論　文

プログラムの振舞いに基づく再帰プログラミングの教育支援

松田　憲幸[†]　柏原　昭博[†]　平嶋　宗[†]　豊田　順一[†]

A Tutoring for Behavior-Based Recursive Programming

Noriyuki MATSUDA[†], Akihiro KASHIHARA[†], Tsukasa HIRASHIMA[†], and Jun'ichi TOYODA[†]

あらまし　プログラミングを行うためには，プログラマは少なくとも，プログラムの動作について明確に理解しておく必要がある．しかしながら初心者の場合プログラムの個々の命令の振舞いを理解できても，プログラム全体の動作を正しく理解できなかったり，プログラム仕様から動作を想定できない場合がよく見られる．このような初心者を対象とする場合，動作とプログラムコード，動作とプログラム仕様の対応関係について説明することが重要となる．本論文では動作の理解が特に難しい再帰プログラムを対象に，プログラムの動作を介したプログラミングを支援する知的教育システムについて述べる．筆者らは再帰プログラミングのモデルを想定した上で，学習者にとって理解が容易となるようにプログラムの動作を表現し，これをプログラムの振舞いと呼んでいる．本論文ではこの振舞い表現を用いて，再帰プログラムの設計過程および理解過程を支援する方法について論じる．特に，雛形を用いた解法による設計過程の支援ならびに，振舞い表現の可視化による理解過程の支援について述べる．更に振舞い表現の評価実験についても述べる．

キーワード　プログラミング教育，ITS，プログラムの振舞い，プログラムの可視化

1. まえがき

プログラムを作成するためには，少なくともコンパイラやインタプリタの解釈，すなわちプログラムが計算機においてどのように動作するのかについて，明確に知っておかなければならない．プログラミングの初心者はifやforなどの個々の命令を比較的容易に理解できる．しかしながら，数行から数十行程度のプログラムであっても，その動作を正しく理解できない，またプログラム仕様からプログラムの動作を想定できない初心者がよく見られる．このような初心者を対象とするプログラミング教育では，プログラムの動作とソースプログラムの対応関係や，プログラムの動作と仕様の対応関係について説明することが重要と考えられる．

プログラムの動作過程の理解を支援するツールとしてトレーサなどがある．これらの支援ツールは，通常メモリの状態などを表示して，計算機におけるプログラムの具体的な動作を可視化する．このような情報によってプログラミングを支援する場合，計算機の仕組みやプログラミング概念を十分理解していることが前提となる．しかしながら，プログラミングの初心者は，これらを十分に理解していないため，トレーサなどの支援ツールを利用しても，表示される動作過程が意味することを理解できない場合が多い．このような初心者を対象とした教育では，プログラムの動作過程を理解容易となるように表現する必要がある．また，そのような表現が理解の妨げになる場合も考えられるため[7]，できるだけ学習者の誤解をまねかないように表現する必要がある．本論文では，プログラムの具体的な動作過程に対して，初心者のためにわかりやすく表現した動作過程をプログラムの振舞いと呼び，再帰プログラミングの教育における，適切な振舞いの表現方法について検討する．

再帰プログラムは，初心者にとってプログラムの動作過程の理解が困難となっている典型的なプログラムである．再帰プログラムは単純なものでも複雑な動作を行う場合が多い．従来の教育では，分割統治法など再帰プログラムを宣言的に作成する方法を説明することが中心となっている．しかしながら振舞いについては，個々のプログラムを事例的に説明することはあっても，仕様から成されるべき振舞いをどう予想するか，あるいはプログラムからどのように振舞いを理解するかということについての系統立てた説明は少ない．筆者らは，プログラム言語について学習を終えた

† 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

ばかりの初心者を対象に，特に再帰プログラミングを取り上げ，プログラム仕様と振舞い，振舞いとソースプログラムの対応関係について教育することを目的とした知的教育システムについて検討している．知的教育システムを実現するためには，プログラミングにおけるどういった情報を，いかに適切に伝えるのかについて検討しなければならない．振舞いに基づく再帰プログラミングの教育において，前者の問題は振舞いを表現する問題に相当し，後者の問題はその振舞い表現を用いた説明方法や教育方法の問題に相当する．本論文では前者を中心に述べ，後者の問題については別稿で述べる[5], [6].

筆者らは振舞いに基づく再帰プログラミングにおいて，プログラム仕様から成されるべき処理過程（振舞い）を考えながらプログラムを設計する過程と，プログラムからその振舞いを理解する過程との二つの過程からなるプログラミングのモデルを想定した．本論文ではこのモデルをもとに，再帰プログラムの振舞いを表現し，その表現を用いた再帰プログラムの設計過程および，理解過程を支援する枠組みについて述べる．

まず再帰プログラムの振舞いを表現するために，再帰プログラミングにおける初心者の誤りの要因について検討した．初心者の誤りの中には，再帰処理を反復処理と同じ処理であるように理解している誤りが多く見られる[1]．この誤りを修正する一つの方法は，再帰と反復との関係を明らかにすることである．また初心者は計算機に特有の概念に不慣れであるために，プログラムの振舞いを正しく理解できず，プログラミングに失敗していると考えられる．特に再帰呼出しの際にデータをスタックに保存し，再帰から戻る際にスタックからデータを取り出すといった，スタックを介したデータ操作を理解するのが難しいと考えられる．このような計算機に特有の処理を，わかりやすく説明できるように振舞いを表現することが不可欠である．

本論文では，以上のような観点から初心者にとって理解容易となるような振舞いの表現を新しく提案し，これをU-behaviorと名づけている．U-behaviorでは，再帰呼出しを一つしか含まないような構造をもつ再帰処理を対象として，再帰処理を二つの反復処理に明確に対応させて表現している．このU-behaviorは初心者向けの教科書で用いられるような再帰プログラムを十分に表現可能としている．また計算機特有の概念を初心者にとって親しみやすい概念に写像して表現している．

次に，初心者の再帰プログラミングの設計過程を支援する，雛形を用いたプログラミング設計解法U-solutionを提案する．これまでの研究で提案されている雛形は，複数のプログラムコードの共通構造で構成され，相違部分を空欄としたものである[2]～[4]．これらの研究では，まず雛形を選択し，次にすべての空欄に部品と呼ばれる適当な処理を代入することによりプログラムを完成させる．本論文で提案するU-solutionでは，複数の再帰プログラムに関するU-behaviorの共通部分を雛形とした．U-solutionではあらかじめ整理された部品をその雛形に代入することによりU-behaviorを作成する．次にプログラムコード用の雛形と部品を利用して，得られたU-behaviorから再帰プログラムを作成する．

更に，再帰プログラミングにおける理解過程を支援するU-behaviorの可視化について述べる．プログラムの理解過程は，プログラムから振舞いを抽出する過程と，その振舞いが仕様を満たすかどうかを判断する過程から構成される．プログラムから振舞いを抽出する作業では，プログラム上の処理がどのようにU-behaviorに対応するのかを知る必要がある．ここではわかりやすい概念を用いてU-behaviorでの処理過程を表現することにより，U-behaviorの理解を支援する方法について検討する．また振舞いから仕様を理解する作業を支援するために，U-behaviorにおけるデータの流れの可視化について検討した．ここでは入力データから出力データまでの処理過程をアニメーションを用いて動的に表現する．筆者らは既にこのようなアニメーションを提示するツールを作成している[5], [6].

最後に，本論文ではU-behaviorを評価する実験について述べる．実験ではU-behaviorをもとにしたプログラミ

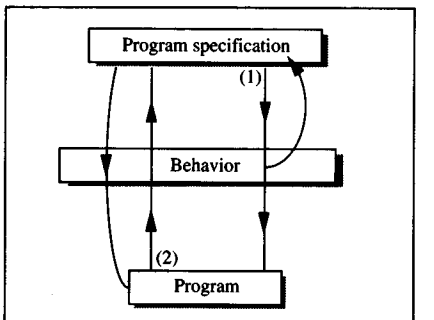


図1　U-behaviorに基づくプログラミングのモデル
Fig. 1　Model of programming with U-behavior.

ング過程の説明とプログラミング教科書における再帰プログラミングの説明を比較した．実験の結果，U-behaviorをもとにした説明が教科書の説明よりも効果があることを確認できた．

2. U-behaviorの設定

2.1 振舞いに基づくプログラミングモデル

筆者らは，振舞いを介したプログラミング過程として，図1のようなモデルを想定した．本モデルでは，振舞い(behavior)に基づくプログラミングをプログラムの設計過程(the process of designing programs)と理解過程(the process of understanding programs)とに分けている．プログラムの設計過程では，まずプログラム仕様(program specification)からなされるべき処理過程（振舞い）を想定する．その際にプログラム仕様を満足することを確認し，もし誤りがあれば振舞いを修正する．次に想定した振舞いからそれを実現するプログラムを作成する．一方プログラムの理解過程では，プログラムから振舞いを抽出してその振舞いがプログラム仕様を満足するかどうかを確認する．もし仕様を満足しない場合は，プログラムから振舞いを再度抽出する．

このようなプログラミングモデルをもとに，初心者の再帰プログラミングにおける設計作業および理解作業を支援するためには，初心者にとって理解容易なように再帰プログラムの振舞いを表現することが重要である．ここではその表現について述べる．

2.2 U-behaviorの設定

本論文では，二つの反復を用いて再帰の振舞いを表現する．図2は再帰プログラムの実行過程を表す．関数A(function A)は関数の中で一度だけ自分自身を呼び出す再帰関数である．関数Aを実行すると，関数の先頭から再帰呼出し(recursive call)までの間の処理が行われる（図3（1））．次に最初の再帰呼出しが行われる．このとき，関数内のすべての変数の状態はスタックと呼ばれるデータ構造に保存される．呼び出された二つ目の関数Aにおいて実行される処理は一つ目の関数と同様に，関数Aの先頭から再帰呼び出しまでの間の処理である．このように，再帰が停止するまでの間，関数の先頭から再帰呼出しまでの間の処理が繰り返し実行される．この繰り返しが1回目の反復 (the first iteration) である．また再帰関数には，再帰呼出しを行わずに関数を終了させるための条件判定が必ず存在する．この判定は再帰呼出しよりも必ず前に位置している．再帰が停止すると一つ前の関数に処理が戻り，再帰呼出しから関数の最後までの間の処理が行われる（図3（2））．このとき，スタックから保存しておいた変数の値を取り出す．以降同様に再帰呼出し直後から関数の最後までの間の処理が繰り返し実行される．これが2回目の反復 (the second iteration) である．

図2の再帰プログラムの振舞いの表現に対して，反復を上下方向に書き直して，データをノード，処理をリンクで表した振舞いの表現をU-behaviorと呼ぶ．U-behaviorを図4に示す．U-behaviorでは左の流れが1回目の反復を表し，右の流れが2回目の反復を表す．また左側の反復処理においてスタックに保存されたデータが右側の反復処理において取り出されるプロセスは左から右への移動を表すリンクによって表現される．このようにU-behaviorでは，再帰処理を二つの反復に対応させる．そのため，表現可能な再帰プログラムは，再帰呼出しが一つのプログラムに限定される．

U-behaviorでは再帰プログラムの振舞いを四つの要素に分類している．一つ目の要素は1回目の反復において繰り返される処理で，これを「再帰の前処理 (Pre-recursion sub-procedure)」と呼ぶ．二つ目の要素は，再帰関数を終了させるための入力データの条件で，これを「再帰の停止条件 (Terminating Condition)」と呼ぶ．例えば入力データが0になったら再帰を停止させる場合

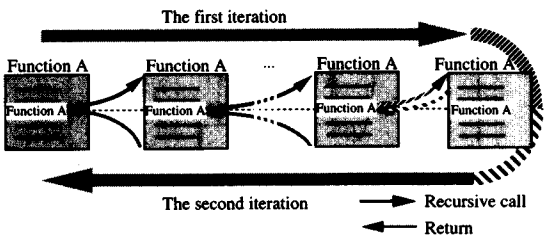
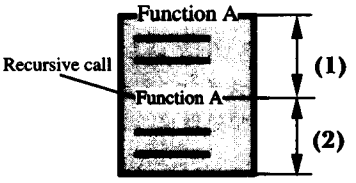


図2 再帰関数の実行過程
Fig. 2 The process of recursive function.



(1) The first iteration
(2) The second iteration

図3 再帰関数
Fig. 3 Recursive function.

は，「入力＝0」が再帰の停止条件である．三つ目の要素は，この停止条件が成立した際に実行される出力データ（再帰関数の戻り値）の設定で，これを「出力の初期化 (Return value initialization)」と呼ぶ．四つ目の要素は，２回目の反復において繰り返される処理で，これを「再帰の後処理 (Post-recursion sub-procedure)」と呼ぶ．

U-behaviorでは，これらの四つの要素を部品と呼ぶ．次のPrologプログラム (注1) は，入力リストの末尾要素を取り除いた残りのリストを出力する再帰プログラムである．

```
delete_last( [A], [] ).
delete_last( [A¦B], [A¦C] ):- delete_last( B, C ).
```

このプログラムの振舞いをU-behaviorで表現すると図５のようになる．図では，初心者にもわかりやすいように，データ構造を数字のリストとして表している (注2)．再帰の前処理では，「入力リストの先頭要素を取り出す」処理を行っている．停止条件は，「リストの長さ＝１」である．すなわち，１回目の反復処理により，入力リストからリスト要素数が１になるまで，先頭要素を一つずつ取り出す．出力の初期化では，「空リスト（要素なし）」で初期化が行われる．再帰の後処理は，前処理で取り出した先頭要素を，「出力リストの先頭に追加」する処理である．従って２回目の反復によって，１回目の反復で取り出された要素が一つずつ先頭に追加される．入力リストの末尾要素は１回目の反復で取り出されていないため出力には含まれない．このようなU字型の一連の処理によって，入力リストの末尾要素が削除される．

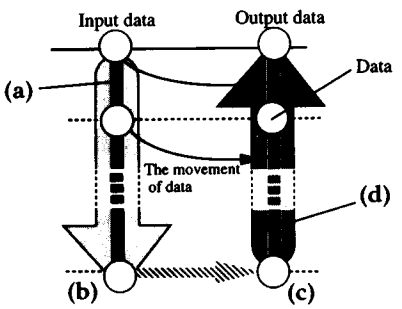


図４　再帰関数の実行過程を表すU字型表現
Fig. 4　U-behavior of recursive function.

3. 再帰プログラミングの教育支援

ここではU-behaviorをもとに，まず初心者の再帰プログラムの設計支援を行うU-solutionについて述べる．次に理解支援を行うU-behaviorの可視化について述べる．

3.1 U-solutionの提案

筆者らは初心者の再帰プログラムの設計を支援するために，プログラム仕様から再帰プログラムを作成するための解法としてU-solutionを提案する．U-solutionでは，複数の再帰プログラムのU-behaviorから共通構造を抽出し，抽出した雛形をU-templateと呼ぶ．このU-templateはU-behaviorにおける四つの構成要素を空欄とした振舞い表現である．この空欄に部品を代入することでU-behaviorを作成する．PrologとC言語の初心者向けの教科書でよく用いられている再帰プログラムから抽出した部品を表１～表３に示す．

プログラム仕様からU-behaviorを作成するには表（表１～表３）の中から適当と思われる部品を選びU-

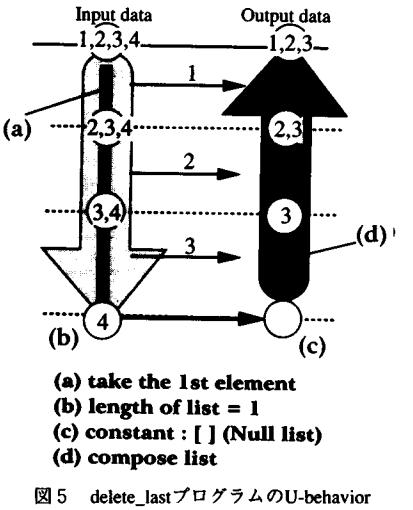


図５　delete_lastプログラムのU-behavior
Fig. 5　U-behavior of delete_last recursive function.

（注1）：U-solutionでは，再帰プログラムを手続き的に設計していると言えるが，Prologは宣言的なプログラミングを特徴とする言語である．しかしながら現在のPrologコンパイラやインタプリタは，このような宣言的プログラミングを可能にするまでには至っていないことからも，Prologでも手続き的な解釈によるプログラミングの教育が重要であると考えられる．

（注2）：C言語のポインタや，Prologのリストなど，複雑なデータ構造を扱うことは，初心者にとって非常に大きな負荷となる．このようなデータ構造の教育は，プログラミング教育の重要な問題の一つである．しかしながら，本論文では再帰プログラムの振舞いの教育に焦点を当てるために，これらの負荷要因をできるだけ軽減したいと考え，データ構造としてポインタや構造体は用いないで，比較的理解しやすい文字や数値を要素とする線形リストを用いた．

表1 U-templateにおける再帰の前処理と再帰の後処理の部品
Table 1 Parts of U-template for pre-recursion sub-procedure and post-recursion sub-procedure.

Part	Icon	Prolog description	C description
copy	bat (go by)	$V1_n = V2_n$	$V1_n = V2_n$;
take first element	bat (hit) ball = first element	$V1_n = [Head_n \mid V2_n]$	*head_list* ($V1_n$, $Head_n$); *tail_list* ($V1_n$, $V2_n$);
compose list	glove (catch) ball = first element	$V2_n = [Head_m \mid V1_n]$	*compose* ($Head_m$, $V1_n$, $V2_n$);
plus K	$+K$	$V1_n = V2_n + K$	$V1_n = V2_n + K$;
plus $Input_m$	$+In_m$	$V1_n = V2_n + Head_m$	$V1_n = V2_n + Head_m$;
subtract K	$-K$	$V1_n = V2_n - K$	$V1_n = V2_n - K$;
multiple $Input_m$	$\times In_m$	$V1_n = V2_n * Head_m$	$V1_n = V2_n * Head_m$;

表2 U-templateにおける再帰の停止条件の部品
Table 2 Parts of U-template for terminating condition.

Part	Icon	Prolog description	C description
none	signal : go	——	——
length = 0	signal : stop (Cond. no ball)	$In_n = []$	null_list(In_n)
length = 1	signal : stop (Cond. only one ball)	$In_n = [AElement_m]$	one_element_list(In_n)
number = 0	signal : stop (Cond. number = 0)	$In_n = 0$	$In_n == 0$
number = 1	signal : stop (Cond. number = 1)	$In_n = 1$	$In_n == 1$
number = K	signal : stop (Cond. number = K)	$In_n = K$	$In_n == K$

表3 U-templateにおける出力の初期化の部品
Table 3 Parts of U-template for return value initialization.

Part	Icon	Prolog description	C description
$Input_m$	magnet (Object : $Input_m$)	$Out_n = In_m$	$Out_n = In_m$
element of $Input_m$	magnet (Object : element of $Input_m$)	$Out_n = AElement_m$	$Out_n = AElement_m$
Constant : A	finger (point to A)	$Out_n = A$	$Out_n = A$

templateの四つの空欄に代入する．次にU-behaviorにおける入力データと出力データの関係を調べることにより，プログラム仕様を満足するかどうかを確認する．もしプログラム仕様を満足していない場合は，部品の選択をやり直す．プログラム仕様を満足すれば，完成したU-behaviorから再帰プログラムを作成する．このときProgram-templateを用いる．これはU-templateをプログラム言語で記述したものであり，現在のところ図6に示すようにPrologとC言語の2引数と3引数のProgram-templateを用意している．また部品には，U-behavior上の処理と，その処理に対応する各プログラム言語の記述があらかじめ設定されている（表1～表3）．各部品に対応するプログラム言語の記述はひととおりではないが，ここでは表1～表3に示すように，初心者にとってわかりやすいと思われる，最も単純な記述を用いることとした（C言語の部品において用いられているリスト処理の関数の仕様を付録1.に示す）．U-templateで選択した部品がもつプログラム言語の記述をProgram-templateに代入することにより再帰プログラムを完成させる．

以下，U-solutionを用いてchange_lastプログラムを作成する手順を述べる．change_lastのプログラム仕様

は，「入力リストの末尾要素を文字"a"に置換したリストを出力する」である．プログラム言語はCプログラムとPrologプログラムとする．

手順１：まず，U-templateに適切な部品を代入する．もし初心者が末尾要素以外の要素がそのままの形で出力されることに気が付けば，再帰の前処理の部品が先頭要素を取り出す処理，再帰の後処理の部品が先頭に追加する処理であることがわかる．末尾要素を操作することから，停止条件がリストの長さ１，初期化を"a"にすれば仕様を満足する．また初心者が図５のdelete_lastプログラムのU-behaviorを既に知っている場合は，change_lastもdelete_lastも入力リストの末尾要素を操作する点においてよく似ていることから，図５のU-behaviorの出力の初期化部品だけを変更すればよいことに気づく場合が考えられる．このように，ある程度はプログラム仕様から部品の見当をつけることができる．U-solutionについてよく理解できてなく，全く見当がつかないような場合は，表１～表３の部品群から試行錯誤して部品を探すことになる．このような見当がつくようになるためには，経験を積まなければならない．

手順２：U-behaviorが得られたら，次に入力値として適当な値を設定して，正しい出力が得られるかどうか確認する．例えば入力として“1,2,3,4”を与えた場合に，U-behaviorをたどって出力“1,2,3,a”が得られるかどうかを確認する．もし，出力がプログラム仕様を満足しない場合は，手順１に戻って部品の選択をやり直す．このとき慣れてくれば，出力の誤り方によって修正個所を予想することができる．例えば，“1,2,3,4,a”という出力であれば，前処理において先頭要素を取り出しすぎている，すなわち停止条件の部品について検討し直せばよい，といった予想が可能となる．

手順３：次に作成したU-behaviorに対応する再帰プログラムを作成する．Prologの再帰プログラムを作成するために，図６に示すPrologのProgram-templateに部品を代入する．代入する部品は，手順１で選択した部品に対応するPrologの記述である（表１～表３）．Program-templateのすべての空欄に部品を代入すると求めるPrologの再帰プログラムが完成する．C言語の再帰プログラムを作成する場合も同様にして，C言語のProgram-templateに，手順１で選択した部品におけるC言語の記述を代入する．このようにして完成したProlog，Cプログラムを図７に示す．

3.2 U-behaviorの可視化

再帰プログラムの理解作業は，再帰プログラムからU-behaviorを抽出する作業と，プログラム仕様との確認を行う作業である．再帰プログラムからU-behaviorを抽出する作業は，プログラムと振舞いの対応付けを行う作業である．ここではこの作業を支援するために，わ

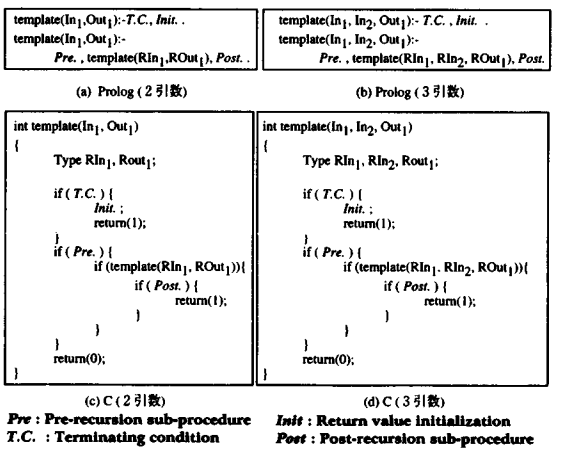


図６　PrologとC言語のProgram-template
Fig. 6　Program-templates of Prolog and C.

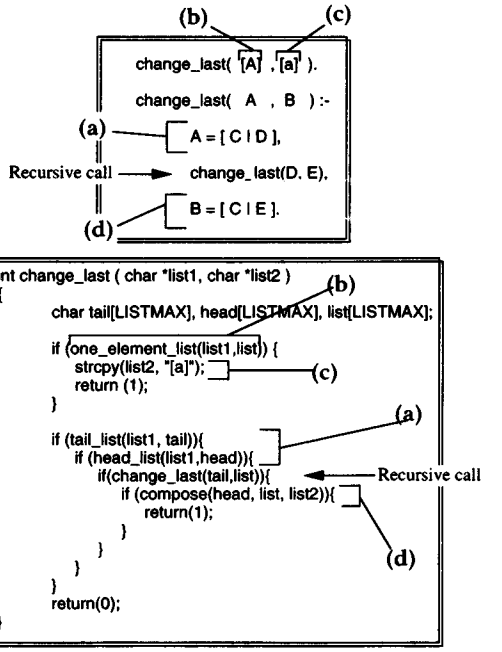


図７　change_lastプログラム（Prolog, C）
Fig. 7　change_last program in Prolog and C.

かりやすい概念を用いてU-behaviorを表現する方法について述べる．また振舞いからプログラム仕様を理解する作業を支援するために，アニメーションを利用したU-behaviorの可視化方法について述べる．

3.2.1 写像を利用したU-behaviorの表現

プログラムから振舞いを理解するためには，さまざまな計算機特有の概念が必要である．U-behaviorでは，リスト処理など初心者にとって理解が困難と思われる部品を親しみやすい概念に写像して表現する(注3)．各部品はアイコンで表示される．図8は再帰プログラムにおいて典型的な操作である入力リストから先頭要素を取り出す処理を，野球の世界に写像して表現した例を示している．この部品の振舞いは「バットでボールを打つ」アイコンで表現される．またこの処理によって取り出された先頭要素はボールで表される．

3.2.2 U-behaviorのアニメーション表示

U-behaviorにおける各部品にあらかじめアニメーション表示機能を設定することにより，初心者が作成したU-behaviorの様子を自動的にアニメーション表示することができる．筆者らはU-behaviorにおいて，入力データから出力データが生成される過程をアニメーション表現するトレースツールを作成している．本ツールで表現可能な範囲は，U-behaviorの対象範囲，すなわち再帰呼出しを一つしか含まない再帰プログラムに限定される．

本ツールでは，U-behaviorのアニメーション表示を行うために，以下に示すような属性をもつリストを各部品の内部表現としている．

[部品タイプ，部品名，アニメーション関数，
Cプログラム表現，Prologプログラム表現]

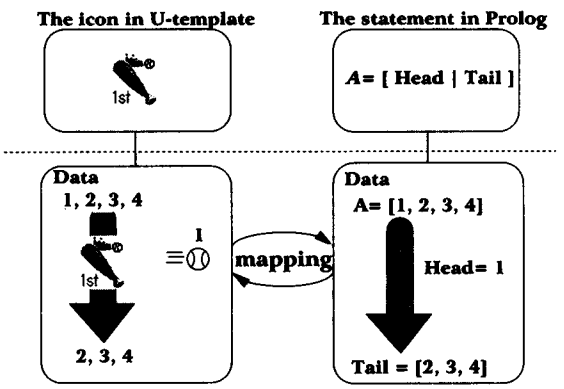


図8 野球への写像の例
Fig. 8 An example of mapping U-behavior into baseball world.

ここで，部品タイプはU-behaviorにおける4種類の構成要素を表し，アニメーション関数は部品の振舞いをアニメーション表示する関数である．

図9に，二つの文字列“1234”と“5678”を結合するプログラム(append)を入力したときのトレースツールの画面を示す．appendプログラムに必要な部品は，再帰の前処理では入力1（一つ目の入力データ），入力2（二つ目の入力データ）に対してそれぞれ「先頭取り出し」，「操作なし」，再帰の停止条件はそれぞれ「文字列の長さが0」，「なし」である．また出力の初期化が「入力2を複写」であり，再帰の後処理が「入力1を先頭に追加する」となる．

すべての部品を選択して，実行ボタンを押すと，まず上から下に文字列が流れ，「再帰の前処理」の部品がこの文字列に対して操作を行う．appendでは，「バット」アイコンで文字列を打ち続けて，文字列が次第に短くなる様子をアニメーション表現する．最終的に文字列がなくなると，停止条件「文字列の長さ0」を表す信号機のアイコンによって反復が停止する．次に出力の初期化の「入力2を受け取る」部品を表す「磁石」アイコンによって入力2が出力側に移動する．次に再帰の後処理の「入力1を先頭に追加する」部品を表す「グローブ」アイコンによって，前処

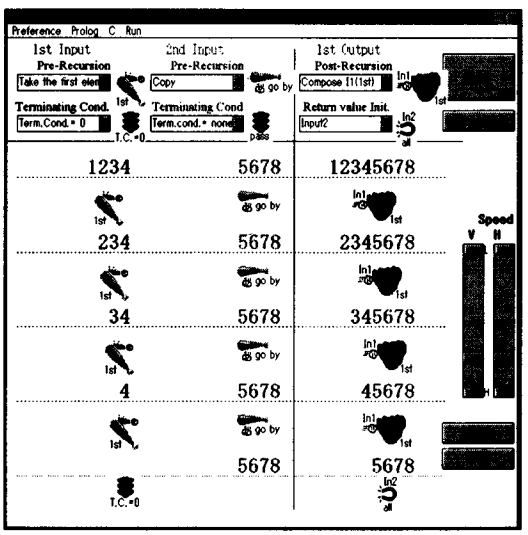


図9 トレースツールの画面（append）
Fig. 9 U-behavior animation in trace tool (append).

(注3)：写像した表現を初心者に与えることによって，初心者のプログラミングの弊害とならないように，適切な表現を設定しなければならない．本論文で対象とする範囲においては，実験により弊害となることは少ないことを確認している．

理の「バット」アイコンによって飛ばされたボール（先頭文字）を受け取る．このボールの移動は，再帰が実行されるたびに行われる，「スタックによるデータの移動」を表現している．このとき出力の文字列の前にこの先頭文字が挿入される．この操作を下から上まで繰り返し，最終的に一番上に到達すると，文字列の結合が完成する．

3.3 U-behaviorに基づくプログラミング教育の特長

市販されているプログラミング教科書では，ほとんどの場合単純な再帰プログラムを数個例に挙げて個々にその実行過程の図を示しながら説明しているが，解法としてプログラミングの方法を説明したり，統一されたわかりやすい表現の上で，再帰プログラム間の類似点や相違点を説明したものはほとんど見られない．一方，U-behaviorに基づくプログラミング教育では，再帰呼出しを一つしか含まないような再帰プログラムに限定されるものの，設計解法を与えることができる．また再帰プログラム間の共通点や差をそれぞれのプログラムにおけるU-behaviorの部品の違いとして説明することができる．例えば以下に示すPrologのappendとreverse_appプログラムについて考える．

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

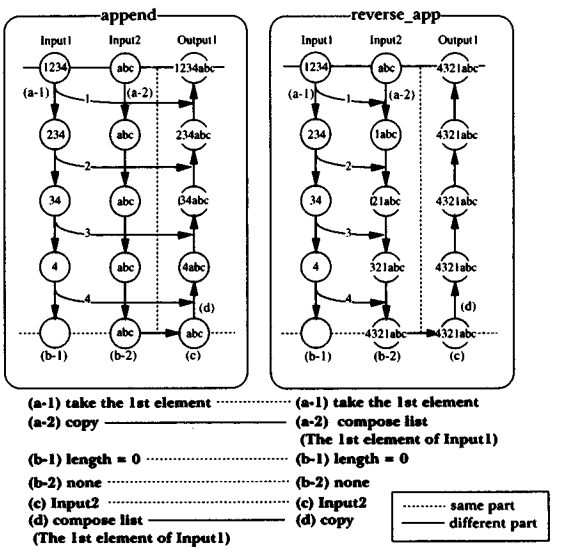


図10 Prologのappendとreverse_appのU-behaviorの違い
Fig. 10 The difference in U-behavior between append and reverse_app in Prolog.

appendプログラムのプログラム仕様は「第１引数のリスト（入力１）と第２引数のリスト（入力２）を結合したリスト（出力３）を第３引数とする」であり，reverse_appプログラムのプログラム仕様は「第１引数のリスト（入力１）のリスト要素を反転して第２引数のリスト（入力２）と結合したリスト（出力３）を第３引数とする」である．各仕様に対するプログラムの振舞いは図10のようになる．図に示すように，これらのプログラムにおける処理過程の共通点，相違点は部品を用いて表される．例えば共通点は，前処理で入力１(Input1)の先頭要素を取り出し，要素数＝０で繰り返しを停止させる点，入力２(Input2)で出力を初期化する点である．また相違点は，appendプログラムでは，入力２の前処理で複写，再帰の後処理で要素を先頭に追加するのに対して，reverse_appプログラムでは入力２の前処理で先頭に追加し，再帰の後処理で複写を行っている．このような違いは，入力１のリスト要素の順序をそのままにして入力２と結合するappendプログラムと，入力１のリスト要素の順序を反転して入力２と結合するreverse_appプログラムのプログラム仕様の違いに対応づけられる．

このように，各プログラム間の処理の類似点や相違点をU-behaviorの部品を用いて体系的に説明することができる．またこのような説明は各部品がプログラム仕様に与える影響を理解することを支援するものとなり，これによって学習者はプログラム仕様の違いから用いるべき部品を推測することができるようになることが期待される．このような推測はプログラムの設計や理解において有効に働くと考えられる．

またU-behaviorは特定のプログラム言語に依存しないで再帰プログラムの振舞いを表している．そのために，U-behaviorの部品に対して言語ごとのプログラムの部品を用意するだけで，複数の言語において統一した再帰プログラミングの設計支援，理解支援を実現することができる．

筆者らは，以上のような特長を生かす再帰プログラミングのための知的CAIシステムを既に開発し，報告している[5], [6].

4. U-behaviorの評価実験

本論文で提案したU-behaviorを評価する実験を行った[5]．本実験では，市販されているプログラミング教

表4 実験結果
Table 4 Results of the experiment.

			N	Mean	SD
Prolog	textbook	1st	4	19.8	10.14
		2nd	4	10.4	4.12
		total	8	15.1	8.06
	U-behavior	1st	4	24.0	6.98
		2nd	4	56.3	14.00
		total	8	40.1	13.49
C	textbook	1st	4	46.9	10.23
		2nd	4	57.3	11.69
		total	8	52.1	11.27
	U-behavior	1st	4	89.6	3.32
		2nd	4	68.8	10.34
		total	8	79.2	9.17

科書における説明と，U-behaviorに基づくプログラムの説明との比較を行った．更に再帰プログラムの振舞いの説明方法がプログラム言語に依存せず効果があることを確かめるためにPrologとC言語を用いて実験を行った．実験の被験者は再帰プログラムの初心者である大学生と大学院生16名である．

4.1 実験方法

被験者を8名ずつの二つのグループに分け，一方のグループはプログラミング教科書の再帰プログラムの説明文をそのまま読んでもらった．もう一方のグループには，筆者らが作成したU-solutionの説明文を読んでもらった．この説明文にはすべてプログラムがU-behaviorによって表現されている．両グループともほぼ同じ量の説明を与えた．また両グループとも全く同じプログラムについて説明した．また，どちらのグループも理解できない点については質問できることとし，説明文を完全に理解してもらうようにした．

使用したプログラム言語はC言語とProlog言語で，言語の順序がテストの得点に与える影響を減らすために，それぞれのグループを更に半分に分けて，一方のグループにはC言語，Prologの順に，残り半分にはProlog，C言語の順とした．

各言語における説明文を両グループともに40分間与え，その直後にプログラム仕様から再帰プログラムを作成する試験を行った．問題は両グループとも共通とし，C言語，Prologそれぞれ3問ずつの計6問，時間は1問10分とした．試験問題の一部を付録2.に示す．

4.2 実験結果と考察

試験の採点は，被験者の作成したプログラムを対応するU-behaviorにおける部品に分解し，正解した部品の数を得点とした．また，綴りミスや明らかに文法上の誤りと考えられるものは減点しないものとした．採点結果を表4に示す．平均点の括弧内の数値は100点満点に換算した得点である．採点の結果，C言語，Prolog言語，どちらの言語においてもU-behaviorの説明文を読んだグループの方が平均点が上回った．

表5 分散分析表
Table 5 Analysis of variance.

	s.v.	*s.s.*	*d.f.*	*m.s.*	*F*
Prolog	*A*	576	1	576.0	4.083†
	error	1975	14	141.1	
	total	2551	15		
	s.v.	*s.s.*	*d.f.*	*m.s.*	*F*
C	*A*	676	1	676.0	5.607*
	error	1688	14	120.6	
	total	2364	15		

†$p < 0.1$ *$p < 0.05$

更に平均点の違いを詳しく調べるために，各言語について説明文の種類を要因とする1要因の被験者間分散分析を行った．分散分析表を表5に示す．その結果，Prologでは10%，Cでは5%の有意差を得た．従ってグループの違いによる平均点の違いは，説明文の種類による効果と考えることができる．またプログラム言語に依存せずU-solutionの効果を確認できた．

5. むすび

本論文では，プログラム言語について学習を終えた初心者プログラマに対して，プログラムの振舞いを介して再帰プログラミングを教えるための振舞いの表現について述べた．更にその表現を用いたプログラムの作成支援および理解支援の枠組みについて述べた．

本論文で提案したU-behaviorは，再帰呼出しを一つもつ再帰プログラムの振舞いを反復に対応づけて表現する．U-solutionは，U-behaviorを基に再帰プログラムの作成を行う解法である．また，本論文ではU-behaviorを可視化することで再帰プログラムの理解を支援する方法を述べた．更にU-behaviorの有効性を確かめるために，市販されている教科書における再帰プログラミングとU-solutionとの比較実験を行った．その結果有意に効果的であることを確認できた．

筆者らは本論文で述べた再帰プログラムの設計・理解支援を適切に運用するシステムを既に開発しており，これについては別稿で述べたい．

今後の課題には，U-behaviorが初心者のプログラム理解の弊害となる場合の分析および，プログラミング教育の観点からのプログラム一般における振舞いの可視

化が挙げられる．

謝辞　実験に御協力を頂いた徳島大学工学部知能情報工学科矢野研究室の諸氏，ならびに大阪大学産業科学研究所知能システム科学研究部門知能アーキテクチャ豊田研究室の諸氏に感謝致します．

文　献

[1] S.Wiedenbeck, "Learning iteration and recursion from examples," Int. J. Man-Machine Studies, 30, pp.1-22, 1989.

[2] T.S, Gegg-Harrison, "Learning Prolog in a schema-based environment," Instructional Science, vol.20, pp.173-192, 1991.

[3] T.S, Gegg-Harrison, "Adapting Instruction to the Student's Capabilities," JI. of Artificial Intelligence in Education, pp.169-181, 1992.

[4] K.Itoh, M.Itami, K.Fukukawa, J.Muramatsu, and Y.Enomoto, "A workbench system for novice prolog programmers: Visually-structured interactive tracer and prototype-based programming support," IEICE Trans. Inf. & Syst., vol.E77-D, no.1, pp.57-67, Jan. 1994.

[5] N.Matsuda, A.Kashihara, T.Hirashima, and J.Toyoda, "An instructional system for constructing algorithms in recursive programming," Proc. of the Sixth International Conference on Human-Computer Interaction, (HCI International '95), pp.889-894, Tokyo, Japan, 1995.

[6] 松田憲幸，柏原昭博，平嶋　宗，豊田順一，"プログラミングにおける再帰概念の形成の支援を行う教育システムの開発，" 信学技報，ET94-12，1994.

[7] Y.He，池田　満，溝口理一郎，"プログラミングITS構築のための初心者の概念ギャップの分析，" 人工知能学会誌，vol.10，no.3，pp.436-445，1995.

付　録

1. C言語のProgram-templateのための部品における関数

int head_list(char *A, char *B)：リストBの先頭要素Aを求める関数．

int tail_list(char *A, char *B)：リストBの先頭要素以外のリストAを求める関数．

int compose(char *A, char *B, char *C)：リストBの先頭にリストAを追加したリストCを求める関数．

int null_list(char *A)：リストAが空理ストかどうかを判定する関数．

int one_element_list(char *A)：リストAが１要素からなるリストかどうかを判定する関数．

但し，リストは文字列で表され，表記はPrologと同じとする．また各関数の戻り値は，処理に成功すれば１，失敗すれば０である．

2. 評価実験で使用した試験問題の一部

問題　Cプログラムのテストです．

階乗を求める再帰関数factorialを作成して下さい．

nの階乗は，1からnまでの自然数を掛けたものです．

4の階乗＝１×２×３×４＝２４

入出力例：factorial(3) = 6, factorial(5) = 120

問題　Prologプログラムのテストです．

２つのリストのうち，最初のリストの末尾要素を取り除いて，他方のリストとつなぐ再帰Prologプログラムappend_delを作成して下さい．

述語 append_del(A, B, X) ：　リストAの末尾要素を削除して，リストBとつないだリストがXである．

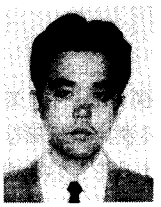
入出力例：append_del([1,2],[3,4],X). -> X = [1,3,4]

append_del([1,d,3],[4,n],X). -> X = [1,d,4,n]

（平成7年10月5日受付，8年3月21日再受付）

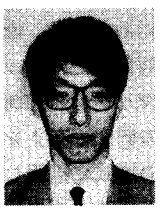
松田　憲幸　（学生員）

平４関西大・工・管理工卒．平６阪大大学院博士前期課程了．現在，同大大学院後期課程在学中．Information FilteringおよびITSの研究に従事．情報処理学会，人工知能学会，教育システム情報学会各会員．



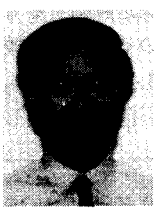
柏原　昭博　（正員）

昭62徳大・工・情報卒．平１同大大学院修士課程了．平４阪大大学院博士後期課程了．同年阪大産業科学研究所助手．工博．現在，人工知能，特に説明，ITS，プログラム理解の研究に従事．1993年度人工知能学会全国大会（第７回）優秀論文賞受賞．情報処理学会，人工知能学会，教育システム情報学会各会員．



平嶋　宗　（正員）

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．平８講師．現在に至る．工博．人間を系に含んだ計算機システムの高度化に興味をもっており，特にIntelligent Tutoring SystemおよびInformation Filteringの研究に従事している．1993年度人工知能学会全国大会優秀論文賞，Outstanding Paper Award at ED-MEDIA'95受賞．情報処理学会，人工知能学会，教育システム情報学会，教育工学会，AACE各会員．



豊田　順一　（正員）

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47阪大産業科学研究所教授．工博．現在，桂枝雀にほれ込み，some rules and imaginationsとVirtual Fidelityについて調べている．情報処理学会，人工知能学会，日本認知科学会各会員．