示す。

6．2．視点条件の照合

存在否定記号～を持たない視点条件について、役割木との照合を以下で定める。視点条件に含まれる変数に適当な値を割り当てて、役割木の視点定義の一つが得られるならば、その視点条件と役割木との照合は成功したと言う。どのように値を割り当てても、役割木の視点定義のどれとも一致しなければ、照合は失敗したと言う。

存在否定記号～を持つ視点条件と役割木との照合を以下で定める。存在否定記号～を無視した視点条件と役割木との照合に成功すれば、～を付加した視点条件と役割木との照合は失敗する。～を無視した視点条件と役割木との照合に失敗すれば、～を付加した視点条件と役割木との照合は成功する。即ち、～は、照合の成功と失敗とを逆転させる。～付きの視点条件と役割木との照合に成功した場合には、視点定義の持つ変数には値が割り当てられない。

表２．役割木変換手順〔続き〕

役割木
例１＝条件実施（条件：渡した額が代金より多い，内容：おつりの計算場面）
渡した額が代金より多い＝大小比較（大：Y，小：Z）
おつりの計算場面＝おつり計算（渡した額：Y，代金：Z，おつり：X）

前提部照合結果

前帰	肯否	視点名	パターン名	役割名	副視点名	照合状態
前提部	肯定	例１	条件実施	条件	渡した額が代金より多い	照合状態
前提部	肯定	例１	条件実施	内容	おつりの計算場面	照合状態

構成された帰結部
例１＝ifThen（条件：渡した額が代金より多い，
内容：おつりの計算場面，
ラベル：＃１）

構成された帰結部の内部表現

前帰	肯否	視点名	パターン名	役割名	副視点名	照合状態
帰結	肯定	例１	ifThen	条件	渡した額が代金より多い	照合状態
帰結	肯定	例１	ifThen	内容	おつりの計算場面	照合状態
帰結	肯定	例１	ifThen	ラベル	＃１	照合状態

強制の発生要因と対応

視点が一致しかつパターンが異なるという不整合
旧パターン「条件実施」は削除 新パターン「ｉｆＴｈｅｎ」は追加

役割木への強制追加結果
例１＝ifThen（条件：渡した額が代金より多い，内容：おつりの計算場面，ラベル：＃１）
渡した額が代金より多い＝大小比較（大：Y，小：Z）
おつりの計算場面＝おつり計算（渡した額：Y，代金：Z，おつり：X）

バッチプログラムの定義

```
rem (1) 初期文書user.docを初期役割木user.trlに変換します
maketree user.doc > user.trl

rem (2) 初期役割木を標準語彙変換します〔変換規則集beginm.vr 、変換結果user.tr2〕
decvoc2  beginm.vr user.trl user.tr2

rem (3) 初期役割木をユーザ語彙変換します〔変換規則集user.vr 、変換結果user.tr3〕
decvoc2  user.vr user.tr2 user.tr3

rem (4) 標準プログラム構造への役割木変換を行います〔変換規則集beginm.rul〕
trans user.cnl

rem (5) ユーザ部品の役割木変換を行います〔変換規則集user.rul〕
trans user.cn2

rem (6) 最終の役割木変換を行います〔変換規則集end.rul 〕
trans user.cn3

rem (7) 表現生成を行います〔MASM用パターン集を参照する〕
showv user.shw
```

図８．プログラム生成用バッチ例

6．3．役割木変換規則の適用

役割木への変換規則の適用は、前提部の照合と帰結部の強制追加によって構成される。

前提部の照合では、前提部を構成する視点条件を順次取り出し、役割木と照合する。照合に成功すれば、照合した状態を維持して次の視点条件が取り出される。照合に失敗すれば、一つ前の視点条件に戻って、旧照合状態の次の候補から照合をとり直す。前提部を構成する視点条件の全てに対して照合が成功すれば、前提部の照合が成功する。照合をとり直していく過程で、最初の視点条件の照合が失敗すれば、前提部の照合が失敗する。前提部と役割木との照合に失敗すれば、当該規則適用は終了として、次の規則の適用に進む。成功すれば、帰結部の強制追加に進む。

帰結部の強制追加ではまず、前提部の照合で変数に割り当てられた値を、帰結部の該当変数に代入して新たな視点定義を構成する。前提部で値が未定の￥で始まる変数については、新しい値を生成して割り当てる。視点条件に存在否定記号～が付加されていない場合には、帰結部で構成した視点定義を役割木に強制的に追加する。記号～が付加されている場合には、帰結部で構成した視点定義に該当する視点定義を役割木から削除する。強制的に追加するとは、追加前の役割木の視点定義と帰結部で構成した視点定義とが矛盾する場合に、追加前の役割木側の矛盾する視点定義を削除して、新しい方を追加し、矛盾しない場合にはそのまま追加する操作のことをいう。一つの変換規則は、適用回数が適用上限に達するか、適用に失敗するまで、繰り返し役割木に適用される。

制御ファイルの内容

```
user.cn1
■=▲+ ■
input =user.tr3▲      ← 入力の開始役割木ファイル (user.tr3)
output=user.tr4▲      ← 出力の最終役割木ファイル (user.tr4)
rule=  begin.rul      ← 適用する変換規則ファイル

user.cn2
■=▲+ ■
input =user.tr4▲      ← 入力の開始役割木ファイル (user.tr4)
output=user.tr5▲      ← 出力の最終役割木ファイル (user.tr5)
rule=  user.rul       ← 適用する変換規則ファイル

user.cn3
■=▲+ ■
input =user.tr5▲      ← 入力の開始役割木ファイル (user.tr5)
output=user.tr6▲      ← 出力の最終役割木ファイル (user.tr6)
rule=  end.rul        ← 適用する変換規則ファイル

user.shw
■=▲+ ■
RoleTree=user.tr6▲    …… 最終役割木ファイル
pattern= begin.pat +  …… begin.rul やend.rul に関連した共通パターン集
    user.pat▲         …… user.rulや初期役割木に関連した独自パターン集
voc = end.voc +       …… begin.rul やend.rul に関連した共通語彙集
      user.voc ▲      …… 最終役割木に関する独自の語彙集
top= top  ▲           …… トップ視点ファイル
output= user.asm ▲    …… 出力のソースプログラムファイル
```

図８．プログラム生成用バッチ例〔続き〕

7．役割プログラミングの作業手順

役割プログラミングには、初期文書、語彙変換規則、役割木変換規則、パターン集、語彙集の５種類のデータを使用する。これらのデータを用意して、例えば図８に示すバッチ・プログラムによってツールを起動し、プログラムを生成する。役割プログラミングの作業はこれら５種類のデータを準備する作業に相当する。これらのデータを準備する作業の順序と役割プログラミングの処理系の処理順序とは独立である。

7．1．初期文書と語彙変換規則の準備方法

初期文書では、目的に応じた言葉を用いて応用上の処理内容を適切に表現すると同時に、オーバーディシジョンにならないように抽象化して記述する。語彙変換規則によって、初期文書の文を詳細化して、視点定義の定義内容になるように変換する。この場合に用いるパターンには、未決定の役割付与部分が残されていてよい。語彙変換規則集の再利用では、初期文書の記述に使用する標準的表現を予め決めると効果的である。例えば、ＡがＢより多い、Ｘに進む、等の表現が考えられる（図６）。

これらの標準的表現を実体間の関係として書き換えるための語彙変換規則は、初期文書を役割木に変換するために再利用できる。

7．2．役割木変換規則集データの準備方法

初期文書から得られる開始役割木には、必ずしもプログラムの構造が特定されていないので、役割木変換によってプログラムの構造に置き換える。

変換規則によって、未決定の役割付与部分を補ったり、あるいは、応用目的のパターンを計算機動作のパターンにおきかえる。計算機動作への対応づけの変換も、いくつかの段階に分けることによって、規則の再利用性を高めることができる。例えば、部品を利用する場合にも、部品利用における抽象化された標準的パターンに置きかえるまでを行う段階と、そのパターンをさらに詳細化する段階に分けると、後者の再利用性が高められる。

役割木変換規則は、プログラミング作業の自動化にも用いられる。プログラミング作業では、例えば、変数を用いると変数宣言が必要になるなど、依存関係が生じる。こうした依存関係に注目して、作業内容を役割木変換規則に変えて再利用する。役割木変換は大きく次の4段階に分けると再利用性が高まる：①プログラムの構造への変換、②そのプログラム独自の変換、③部品の再利用に関する変換、④プログラム言語に依存した変換。

7．3．パターン集と語彙集の準備方法

パターン集、語彙集は、計算機プログラム言語やインプリメントの版数などによって、分けておくことができる。特にパターン集については、プログラム言語を特定すると、きわめて再利用性の高いデータが作成できる。プログラムに用いられる各種の定数パラメタなどを語彙集で定義することによって、パラメタ変更などに容易に対処することができる。

8．おわりに

役割プログラミングの考え方を導入することによって、①応用目的とする内容、②用いたインプリメント技術、③プログラム言語表現など、プログラム作成に用いた様々の情報をそれぞれデータとして分離することができた。また、それらのデータを入力してプログラムを生成するツールを作成した。

現在、利用者表現からMASMソースプログラムを生成するためのRPS データを使って、我々は、RPS自身を生成する実験を行っている。四本のツール、makeTree、decVoc2 、trans 、showV のうち、前二者については生成を終了した。

基礎部品を固定し、部品展開をルール化したことにより、煩雑なソースコードの手直しが不要になった。既存のデータの再利用性を高めることによって、新規作成データの量を削減することができた。プログラム作成初期段階で決定が困難な項目も、役割木変換規則やパターン集・語彙集で指定するようにして、決定を後まわしにすることができた。プログラミング作業間の依存関係が予想以上に存在していることが判り、プログラミング作業の自動化の効果が高いことが判った。

役割プログラミングツールRPS によって、プログラムの保守作業や改良、流用の作業が改善されるものと考える。

文献

1) Knuth, D. E.: “Literate programming”, The Comput. Journ., Vol. 27, No.2, 1984, Wiley.
2) Partsch, H. et. al.: “Program Transformation Systems”, ACM Computing Surveys, Vol. 15, No. 3, pp. 199-236.
3) Knuth. D. E. :“Semantics of Context- Free Languages”, Mathematical System Theory, Vol. 2, No.2(1968), pp. 127-145.

盛光印刷所