再帰プログラミングを対象とした
学習支援システムに関する研究

1997年1月

松田憲幸

再帰プログラミングを対象とした
学習支援システムに関する研究

1997年1月

松田憲幸

内容梗概

本論文は，筆者が大阪大学大学院基礎工学研究科博士後期課程（物理系専攻情報工学分野）在学中に行った，再帰プログラミングを対象とした学習支援システムに関する研究の成果をまとめたものであり，以下の５章をもって構成されている．

第１章は序論であり，本研究の目的および意義について述べ，本研究により得られた諸成果を概説している．

第２章では，再帰プログラムの動作過程を表現する方法について述べている．まず，再帰プログラミングにおける学習者の誤りについて整理し，誤りを起こす原因について考察する．さらに，これらの誤り原因を踏まえて，再帰プログラムの動作過程の表現方法としてU-behaviorを提案する．最後に，U-behaviorの評価を行うための比較実験について述べる．実験の結果，U-behaviorの有効性が示された．

第３章では，学習支援システムにおける比較を促す問題演習機能の設計方法について述べている．学習支援システムの設計においては，蓄積した教育行動のもととなるデータに付加する，学習内容･方針を反映した問題表現が不可欠となる．まず，第２章で述べたU-behaviorを用いた問題表現の設定について述べる．さらに，U-behaivorのアニメーション表示機能，比較を促す問題演習機能として，問題間の差の説明機能，問題の検索機能，分類問題の作成機能について述べる．最後に，これらの機能を実装したシステムを評価するために行った実験について述べる．実験の結果，本システムが有望であることを確認できた．

第４章においては，本研究で得られた成果をまとめ，今後に残された課題について述べる．

関連発表論文

A. 学会誌投稿論文

1 松田憲幸，柏原昭博，平嶋宗，豊田順一："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"，電子情報通信学会論文誌，D-II Vol. J80-DII, No.1, (1997). (掲載予定)

2 松田憲幸，柏原昭博，平嶋宗，豊田順一："再帰プログラミングを対象とした問題の比較に基づく問題演習機能の実現"，教育システム情報学会，(条件付き採録)

B. 国際会議発表論文

1 Matsuda, N., Kashihara, A., Hirashima, T., Toyoda, J.:"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).

C. 研究会発表論文

1 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支援システムの開発", 電子情報通信学会技術研究報告 KBSE93-20, pp.9-16, (1993-09).

2 松田憲幸，柏原昭博，平嶋宗，豊田順一："Prologプログラムを対象としたスキーマの獲得支援方法について", 電子情報通信学会技術研究報告 ET94-12, pp.85-92, (1994).

D. 全国大会発表論文

1 松田憲幸，柏原昭博，平嶋宗，豊田順一："プログラム間差異を利用したプログラム理解支援 -Prologを対象とした差異の抽出メカニズム-", 人工知能学会全国大会（第７回） 論文集, pp.797-800, 1993.

2 松田憲幸，柏原昭博，平嶋宗，豊田順一："再帰プログラミングを対象とした教育システムの設計", 教育システム情報学会 第２０回全国大会, pp.55-58, 1995.

目次

図目次

表目次

第1章

序論

学習は高度な知的タスクであり,その達成は人間にとっても必ずしも容易なものではない.このため学習を補助する目的で,教科書やノート,黒板,VTRなど様々な道具を用い,さらにテキストや音,静止画,動画など,各メディアの特徴を効果的に利用する工夫が行われている.近年,このような学習支援を行うツールとして計算機が注目されている.計算機における通信・メディア利用能力の進歩は目覚ましく,教育現場への導入もより容易となる環境が整いつつある.このような目的で利用される計算機システムを学習支援システムと呼ぶ.

学習支援システムの対象領域としては,数学や語学などの学校教育の領域から,プラントの運転支援といった社会人教育まで,多岐にわたっており,各領域の特徴及び必要性に応じて様々な研究・開発が行われている[Sleeman 82], [Wenger 87], [Mandl 88], [Polson 88], [Burns 91], [Goodyear 91]. 中でもプログラミングに関する学習支援は,(1)プログラマ養成に対する社会的要請の急増,(2)プログラミング学習自体の難しさ,(3)計算機利用における親和性,などのために最も重要な研究対象となっている.

従来のプログラミングに関する学習支援システムの多くは,学習者の作成した

プログラムの診断・バグの発見とその修正指導に重点をおいていた．たとえば，SolowayらのPROUSTは，プログラムの典型的な断片を整理した表現と，各断片に対して初心者プログラマが犯しやすい誤りの分析結果をもとにして設計された学習支援システムであり，初心者プログラマを対象としたプログラムの誤りの診断とその修正指導を行うことができる[Soloway 84],[Johnson 85]．また，上野らは，アルゴリズムを複数の抽象レベルで階層的に記述した手続きグラフを用いることにより，プログラムの誤りをコードレベルだけではなく，複数の抽象レベルにおいて診断することのできるシステムINTELLITUTORを開発している[上野 87], [Ueno 94].

これらの学習支援システムは，いわばデバッグに焦点を当てたものであり，その意味において十分な成果を挙げているということができる．しかしながら，プログラミングに必要な能力はデバッグだけではない．プログラミング能力の適切な向上を図る上で，プログラムを理解する能力は非常に重要なものであるといえる．さらに，プログラムの設計手法の習得も不可欠である．本研究では，小規模でかつ基本的ではあるが初心者プログラマにとって，その理解と設計が極めて困難であるといわれている再帰プログラムに焦点を当て，その理解と設計に関する学習支援の高度化を目指したものである．具体的には，再帰プログラムの振舞いをわかりやすく説明する表現としてのU-behaviorを提案し[松田 97a]，この表現を用いることによって，再帰プログラムの振舞い理解支援[Matsuda 95]，仕様からのプログラム作成支援，およびプログラム間の振舞いにおける異同の理解を指向した問題演習[松田 97b]を実現している．

プログラマがプログラムの振舞いを理解することは，プログラミングにおいて最も重要なタスクの一つである．学習者のプログラムの振舞い理解を支援するためには，まずその振舞いをわかりやすい形で表現することが求められる．再帰プログラムは，プログラム中の命令のシーケンスとその振舞いが大きく異なるために，学習者にとって特にその振舞いの理解が困難なプログラムとなっているため，

振舞いをわかりやすく表現することの必要性は極めて大きいといえる.再帰プログラミングの学習支援に関する先行研究としてTimothy[Timothy 91],[Timothy 92], Bowles[Bowles 93]らの研究を挙げることができる．ここでは，Prologプログラムの典型的な処理を，プログラマが利用容易な形態に整理・部品化することによって支援を実現している．しかしながら，ここで整理されている表現は，Prologプログラムにおける記述をほとんどそのまま利用したものであり,再帰プログラムの振舞いを表現したものとはいえない．そのため，振舞いを理解できている学習者に対してはうまく支援を行えるが,振舞いを正しく理解できていない初心者には支援が困難となっている．

本研究では，まず，初心者の再帰プログラムの振舞い理解における困難さについて考察した．次にこれをもとに，学習者にとって理解容易となるような再帰プログラムの振舞い表現U-behaviorを提案した．再帰プログラミングにおける初心者の誤りの多くは,反復との混同や変数の扱い方に関するものである[Wiedenbeck 89],[谷川 96]．U-behaviorでは，反復との違いを明確にし，変数の流れをわかりやすく表現するために,再帰呼び出しを一つしか含まない再帰プログラムを２つの反復に対応させて表現する．さらにU-behaviorにおける雛形を整理し，雛形と代入する部品を利用した再帰プログラミング解法U-solutionを設定した．

さらに，U-behaviorを用いた仕様からのプログラム作成法の習得を目的とした問題演習支援についても検討している.従来のプログラミングに関する問題演習では,問題間の異同については必ずしも十分な注意が払われていなかった[Itoh 94], [海尻 95]．このため，学習者は個々の問題を独立したものであるかのように解いてしまい,過去の類似問題の解決結果を利用するといったことができない場合が多かった．問題演習をより効率的なものとするためには，問題間の異同を制御し，また必要に応じて学習者に対して説明することができなくてはならない[Polya 54], [Hirashima 94]．ここでは，U-behaviorを用いた仕様からのプログラム作成を前提として，U-behavior上における異同を制御し，また必要に応じて学習

者に対してその異同を説明する機能を備えた問題演習支援を実現している.

以下, 第2章では, プログラミング教育, 特に再帰プログラミングの問題点を整理し, 提案する再帰プログラムの振舞い表現U-behaviorについて述べる. さらにU-behaviorの評価を行うために, 市販されているプログラミング教科書における表現との比較を行った実験について述べる. 実験の結果, U-behaviorが有望であることを確認できた.

第3章では, 第2章で提案したU-behaviorにもとづく, 再帰プログラミング問題演習支援システムの設計について述べる.特に比較を促す問題演習機能を取り上げ, 問題間の差の説明機能, 問題の検索機能, 分類問題の作成機能の設計について述べる. さらに, これらの機能を実装した問題演習支援システムを評価するために行った, 予備的な実験について述べる. 実験の結果, 本システムが有望であることを確認できた.

最後に, 第4章において本研究を総括する.

第 2 章

再帰プログラムの振舞い表現

2.1 緒言

プログラミングの学習において,プログラムの理解能力の向上とプログラムの設計手法の修得は,ともに不可欠である.本章では,初心者にとって理解と設計が極めて困難といわれている再帰プログラムについて,その振舞いの理解支援に必要となる振舞いの表現と,その表現を用いた再帰プログラムの作成手法を提案する.まず,初心者の再帰プログラミングにおける誤りを整理し,振舞いの表現について検討した.さらにこの表現の部品化を行うことにより,雛形と部品を利用したプログラム作成手法を作成した.

以下では,2.2 において,再帰プログラミング教育について述べ,2.3 において手続き的な再帰プログラミングにおける難しさについて整理する.これを踏まえて,2.4 において,再帰の振舞い表現 U-behavior を提案する.さらに 2.5 において U-behavior を用いた再帰プログラムの設計手法 U-solution を提案する.さらに 2.6 おいて,U-behavior の評価実験について述べる.

2.2 再帰プログラミング教育

再帰プログラミングの教育では,宣言的なプログラミングが最終的な教育目標となる.しかしながら,C言語のような手続き型の言語において再帰プログラミングを行うには,手続き的なプログラミングが不可欠である.また,Prologのように宣言的なプログラミングを重視した言語においても,現在のところ,完全に宣言的なプログラミングのみで再帰プログラムを作成できるコンパイラやインタプリタは存在しないために,手続き的なプログラミングが不可欠となっている[Sterling 88].

再帰プログラムは,停止処理と再帰呼び出しから構成されるプログラムである.初心者を対象とした再帰プログラミング教育では,まず,この2つの処理の働きについて説明を行うことが不可欠である.本研究では,この2つの処理を1つずつ含む最小再帰プログラムを対象とした学習支援について検討する.再帰プログラムは,プログラムが数行であっても,その実行過程は非常に長く,複雑となる傾向がある.したがって,最小再帰プログラムは再帰プログラムの基本形であるとともに,初心者にとっては十分困難な学習課題である.また,再帰プログラミングを修得したといえるためには,再帰呼び出しを複数含む重再帰プログラムを理解・設計できるようにならなければならないが,その学習段階に進むためには,最小再帰の理解・設計ができるようになる必要がある.したがって,最小再帰プログラムを支援対象とすることは十分妥当であるといえる.

プログラミングを学習する上で,データ構造の扱い方を修得することは重要な課題である.しかしながら,再帰プログラミング技法を修得する上で,どのデータ構造を使用するかは本質ではない.本研究では,プログラミングの初心者においても理解が容易となるように,扱うデータ構造を,線形リスト(文字列)と数値データに限定した.また,Prolog言語においては,バックトラック処理を含まないような入出力引数の組のみを扱うこととした.

2.3 手続き的な再帰プログラミングの難しさ

手続き的なプログラミングにおける初心者の誤りには,(a)再帰と反復との違いを正しく理解していないために反復と混同してしまう,(b)再帰プログラム中の変数の扱い方,特にスタックを介したデータ処理について正しく理解ができていない，といった事例が報告されている[Wiedenbeck 89], [Roberts 93], [谷川 96]．このような誤りを引き起こす原因として，以下のような要因が挙げられる．

・一般に学習者が再帰概念に不慣れであることが大きな要因となっていると考えられる．すなわち日常概念に類似の概念が存在していないために，理解困難な概念となっている．

・再帰プログラムは，プログラムの字面上の順序とその振舞が大きく異なっているために，初心者が振舞いの理解において誤りやすいことが考えられる．このため，比較的理解し易い反復のように振舞いを捉える学習者が多い．

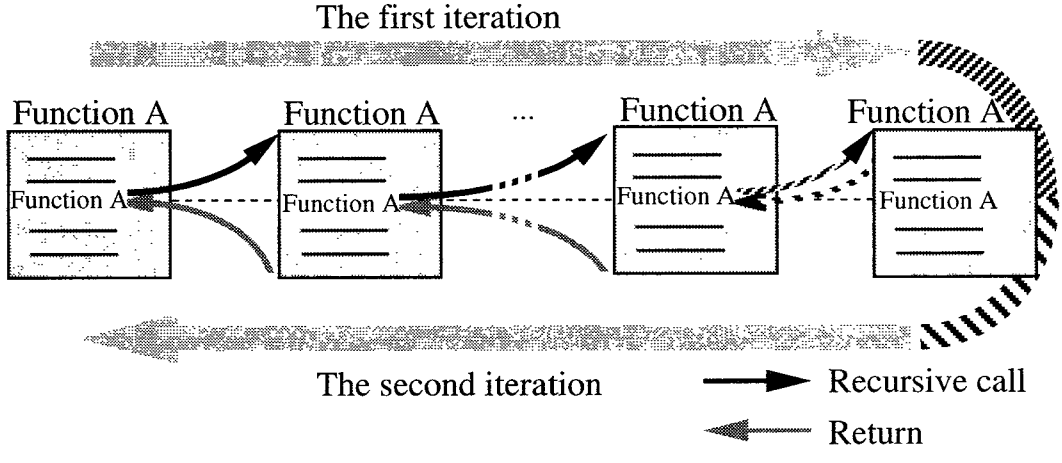


図 2.1 再帰関数の実行過程

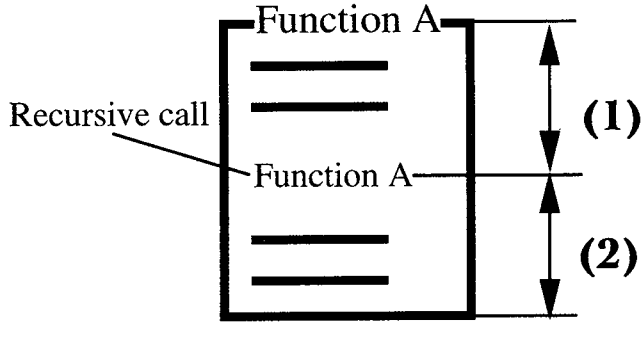
・再帰の手続き的な処理における変数のスタックへの退避処理が複雑であることが，学習者の理解を困難にしている一因と考えられる．スッタクというデータ構造自体に不慣れであることや，手続きを正確に辿ることができないために，変数の扱い方を誤っていると考えられる．

本研究では，以上のような初心者の誤り原因をもとに，手続き的な再帰プログラムの振舞いの表現を作成し，手続き的な再帰プログラミングの教育手法について検討する．

2.4 U-behaviorの作成

2.4.1 方針

前節の再帰プログラミングの難しさを踏まえて，初心者プログラマのための



(1) The first iteration

(2) The second iteration

図2.2 再帰関数

再帰プログラムの振舞いの表現に必要となる条件を以下のように設定した.

再帰と反復の違いを明確にすること.

再帰処理におけるデータの流れを初心者にとって分かりやすく表現すること.

このような観点から,再帰呼び出しを一つしか含まない再帰プログラムを対象に2つの反復に対応させた振舞い表現U-behaviorを提案した.

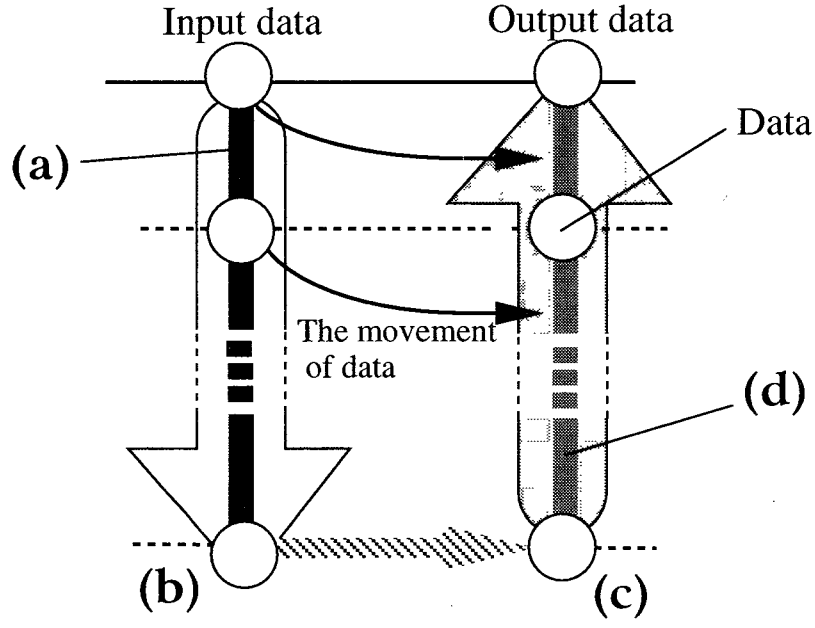


図2.3 再帰関数の実行過程を表すU字型表現

2.4.2 U字形の振舞い表現

図2.1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び出し(Recursive call)までの間の処理が行われる（図2.2(1)）．次に最初の再帰呼び出しが行われる．この時，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2つ目の関数Aにおいて実行される処理は1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び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，再帰呼び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．この判定は再帰呼び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，再帰呼び出しから関数の最後までの間の処理が行われる（図2.2(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．以降同様に再帰呼び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2.1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とよぶ．U-behaviorを図2.3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2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，再帰呼び出しが1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4つの要素に分類している．1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)」とよぶ．２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)」とよぶ．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は，「入力＝0」が再帰の停止条件である．３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)」とよぶ．４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)」とよぶ．

U-behavior では，これらの４つの要素を振舞い部品とよぶ．次の Prolog プログ

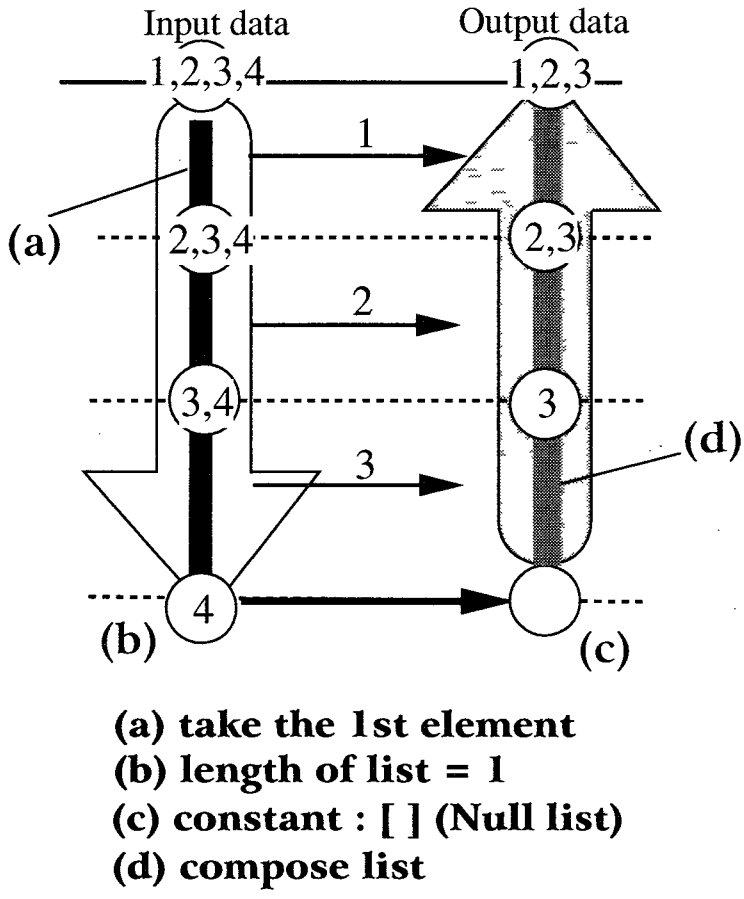


図 2.4　delete_last プログラムの U-behavior

ラムは,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2.4のようになる.図では,初心者にも分かりやすいように,データ構造を数字のリストとして表している.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1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1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.

2.5 U-solutionの作成

U-behaviorによる振舞い表現をもとに,再帰プログラムを作成する手法を設定した.これをU-solutionと呼ぶ.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,この雛形を利用して再帰プログラムを作成する.ここでは,特にこの雛形をU-templateとよぶ.このU-templateはU-behaviorにおける4つの構成要素を空欄とした振舞い表現である.この空欄に振舞い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振舞い部品を表2.1～2.3に示す．

プログラム仕様からU-behaviorを作成するには表（表2.1～2.3）の中から適当と思われる振舞い部品を選びU-templateの4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，振舞い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2.5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

```
template(In1,Out1):-T.C., Init. .
template(In1,Out1):-
        Pre. , template(RIn1,ROut1), Post. .
```

(a) Prolog (2引数)

```
template(In1, In2, Out1):- T.C. , Init. .
template(In1, In2, Out1):-
        Pre. , template(RIn1, RIn2, ROut1), Post.
```

(b) Prolog (3引数)

```
int template(In1, Out1)
{
        Type RIn1, Rout1;

        if ( T.C. ) {
                Init. ;
                return(1);
        }
        if ( Pre. ) {
                if (template(RIn1, ROut1)){
                        if ( Post. ) {
                                return(1);
                        }
                }
        }
        return(0);
}
```

(c) C (2引数)

```
int template(In1, In2, Out1)
{
        Type RIn1, RIn2, Rout1;

        if ( T.C. ) {
                Init. ;
                return(1);
        }
        if ( Pre. ) {
                if (template(RIn1, RIn2, ROut1)){
                        if ( Post. ) {
                                return(1);
                        }
                }
        }
        return(0);
}
```

(d) C (3引数)

***Pre* : Pre-recursion sub-procedure**
***T.C.* : Terminating condition**
***Init* : Return value initialization**
***Post* : Post-recursion sub-procedure**

図2.5 PrologとC言語のProgram-template

表2.1　再帰の前処理と再帰の後処理における振舞い部品とプログラム部品

部品名称	記号	Prolog	C
部品なし	C_0, E_0	——	——
複写	C_1, E_1	***V1 = V2***	***V1 = V2 ;***
先頭要素取り出し	C_2, E_2	***V1 = [Head|V2]***	***head_list (Head , V2);*** ***tail_list (V1 , V2);***
先頭に追加	C_3, E_3	***V2 = [$Head_m$|V1]***	***compose ($Head_m$,*** ***V1 , V2);***
Kを足す	C_4, E_4	***V1 =V2 + K***	***V1 =V2 + K ;***
入力**m**を足す	C_5, E_5	***V1 =V2 + $Input_m$***	***V1 =V2 + $Input_m$;***
Kを引く	C_6, E_6	***V1 =V2 - K***	***V1 =V2 - K ;***
入力**m**をかける	C_7, E_7	***V1 =V2 * $Input_m$***	***V1 =V2 * $Input_m$;***

V1, V2, Head, Inputは変数．添字のmは，第m引数を表す．Input は入力引数，Headは入力データの先頭要素を表す．

表2.2 再帰の停止条件における振舞い部品とプログラム部品

部品名称	記号	**Prolog**	**C**
部品なし	S_0	——	——
条件なし	S_1	——	——
長さ＝0	S_2	***Input* = []**	**null_list(*Input*)**
長さ＝1	S_3	***Input* = [_]**	**one_element_list(*Input*)**
数値***K***	S_4	***Input* = *K***	***Input* == *K***

Inputは変数．null_list(X)は，変数Xがnullリストかどうかを調べる関数．
one_element_list(X)は，変数Xが１要素からなるリストかどうかを調べる関数．

表2.3 出力の初期化における振舞い部品とプログラム部品

部品名称	記号	**Prolog**	**C**
部品なし	I_0	——	——
入力$_{\mathbf{m}}$	I_1	**Out = *Input*$_m$**	**Out = *Input*$_m$**
入力$_{\mathbf{m}}$の先頭要素	I_2	**Out= *Head*$_m$**	**Out = *Head*$_m$**
Constant : *A*	I_3	**Out = *A***	**Out = *A***

Out, Input, Head は変数．添字のmは第m引数を表す．
Inputは入力引数を，Headは先頭要素を表す．

振舞い部品には，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2.1～2.3）．このプログラム言語で記述された振舞い部品を特にプログラム部品と呼ぶ．各振舞い部品に対応するプログラム言語の記述は一とおりではないが，ここでは表2.1～2.3に示すように，初心者にとって分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．U-templateで選択した振舞い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に適切な振舞い部品を代入する．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つけば，再帰の前処理の振舞い部品が先頭要素を取り出す処理，再帰の後処理の振舞い部品が先頭に追加する処理であることが分かる．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2.4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2.4のU-behaviorの出力の初期化振舞い部品だけを変更すればいいことに気付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振舞い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2.1～2.3の振舞い部品群から試行錯誤して振舞い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．

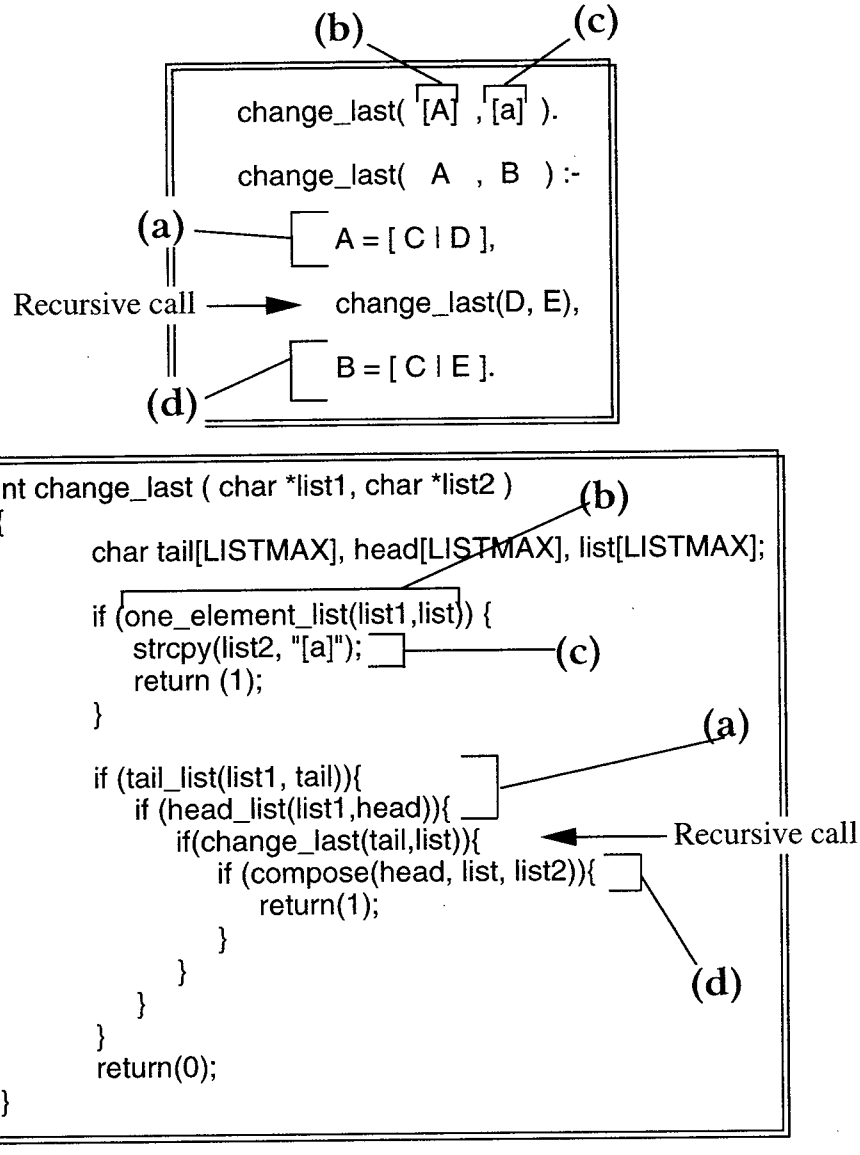
手順2：U-behaviorが得られたら，次に入力値として適当な値を設定して，正

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を辿って出力”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に戻って振舞い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振舞い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2.5に示すPrologのProgram-templateにプログラム部品を代入する．代入するプログラム部品は，手順1で選択した振舞い部品に対応するPrologの記述である（表2.1～2.3）．Program-templateのすべての空欄に振舞い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C言語のプログラム部品を代入する．このようにして完成したProlog，Cプログラムを図2.6に示す．

2.6 評価実験 1

U-behaviorによる振舞いの表現が，学習者にとって分かり易い表現かどうかを評価するための実験を行った．本実験では，市販されているプログラミング教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．さら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ミングの初心者である大学生と大学院生16名である．



(a) Pre-recursion sub-procedure
(b) Terminating condition
(c) Return value initialization
(d) Post-recursion sub-procedure

図2.6 change_last プログラム（Prolog, C）

2.6.1 実験方法

被験者を 8 名ずつの 2 つのグループに分け，一方のグループには，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，本研究において作成した U-solution の説明文を読んでもらった．この説明文で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

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 = 2 4

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]

図 2.7　評価実験で使用した試験問題の一部

表2.4 実験結果

			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

に与える影響を減らすために，それぞれのグループをさらに半分に分けて，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共に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．試験問題の一部を図2.7に示す．

2.6.2 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と対応するU-behaviorにおける振舞い部品に分解し，正解した振舞い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2.4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

表2.5 分散分析表

	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		

†$p < 0.1$ *$p < 0.05$

グループの方が平均点が上回った.

さら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2.5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の有効性を確認できた.

2.7 結言

本章では, 再帰プログラムの振舞い表現U-behaviorを提案した. U-behaviorの作成方針は, (1) 再帰と反復の違いを明確にすること, (2) 再帰プログラムにおけるデータの流れを分かりやすく表現すること, の2点である. さらに, U-behaviorを用いて再帰プログラムを作成する手法U-solutionについて述べた. U-

solutionは，手続き的な再帰プログラミング解法である．最後にU-behaviorの評価を行う実験について述べた．実験の結果，U-behaviorの有効性を確認できた．

第3章

問題演習機能の実現

3.1 緒言

本章では，本研究で設計・実装を行った，再帰プログラミングにおける理解・設計支援システムについて述べる．再帰プログラムの振舞い理解支援機能としてU-behaviorのアニメーション表示機能を，再帰プログラムの作成支援機能としてU-solutionの問題演習支援システムを設計・実装した．

U-solutionは，雛形と振舞い部品を利用したプログラミング手法である．雛形は，複数の再帰プログラムの比較により共通構造を抽出したものであり，振舞い部品は各プログラムと共通構造との差分を表している．従って，U-solutionを習得するためには，問題間の比較が重要と考えられる．本研究では，学習者に問題間の比較を促す問題演習支援機能について検討を行った．

以下ではまず，3.2において，U-behaviorの可視化機能について述べる．次に，3.3において，比較を促す問題演習支援システムにおいて必要となる問題間の異同の表現として，U-behaviorを利用する方法について述べる．さらに3.4において，U-behaviorで表現された問題間の異同をもとに，問題演習支援を行う手法に

ついて述べる．さらに3.5において，実際に実装した問題演習支援システムの評価実験について述べる．

3.2 U-behaviorの可視化

U-behaviorは，再帰プログラムの動的な振舞いを表現しているため，アニメーションを用いて表現することによって,学習者にとってより分かり易い表現となると考えられる．本研究では，学習者にとって親しみやすい概念への写像を利用して，振舞い部品をアイコン表示するU-behaviorの可視化機能を設計･実装した．以下では，まず振舞い部品のアイコン表示について述べ，さらにアニメーション表示について述べる．最後に，プログラムとの対応関係の表現について述べる．

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.このような学習者にとって理解困難な概念をより分かり易く説明するために，学習者にとって理解容易な概念への写像を利用する方法がある[He 95].

本研究では，学習者にとって理解が困難と思われる振舞い部品を，野球の道具や信号機，磁石などのアイコンを用いて表現する．図3.1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.この振舞い部品を「バットでボールを打つ」アイコンで表現する.また,この処理によって取り出された先頭要素を「ボール」アイコンで表す.

3.2.2 U-behaviorのアニメーション表示

U-behaviorにおける各振舞い部品にあらかじめアニメーション表示機能を用意することにより，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，すなわち再帰呼び出しを１つしか含まない再帰プログラムに限定される．

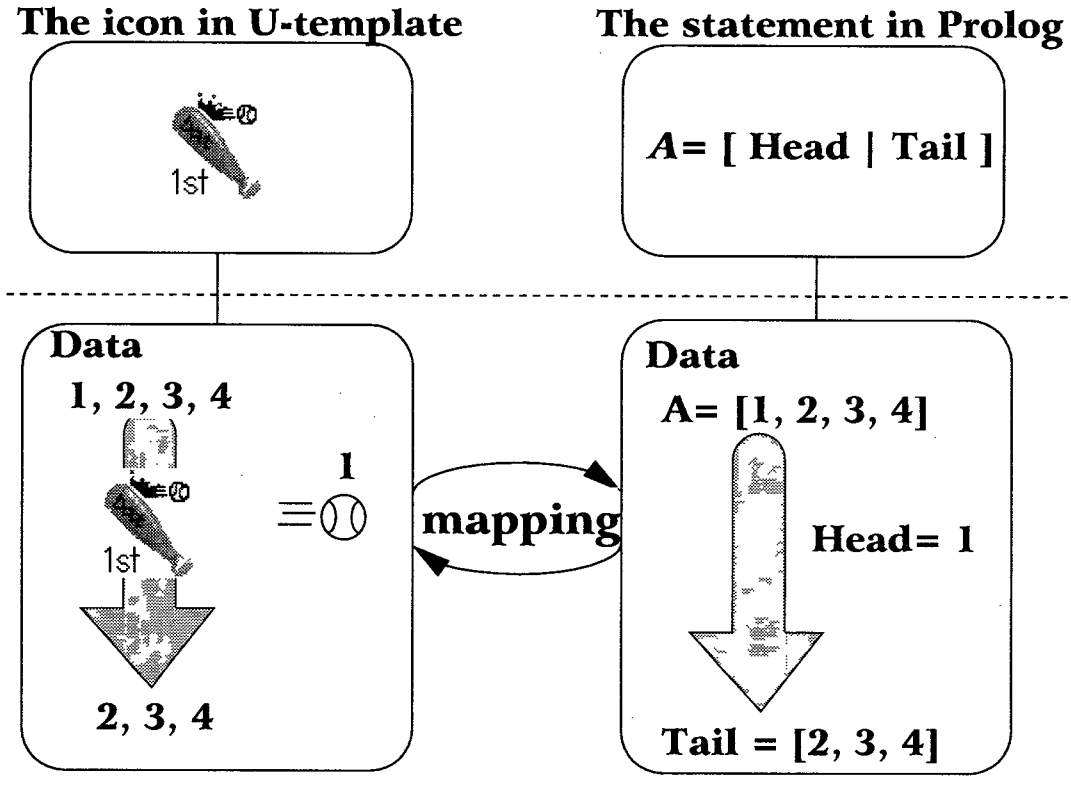


図3.1 野球を表すアイコンへの写像の例

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振舞い部品の内部表現としている．

[部品タイプ，振舞い部品名，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]

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振舞い部品の振舞いをアニメーション表示する関数である．

図3.4に，2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振舞い部品は，再帰の前処理では入力1（1つ目の入力データ），入力2（2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．

すべての振舞い部品を選択して，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」の振舞い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」振舞い部品を表す「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」振舞い部品を表す「グローブ」アイコンによって，前処理の「バット」アイコンから飛ばされたボール（先頭文字）を受け取る．このボールの移動は，再帰が実行される度に行われる，「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．

3.2.3 U-behaviorとプログラムの対応関係の表現

U-solutionの理解において，U-behaviorとプログラムの関係，すなわち振舞い部品とプログラムの記述についての理解支援が必要となる．本研究では，簡単な支援機能を作成している．図3.2は，U-behaviorのアニメーション表示を行う際の全体画面を示している．図のように，アニメーションを表示するウィンドウに加えて，Cプログラムが提供される．これらのウィンドウの内容は，相互に関係づけられており，学習者が振舞い部品のメニューから部品を選択すると，その部品に対応するプログラムの記述が表示される．また，アニメーション表示中では，実行中の処理に対応するプログラムの記述が点滅する．

3.3 問題間の類似点・相違点の表現

学習支援システムにおいて，問題間の比較を促す制御を実現するためには，問題間の異同についてのシステム内部表現が必要となる．ここでは，各問題の正解プログラムをU-behaviorで表現し，U-behavior上の異同を振舞い部品を用いて表現し，これをプログラム間の異同の表現とした．

図3.3(A)(B)に，Prologのappendとreverse_appの類似点・相違点を振舞い部品によって表現する例を示す．appendは，第1引数（入力1）のリストと第2引数（入力2）のリストを結合したリストを第3引数（出力1）とするプログラムである．reverse_appは，第1引数（入力1）のリスト要素の順序を反転して，第2引数（入力2）のリストと結合し，結合したリストを第3引数（出力1）とするプログラムである．この2つのプログラムの動作過程の類似点は，再帰の前処理においてリストの長さが0になるまで先頭要素を取り出し，再帰が停止したら入力2で出力を初期化する点である．また相違点は，再帰の前処理において入力1から取り出した先頭要素を追加するタイミングである．すなわちappendでは再帰

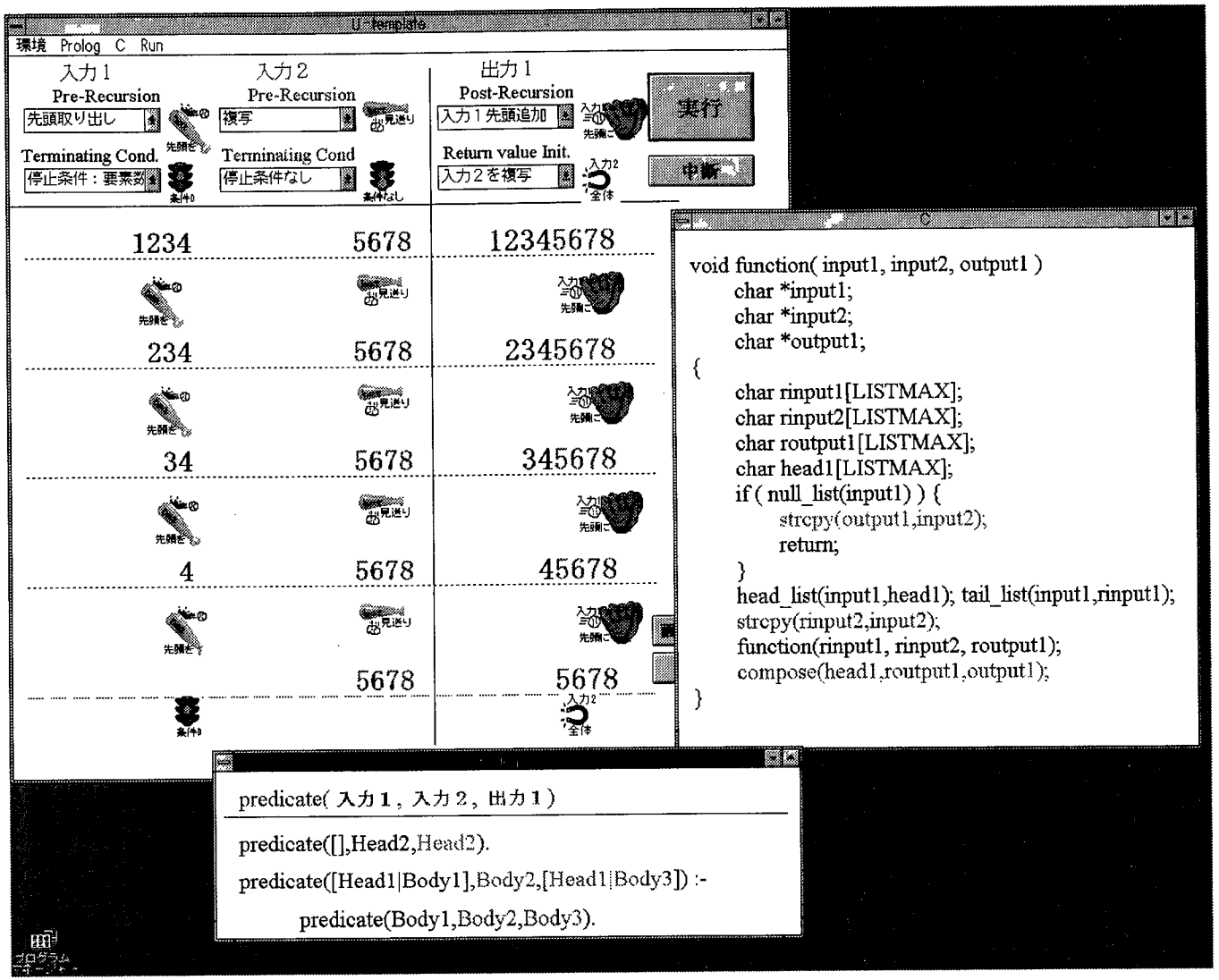


図 3.2　U-behavior の可視化機能の全体画面

の後処理で追加するのに対し，reverse_appでは再帰の前処理において追加する点である．このようなU-behavior上の異同を振舞い部品を用いて表現する（図3.3(C)）．append，reverse_appのU-behaviorにおける各部品タイプにおいて一致している振舞い部品（図3.3(C)実線部分）を類似点の表現とし，不一致となる振舞い

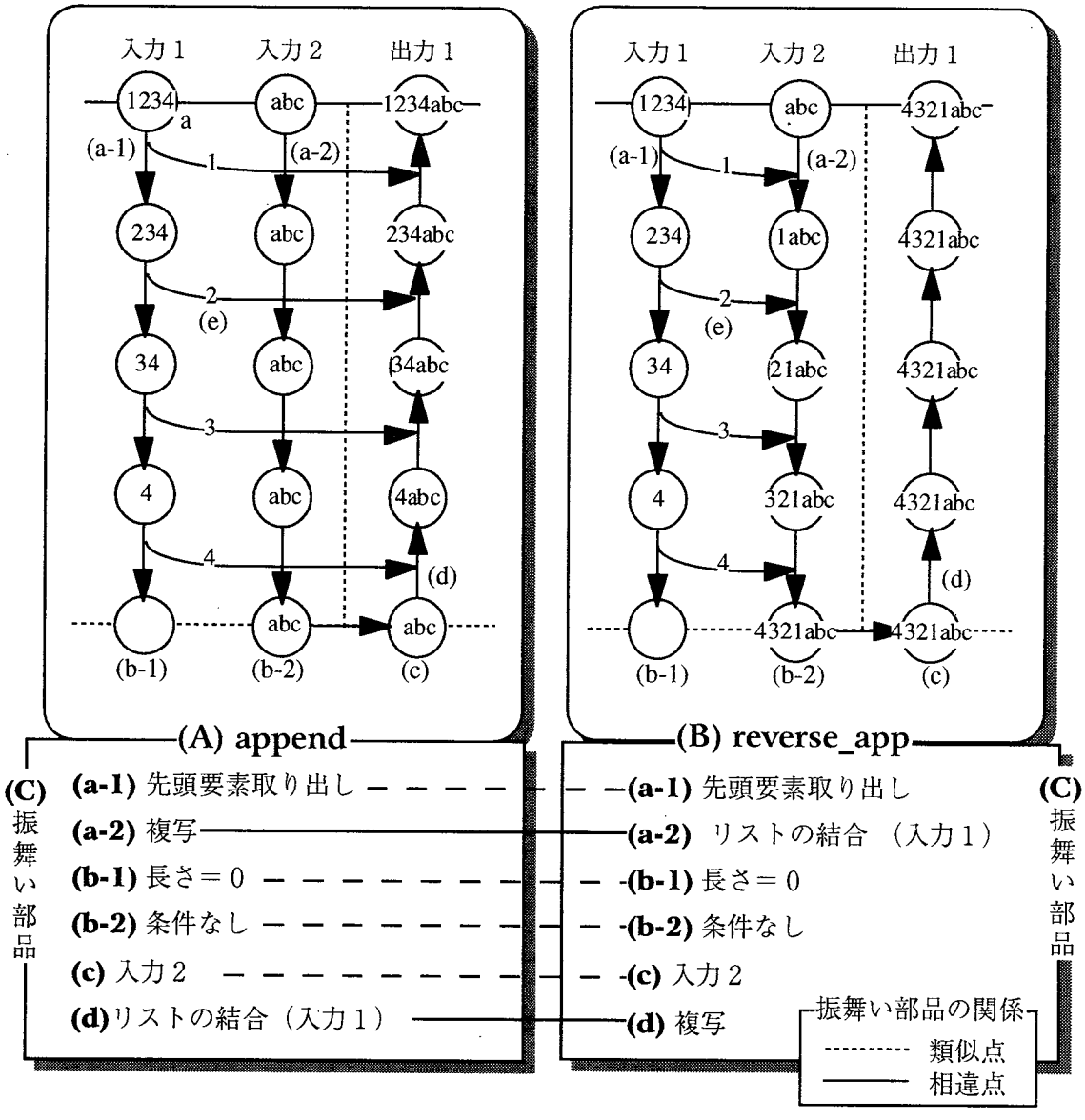


図3.3 appendとreverse_appのU-behaviorにおける類似点・相違点

部品（図3.3(C)破線部分）を相違点の表現とする．

3.4 比較を促す問題演習支援機能

U-behaviorの振舞い部品によって表現された問題の類似点・相違点の表現をもとに，問題間の比較を促す問題演習支援機能を作成した．作成した支援機能は，（1）問題間の差の説明，（2）問題の検索，（3）分類問題の作成機能である．以下にそれぞれの実現手法を述べる．

3.4.1 問題間の差の説明機能

問題間の比較を支援する最も基本となる機能が，問題間の差の説明機能であり，U-behaviorにおける異同を自然言語およびアニメーションによって説明する．説明文はプログラム間の類似点や相違点となる振舞い部品について自然言語で説明したものであり，プログラム仕様と対応づけて提示される．アニメーションは，それぞれのプログラムの振舞いをU-behavior表現の上でデータが処理される過程をアニメーション表現したものである．学習者にそれぞれのプログラムのアニメーションを提示して比較を促す．

まず，説明文の作成方法について述べる．ここでは，あらかじめ教師によって作成された，各問題のプログラム仕様の説明文と，各振舞い部品の働きに関する説明文を用いて，プログラムの説明文を作成する．ここでは，部分的に分割された文をテンプレートに代入することによって説明文を作成する．類似点に関する説明文の生成では，類似点を表す振舞い部品にあらかじめ設定されている説明文を，用意したテンプレートに代入する．同様に相違点に関する説明文の生成では，相違点を表す振舞い部品の説明文をテンプレートに代入する．

例えば，Prologのappend，reverse_appにおいて，説明文を生成する場合につい

て考える．図3.3(C)はこの２つのプログラムのU-behaviorとそこで使用される振舞い部品を示している.プログラムの振舞いについての説明文を生成するときは，振舞い部品に予め設定されている説明文を「再帰の前処理」,「再帰の停止条件」,「出力の初期化」,「再帰の後処理」の順に組み合わせることによって，プログラムの説明文を作成する．appendのU-behaviorの説明文は，次のようになる.「入力１は先頭要素の長さが０になるまで１つずつ取り出す.入力２は複写を繰り返す.再帰が停止したら，出力を入力２で初期化する．出力は先頭に入力１で取り出した要素を追加する.」下線部分が，各振舞い部品に設定されている説明文である．同様にreverse_appの振舞いの説明文は，「入力１は先頭要素の長さが０になるまで１つずつ取り出す.入力２は先頭に入力１で取り出した要素を追加する.再帰が停止したら，出力を入力２で初期化する．出力は複写を繰り返す.」となる.次に,この２つのプログラムの類似点と相違点についての説明文を作成する．２つのプログラムの間で,一致している振舞い部品を利用して類似点の説明文を，一致していない振舞い部品を用いて相違点の説明文を作成する．appendとreverse_appの間の類似点の説明文は次のようになる.「入力１の再帰の前処理において,先頭要素の長さが０になるまで１つずつ取り出す点が類似点である.出力の初期化は，出力を入力２で初期化する点が類似点である.」appendとreverse_appの間の相違点の説明文は次のようになる.「appendにおける相違点は，入力２の再帰の前処理において，複写を繰り返す点である．また，再帰の後処理は，出力は先頭に入力１で取り出した要素を追加する点である．reverse_appにおける相違点は，入力２の再帰の前処理において，先頭に入力１で取り出した要素を追加する点である．また，再帰の後処理は，出力で複写を繰り返す点である.」

さらに,それぞれのプログラムのプログラム仕様について説明した文章を提示する．各問題のプログラム仕様についての文章は，あらかじめ教師により与えられている．appendのプログラム仕様は「入力１と入力２のリストを結合して出力３とする」であり，reverse_appのプログラム仕様は「入力１の要素の順序を反転

して，入力2と結合して出力3とする」となる．このようなプログラム仕様についての説明文を，前述のU-behavior上の説明文と対応づけて学習者に提示することにより，プログラム仕様とU-behaviorの対応関係についての理解を促す．

次に，U-behaviorのアニメーションを利用した差の説明方法について述べる．U-behaviorはプログラムの動作過程を表す表現であるため，アニメーションを用いてデータの動きを表現することで，学習者にとって理解容易な表現となると考えられる．U-behaviorのアニメーション表示を利用したプログラム間の差の説明の具体例として，append，reverse_appのアニメーション画面を図3.4，図3.5に示す．図3.4に示すように，appendの再帰の前処理である入力リストから1要素取り出す様子は，入力データが下方向に動く過程で表現される．すなわち，振舞い部品「先頭要素取り出し」を表すバットのアイコンにより，入力データが2つに分岐して，入力データの先頭要素が右方向に，残りのデータが下方向に移動する．appendでは，右方向に移動した先頭要素が，再帰の後処理における振舞い部品「先頭に追加」を表すグローブのアイコンによって，出力の先頭に追加される．一方，reverse_appでは，右方向に移動している先頭要素が，もう一つの入力引数の再帰の前処理に位置する振舞い部品「先頭に追加」のグローブアイコンにより，入力データの先頭に追加される．このように振舞い部品「先頭に追加」を表すグローブアイコンの位置の違いによって，取り出されたデータが追加されるタイミングの違いをアニメーションで表し，さらに，このような動作上の違いが，出力結果における要素順序の違いに表れることを表現する．

このようなU-behaviorに基づく問題間の差の説明が有効であるかどうかは，プログラムを比較するための表現としてのU-behaviorの適切さを示すものである．この説明機能における有効性の評価については後述する．他の支援機能については，問題演習全体を対象とした学習支援システムの完成後に行う予定である．

3.4.2 問題の検索機能

ここでは，比較を促す問題を検索する機能を作成した．比較において説明する内容によって２つの検索条件を設定した．

(A-1)振舞い部品の説明

問題p_oとの比較を通して振舞い部品bの働きを説明することを目的に，問題p_o以外の問題p_sを学習者に提示する場合．この場合の問題p_sの検索条件を，「問題p_sは，説明する振舞い部品bを含み，かつ，問題p_sの振舞い部品が問題p_oの振舞い部品とできるだけ多く一致すること」とした．

(A-2)部品タイプの説明

問題p_oとの比較を通して部品タイプtの説明を行うことを目的に，問題p_o以外の問題p_sを学習者に提示する場合．この場合の問題p_sの検索条件を，「問題p_sは，部品タイプtにおいて問題p_oと振舞い部品が異なり，かつ，部品タイプt以外の部品タイプにおいて問題p_sの振舞い部品が問題p_oの振舞い部品とできるだけ多く一致すること」とした．

さらに，検索対象とする問題によって，本機能が使われる状況を以下の２通りに分類できる．

(B-1)例題の検索

学習者が過去に解いた問題を対象に問題検索する場合．この場合，検索された問題は，例題として学習者に提示する．

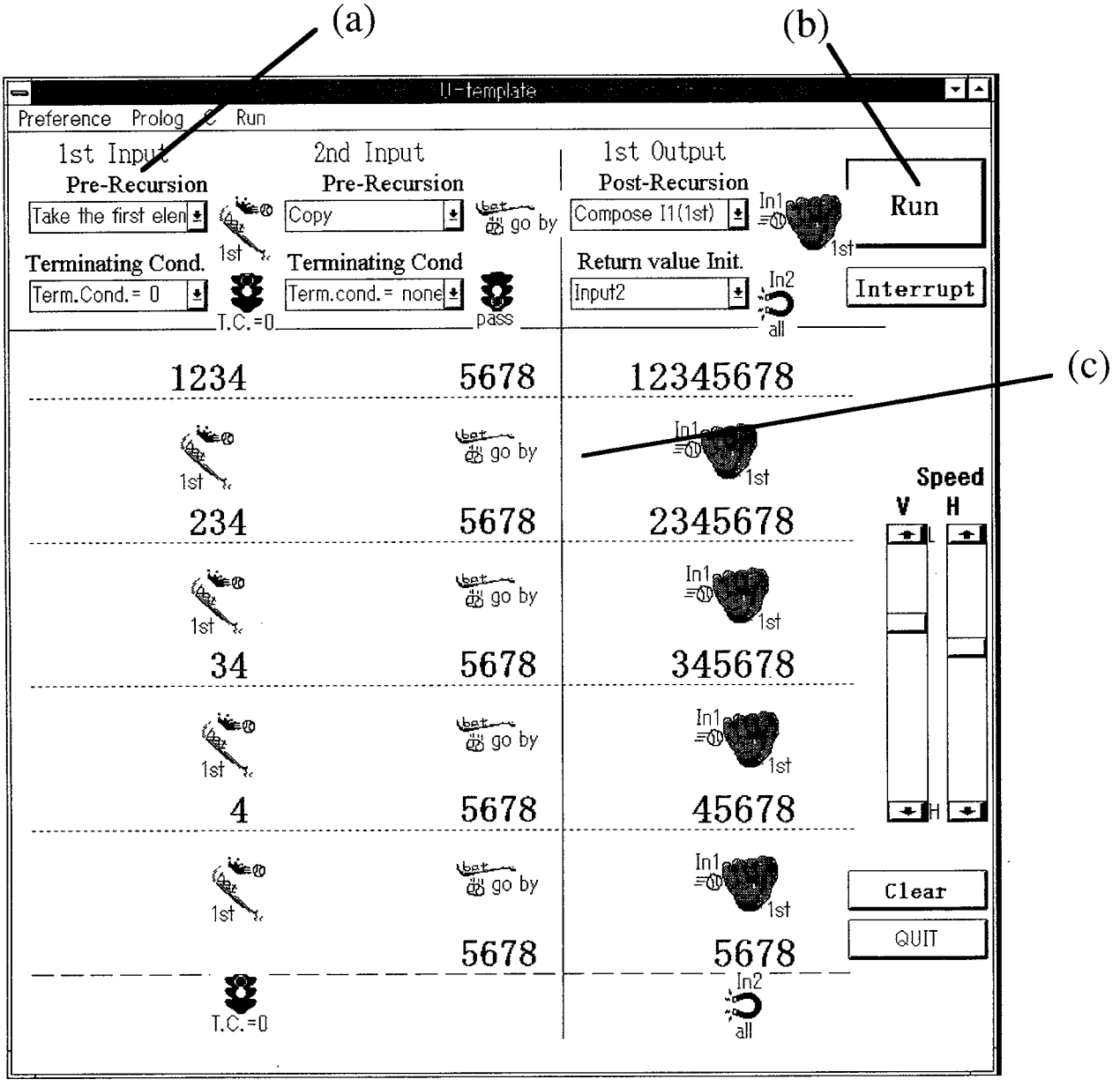


図3.4 appendプログラムのU-behavior

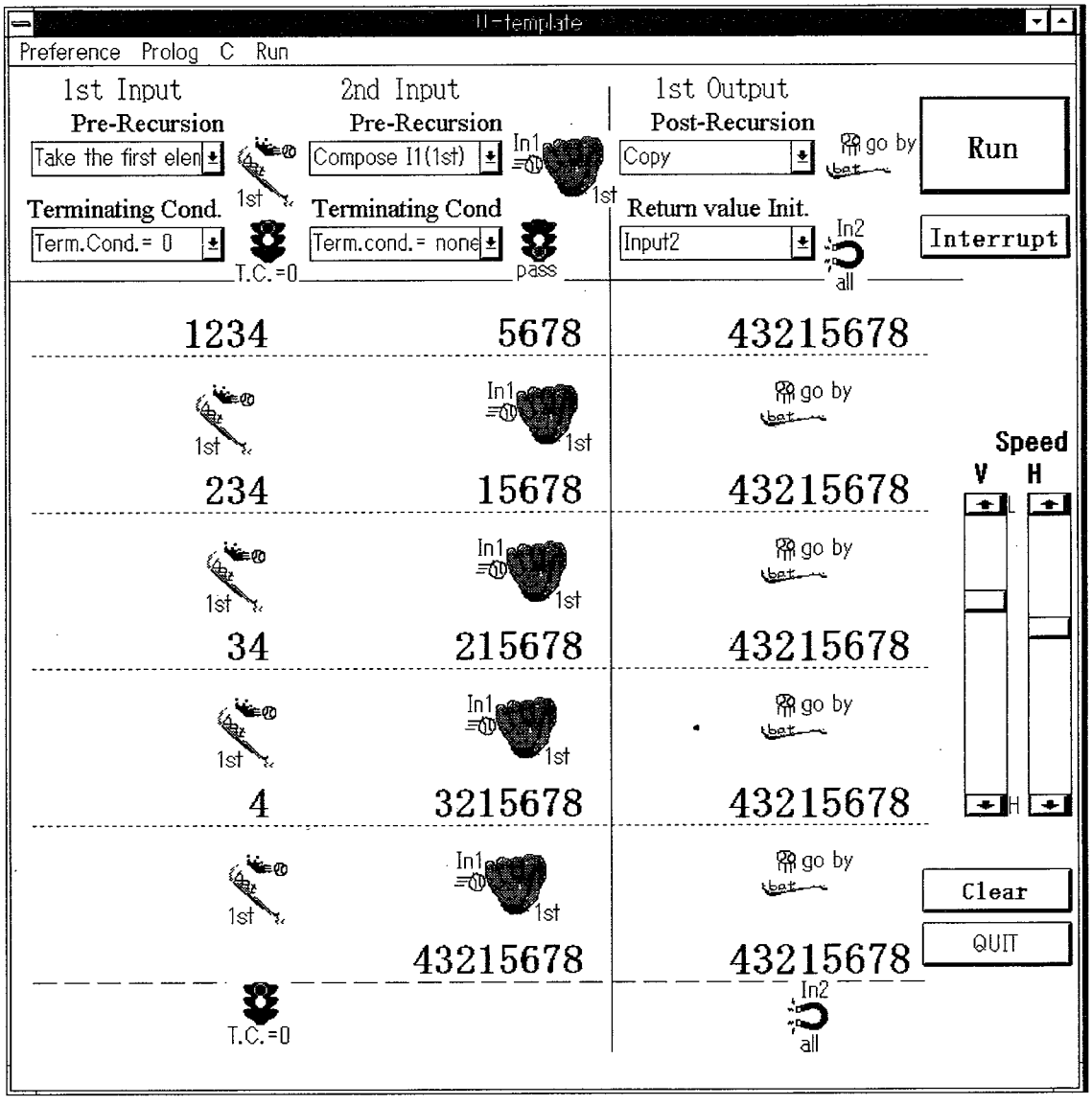


図 3.5　reverse_app プログラムの U-behavior

表3.1 振舞い部品の経験回数による評価値

	一致	不一致				
		経験回数 4回以上	3回	2回	1回	0回
評価値	1	0.8	0.6	0.4	0.2	0

(B-2)次の問題の検索

学習者がまだ解いていない問題を対象に問題検索する場合．この場合，検索された問題は，学習者が次に解くべき問題として提示する．

以上の検索条件をもとに，検索対象となる問題の中から，最もよく条件を満足する問題を全探索する．また，検索の結果，複数の問題が選ばれた場合には，競合を解消するための条件が必要となる．そこで，ここでは不一致となっている振舞い部品について，学習者がその振舞い部品を過去に経験した回数を考慮して，よく経験した振舞い部品が多く含まれる問題を選択する条件を設定した．その手法として経験頻度を表す評価値を利用した．評価値は，表3.1に示すように，問題に属する各振舞い部品について，一致していれば最高値である1を，不一致の場合は経験回数により1未満0以上の値とした．問題の評価値は，その問題に属する全ての振舞い部品の評価値の合計値とした．競合している問題について評価値を求め，その中で最高値となる問題を検索結果として提示する．

問題検索の具体例を示す．例として，問題reverse_appとの比較を通して，出力の初期化に属する振舞い部品I_1の説明を行うための問題を，過去に解いた問題から検索する場合について述べる．すなわち，本例は前述の(A-1)(B-1)の状況に相当

表3.2 例題検索の例

	問題系列	再帰の前処理 入力1	再帰の前処理 入力2	停止条件 入力1	停止条件 入力2	出力の初期化	再帰の後処理
	select	C_2	C_1	S_6	S_1	I_2	E_3
	last	C_2	C_0	S_3	S_0	I_1	E_1
	append	C_2	C_1	S_2	S_1	I_1	E_3
	append_del	C_2	C_1	S_3	S_1	I_1	E_3
	suffix	C_2	C_1	S_5	S_1	I_0	E_0
現在の問題	reverse_app	C_2	C_3	S_2	S_1	I_1	E_1

する．なおここでは，学習者が表3.2に示す問題を解いてきたものとする．まず，学習者が解いた問題の中から，振舞い部品I_1を含まない問題を除くと，last, append, append_del が候補として残る．次に，これらの候補の中から reverse_app の振舞い部品と最も一致する振舞い部品が多いプログラムを検索する．この中では，appendが最も一致する振舞い部品が多いので，このプログラムを例題とし提示する．

3.4.3 分類問題の作成機能

学習者にU-behaviorについて，より深く理解させるためには，用意された問題の色々な組み合わせにおいて，U-behaviorの類似点や相違点を考えさせることが重要と考えられる．ここでは，一とおり問題演習を終えた問題について，プログラムの振舞いの観点で分類させる，分類問題を作成する．分類問題を用いた問題演習では，U-behaviorの上で複数の問題をあらかじめ分類した「分類図」を学習者に提示し，ここに新たなプログラムを追加させる．従って分類問題を解くため

```
append(入力 , 入力 , 出力).
append([],A,A).
append([A¦B],C,[A¦D]):-append(B,C,D).
reverse_app(入力 , 入力 , 出力).
reverse_app([],A,A).
reverse_app([A¦B],C,D):-
    reverse_app(B,[A¦C],D).
copy1(入力 , 出力).
copy1([],[]).
copy1([A¦B],C):-copy1(B,C).
copy2(入力 , 出力).
copy2([A],[A]).
copy2([A¦B],C):-copy2(B,C).
only_a(入力 , 出力).
only_a([],a).
only_a([A¦B],C):-only_a(B,C).
is_list(入力).
is_list([]).
is_list([A¦B]):-is_list(B).
symmetry_app(入力 , 入力 , 出力).
symmetry_app([],A,A).
symmetry_app([A¦B],C,[A¦D]):-
    symmetry_app(B,[A¦C],D).
last(入力 , 出力).
last([A],A).
last([A¦B],C):-last(B,C).
```

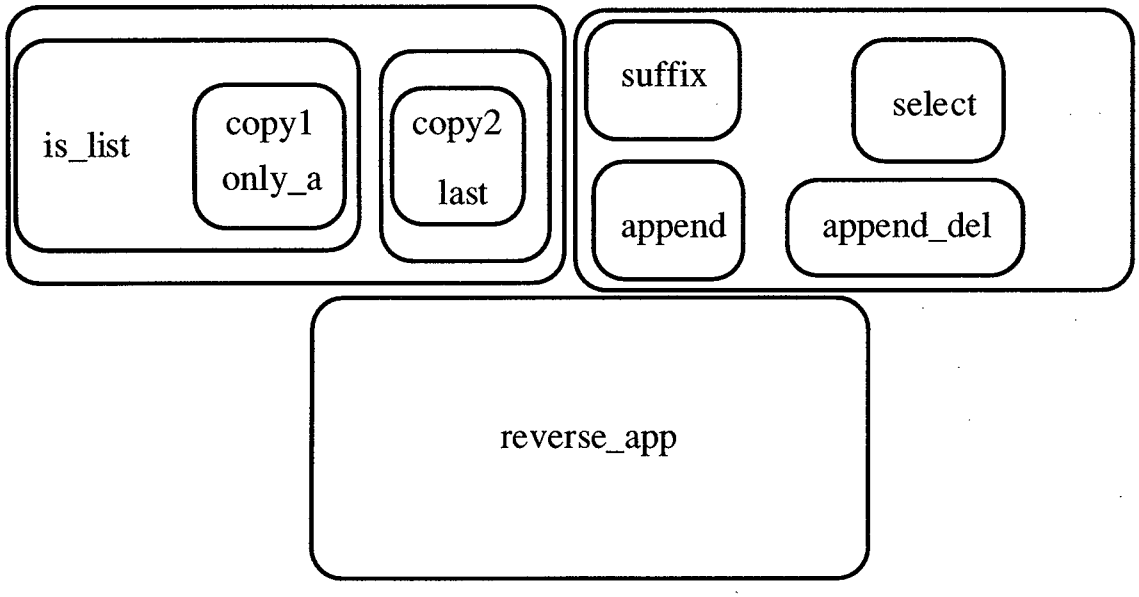


図3.6 プログラムの分類図の例

には，学習者は追加するプログラムの動作過程と，既に分類されているプログラムとの動作過程を比較し，適切な位置を検討する必要がある．このような比較作業を学習者に行わせることで，振舞い部品の働き，部品タイプの働きなど，問題解決知識の理解を深めることが期待できる．

分類問題の作成では，学習者が過去に経験した問題の中からランダムに問題を１つ取り除き，残った問題において，問題の分類図を作成する．分類図の作成では，まず，４種類の部品タイプについて分類する順序を決める．このとき，学習者がどのような順序においても正しく分類できるように，ランダムに順序を決定する．次に，この順序に従って，最初の部品タイプについて共通する振舞い部品を持つ問題を同じグループとする．さらに各グループにおいて，次の部品タイプで分類したグループを作成する．このようにして作成した分類図の具体例を図3.6に示す．この分類図は，１１個のプログラムを，「再帰の前処理」，「再帰の停止条件」，「出力の初期化」，「再帰の後処理」の順に分類したものである．分類問題では，１１個の中から任意の１つを取り除いておき，そのプログラムを学習者に代入させる．

3.5 評価実験2

U-behaviorにもとづく問題間の差の説明の有用性を評価するための予備的な実験を行った．実験では，（１）本システムにより問題間の差の説明を行い（これを教育フェーズと呼ぶ），（２）その直後にプログラミングテストを行った（これをテストフェーズと呼ぶ）．テストフェーズにおけるテストの点数の平均点により評価を行う．被験者は，C言語についての文法の知識は既に持ってるが，再帰プログラムの作成ができない初心者プログラマ（大学生と大学院生）１２名である．システムを用いた再帰プログラミング教育を約４５分，その直後にプログラ

ミングテストを３０分行った．実験において配布した資料を付録Ａに示す．

教育フェーズでは，まず，U-behaviorと，U-solutionについて説明を行い，次に，プログラム仕様と入出力例で構成される４つの問題を提示して，U-behaviorとCプログラムを作成する問題演習を行った．また，演習では，本稿で作成した比較を促す支援機能を実装したプロトタイプシステムを利用した．システムは，（１）問題間の差の説明（問２，４のみ），（２）振舞い部品の説明文の提示，（３）正解・不正解の提示，（４）U-behaviorのアニメーション表示を行った．さらに，U-behaviorを作成する際には，図3.4に示すU-behaviorの作成ツールを利用してもらい，作成ツールが出力するアニメーションで，意図したU-behaviorが得られているかどうかを確認してもらった．また，問２，問４では，U-behaviorを正解した後，Ｃ言語のプログラムを紙に書いてもらった．書き終えた後，正解を提示し，誤りがあれば訂正してもらった．

テストフェーズでは，３問の問題を解かせるテストを行った．これらの問題は，教育フェーズの４問の問題で利用した振舞い部品を組み合わせることで正解できるような問題を設定した．それぞれの問題において，U-behaviorとCプログラムをそれぞれ５分間で作成してもらった．U-behaviorの作成には，教育フェーズと同様に作成ツールを利用してもらった．

テストフェーズにおける問題の採点では，U-behaviorの４つの部品タイプと再帰部分の計５カ所における正解個数を点数とした．従って，U-behavior，Cプログラムそれぞれ満点は５点となる．採点の結果，U-behaviorの正解率が９２％，Cプログラムの正解率が８７％となり，いずれも高い得点となった．実験前には再帰プログラムが作成できなかった被験者が，約４５分間の教育フェーズを通して高い正解率をあげていることは，本教育手法が有望であることを示唆している．

本実験では，さらに，実験終了後に被験者に対して，問題間の説明がプログラミングに効果があったかどうかを問うアンケートを実施した．その結果，１２人中７人の被験者が問題間の差の説明がプログラミングに有効であったと述べ，４

人はどちらとも言えないとし，1人は効果がないと答えた．有効であると判断した1人と，どちらともいえないと判断した2人の被験者から，実験に使用したシステムにおけるウィンドウの配置等のインタフェースに問題があるとの指摘があった．この指摘を行った被験者はいずれも，この問題は，システム上の問題であり，問題間の差に関する情報はプログラミングに有益であると述べた．これらの事実は，問題間の差の説明がプログラミングに有望であることを示唆している．

3.6 結言

本章では，再帰プログラムの設計支援として，問題間の比較を促す問題演習支援機能を設計した．問題演習機能を制御する問題表現としてU-behaviorを利用した．また，問題演習における問題の比較を促す機能は，（1）問題間の差の説明機能，（2）問題の検索機能，（3）分類問題の作成機能である．支援機能を実装した学習支援システムを評価するために行った実験の結果，本システムが有望であることを確認できた．

第 4 章

結論

本章においては，本研究で得られた成果を総括し，今後に残された課題について検討する．

第 2 章では，再帰プログラミングの動作過程を表す振舞いの表現方法について述べた．初心者プログラマに見られる再帰プログラミングの誤りをもとにU-behaviorと呼ぶ，再帰を反復に対応させた振舞い表現を提案した．さらに，U-behaviorを利用した再帰プログラミング解法について述べた．最後にU-behaviorの評価実験について述べた．実験の結果，U-behaviorの有効性を示す結果を得た．

第 3 章で，学習支援システムにおける比較を促す問題演習機能の設計方法について述べた．学習支援システムの設計において必要となる問題表現にU-behaviorを利用した．比較を促す問題演習機能として，問題間の差の説明機能，問題の検索機能，分類問題の作成機能を設定した．最後に，これらの機能を実装したシステムの評価を行うために行った実験について述べた．実験の結果，本システムが有望であることを確認できた．

本研究では，再帰プログラムの振舞い表現を用いて，手続き的な再帰プログラミングの教育支援について検討し，単純な構造の再帰プログラムにおいてその有

効性を確認した.本教育手法の後に続く宣言的なプログラミングを対象とした教育支援手法については，今後に残された課題である.

もう１つの課題として,問題演習全体を対象にした支援機能の実現が挙げられる.現在の問題演習支援機能は,問題演習の中での部分的な範囲に対する支援を行っている.学習者により有効な支援を行うためには,長期的な教育方針を考慮するなど，問題演習全体を対象にした支援が必要になると考えられる.

謝辞

本研究の全般にわたり終始懇切なる御指導を賜りました大阪大学産業科学研究所 豊田順一教授に心から深謝いたします.

本研究に関して,貴重な御教示を頂きました情報工学科 藤井護教授,大阪大学産業科学研究所 溝口理一郎教授,北橋忠宏教授に心から感謝いたします.

大学院後期課程において御指導を賜りました情報工学科 柏原敏伸教授,今井正治教授,菊野亨教授,西川清史教授,谷口健一教授,都倉信樹教授,萩原兼一教授,井上克郎教授,首藤勝教授,橋本昭洋教授,宮原秀夫教授ならびに医学部 田村進一教授に厚く御礼申し上げます.

本研究の遂行に際し,御教授を頂きました関西大学総合情報学部 江澤義典教授に厚く感謝申し上げます.

本研究の全般にわたり,常に有益なる御討論,御助言を頂いた大阪大学産業科学研究所 平嶋宗講師,柏原昭博助手,ならびに池田満助手に感謝いたします.

さらには,本研究を行うにあたり種々の御援助,御協力を頂いた大阪大学産業科学研究所豊田研究室の諸氏に感謝いたします.また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に感謝いたします.そして,様々な面で御世話になりました向井佐江子嬢に感謝いたします.

最後に,終始あたかかく見守り励ましてくれた両親に感謝申し上げます.

参考文献

[Bowles 93] A.Bowles, P.Brna : "Programming Plans and Programming Techniques", Proc. of AI-ED 93, pp.378-385 (1993).

[Burns 91] Hugh Burns, James W. Parlett, Carol Luckhardt Redfield : "Intelligent Tutoring Systems, Evolutions in Design", Lawrence Erlbaum Associates, Publishers, (1991).

[Goodyear 91] Peter Goodyear : "Teaching Knowledge and Intelligent Tutoring", Ablex Publishing Corporation, (1991).

[He 95] Yu 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).

[Hirashima 94] Hirashima, T., Niitsu, T., Hirose, K., et al. : "An Indexing Framework for Adaptive Arrangement of Mechanics Problems for ITS", IEICE TRANS. INF & SYST., VOL.E77-D, NO.1. JANUARY 1994, pp19-26.

[Itoh 94] Kohji ITOH, Makoto ITAMI, Kazuo FUKUKAWA, Jun MURAMATSU and Yoshitaka ENOMOTO : "A Workbench Sys-

tem for Novice Prolog Programmers: Visually-Structured Interactive Tracer and Prototype-Based Programming Support", IEICE TRANS. INF. & SYST., VOL.E77-D, NO.1, JANUARY 1994, pp57-67.

[Johnson 85] Johnson, W. E. and Soloway, E. : "PROUST: Knowledge-Based Program Understanding", IEEE Trans. on Soft. Eng., Vol.SE-11, No.3, pp.11-19, (1986).

[海尻 95] 海尻賢二：”ゴール／プランに基づく初心者プログラムの認識システム”，電子情報通信学会論文誌，D-II Vol.J78-D-II No.2,pp.321-332 (1995).

[Mandl 88] Heinz Mandl, Alan Lesgold : "Learning Issues for Intelligent Tutoring Systems", Springer-Verlag, (1988).

[Matsuda 95] Noriyuki Matsuda, Akihiro Kashihara, Tsukasa Hirashima and Jun'ichi 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).

[松田 97a] 松田憲幸, 柏原昭博, 平嶋宗, 豊田順一："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", 電子情報通信学会論文誌, D-II（採録決定）

[松田 97b] 松田憲幸,柏原昭博,平嶋宗, 豊田順一："再帰プログラミングを対象とした問題の比較に基づく問題演習機能の実現", 教育システム情報学会, (条件付き採録)

[Polson 88] Martha C. Polson, J. Jeffrey Richardson : "Intelligent Tutoring System", Lawrence Erlbaum Associates, Publishers, (1988).

[Polya 54] Polya, G. : "いかにして問題を解くか", 丸善, (1954).

[Roberts 93] Eric S. Roberts 著, 有川節夫監 訳, 篠原武 訳 : "再帰的思考法", オーム社, (1993).

[Sleeman 82] D.Sleeman, J. S. Brown : "Intelligent Tutoring Systems", Academic Press, INC., (1982).

[Soloway 84] Soloway, E. and Ehrich, K. : "Empilical Studies of Programming Knowledge", IEEE Trans. on Soft. Eng., Vol. SE-10, No.5, pp.595-609, (1984).

[Sterling 88] Leon Sterling, Ehud Shapiro 著, 松田利夫 訳 : "Prologの技芸", 共立出版, (1988).

[谷川 96] 谷川真樹, 市川伸一 : "プラグマティックな教示による再帰概念の理解", 認知科学, Vol.3 No.2, pp.83-95., (June 1996)

[Timothy 91] Timothy S, GEGG-HARRISON : "Learning Prolog in a schema-based environment", Instructional Science, Vol.20, pp.173-192 (1991).

[Timothy 92] Timothy, S., GEGG-HARRISON : "Adapting Instruction to the Student's Capabilities", JI. of Artificial Intelligence in Education, pp.169-181(1992).

[上野 87] 上野晴樹 : "知的プログラミング環境 -プログラム理解を中心に-", 情報処理学会学会誌, Vol.28, No.10, pp1280-1296, (1987).

[Ueno 94] Ueno, H. : "Integrated Intelligent Programming Environment for Learning Programming", IEICE TRANS. INF. & SYST., VOL.E77-D, NO.1 JANUARY 1994, pp.68-79.

[Wiedenbeck 89] Susan Wiedenbeck : "Learning iteration and recursion from examples", Int. J. Man-Machine Studies (1989) 30, 1-22

[Wenger 87] Etienne Wenger : "Artificial Intelligence and Tutoring Systems", Morgan Kaufmann Publishers, INC., (1987).

付録A

評価実験2における配布テキスト

実験にご協力いただきありがとうございます.

氏名

目次

6ページ以降は画面の指示でページ番号が指示されてから，開いて下さい.

1ページ

■リスト

「リスト」と呼ばれるデータ構造について、説明します。

リストは、任意個の要素を順番に並べたデータです。

例)
リスト [a, b, c] は、a, b, cの3つの英文字を要素とするリストです。
[] は、要素を1つも持たないリストで，特別に空リストと呼びます.

英文字、数字、ひらがなは、リストの要素です。

■リスト要素の操作方法について

・リストの要素を操作するには，次の２種類の操作を使用します．

（１）リストの分解

（２）リストの連結

（１）リストの分解

リスト[1, 2, 3, 4]を　分解すると

先頭要素　1　と
リスト　　[2, 3, 4]

に分けることができる．

すなわち，リストの先頭要素と残りのリストに分けることができる．

（２）リストの連結

要素　1　とリスト[2, 3, 4]を連結すると，

リスト　[1, 2, 3, 4]　になる．

リスト[1]　と　リスト[2, 3, 4]を連結すると，

リスト　[[1], 2, 3, 4]になる．

上の２つの例の違いに　注意して下さい．

このように，リストデータの操作は，先頭要素から１つずつ操作します．

リストの先頭以外の要素に対して，操作することはできません．

3ページ

■C言語におけるリストの操作

リストの分解，連結を行う関数を用意します．

以降のプログラミングでは，以下の4つの関数を利用できるものとします．

・リストを表現する変数の宣言

リスト List1, List2; ←―― リスト型の変数List1, List2の宣言

・リストの連結を行う関数

リスト List1, List2, List3;

List3 = 連結(List1, List2); ←―― List1とList2を連結したリストList3を返す関数

・リストの分解を行う関数

リスト List1, List2, List3;

List2 = 先頭要素(List1); ←―― List1の先頭要素List3を返す関数

List3 = 残り(List1); ←―― List1の先頭要素以外のリストList3を返す関数

・リストの長さを図る関数

リスト List1, List2, List3;

長さ0(List1) ←―― List1の要素が0要素なら真，それ以外のとき偽

長さ1 (List1) ←―― List1の要素が1要素なら真，それ以外のとき偽

■再帰について

ある言葉を定義するときに、定義の中にその言葉自身が用いられているような定義を、再帰的定義と呼びます。

再帰的な定義の例として、「整数の和」の定義について説明します。

「整数の和」とは、1から、ある数nまでの全ての数値を足し合わせることとします。

例）

5の整数の和は、1+2+3+4+5　＝　15　です。
8の整数の和は、1+2+3+4+5+6+7+8　＝　36　です。
1の整数の和は、1です。

5の整数の和は、1+2+3+4+5
4の整数の和は、1+2+3+4

上の2つの例をよく見ると、5の整数の和の1から4までの和は4の整数の和と同じです。すなわち　5の整数の和は、（4の整数の和）+5　であると言えます。

このような考え方を元にして、以下に示すような再帰的な整数の和の定義が行えます。

整数の和の再帰的定義

n＞1のとき、nの整数の和は、（n-1）の整数の和にnを足した値である。
1の整数の和は1である。

このように、ある概念の定義の中に、自分自身が含まれているような定義を、再帰的定義と呼びます。

5ページ

■Cプログラムの書き方

・出題された問題に対する解答 (C program) を，次ページ以降の解答用紙に，書いて下さい．

・厳密にC言語の文法通りでなくても構いません．
行末のセミコロンはつけなくても結構です．

・C言語の文法において，分からないところは，日本語でも結構です．

・制限時間は1問　4分です．

・解答には，main関数は不要です．再帰関数のみ書いて下さい．

・以下に示す関数と標準的な関数以外の，自作の関数は使用しないで下さい．

解答におけるリスト操作の表現方法

・リストを表現する変数の宣言

リスト　List1, List2;　◀―― リスト型の変数List1, List2の宣言

・リストの連結を行う関数

リスト　List1, List2, List3;

List3 = 連結(List1, List2);　◀―― List1とList2を連結したリストList3を返す関数

・リストの分解を行う関数

リスト　List1, List2, List3;

List2 = 先頭要素(List1);　◀―― List1の先頭要素List3を返す関数

List3 = 残り(List1);　◀―― List1の先頭要素以外のリストList3を返す関数

・リストの長さを図る関数

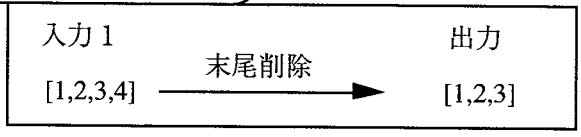
リスト　List1, List2, List3;

長さ0(List1)　◀―― List1の要素が0要素なら真，それ以外のとき偽

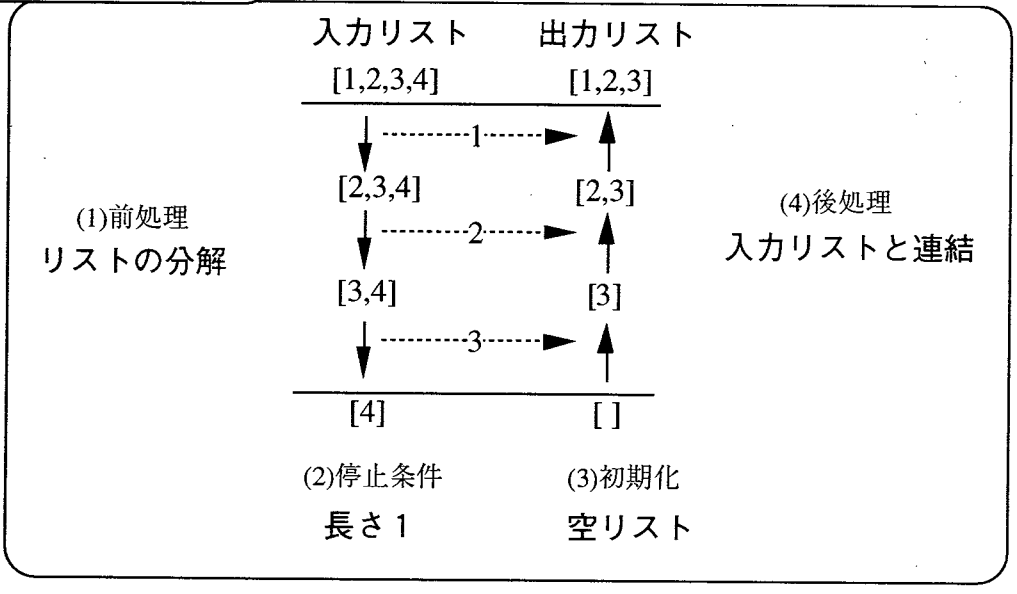
長さ1 (List1)　◀―― List1の要素が1要素なら真，それ以外のとき偽

■リストの末尾要素の削除を行うCの再帰プロプログラムの振舞い

末尾削除プログラムの入出力例



末尾削除プログラム



（1）前処理　入力リストを分解して先頭要素を1つずつ取り出す.

（2）停止条件　リストの長さが1になるまで，前処理を繰り返す.

（3）初期化　出力リストを空リストで初期化する.

（4）後処理　入力リストの先頭要素と出力リストを連結する.

7ページ

■リストの操作を行うアイコン

◎前処理，後処理

リストの分解．リストを先頭要素と残りのリストに分割する．
ボールが先頭要素となって，右に飛んでいき，残りのリストは
下に流れる．

リストの連結．要素とリストを連結する．
左が飛んできた要素を，先頭に連結して出力する．

リストに対して，手を加えない．
そのまま複写する．

◎停止条件

再帰が停止するための条件
リストの長さが0，長さが1などがある．

◎初期化

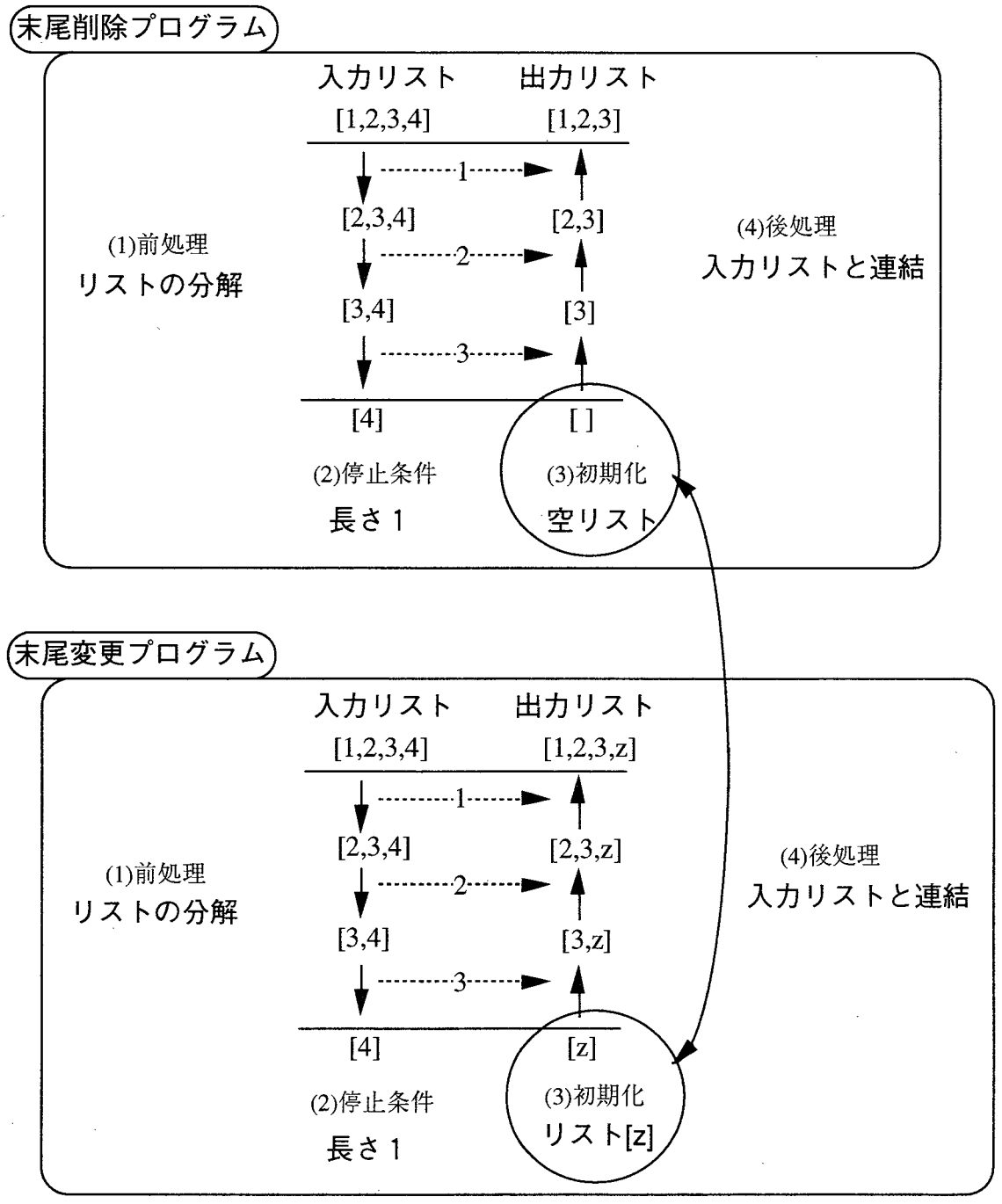
出力に入力リストの値を代入する．

出力にある値を代入する．
文字や空リストなどで出力を初期化する．

8ページ

■入力リストの末尾要素を要素"z"に変更する再帰プログラムの振舞い

前の問題「末尾削除」と「末尾変更」との違いを下図に示す.



9ページ

■リストの末尾削除を行うCの再帰プログラム

末尾削除プログラム

```
リスト　末尾削除( list1 )

{

        変数はリスト型;

        if ( 長さ 1 ( list1 ) ) {          ← 停止条件
                return ( [ ] );            ← 初期化
        }

        head = 先頭要素 ( list1 );         ← 前処理
        tail = 残り ( list1 );

        list2 = 末尾削除 ( tail );         ← 再帰呼び出し

        list3 = 連結 ( head, list2 );      ← 後処理
        return ( list3 );
}
```

■リストの末尾置換を行うCの再帰プログラム

上のプログラムを参考にして，末尾置換を行うCの再帰プログラムを作成して下さい．

１０ページ

■リストの末尾削除を行うＣの再帰プログラム

末尾削除プログラム

```
リスト　末尾削除( list1 )

{

        変数はリスト型;

        if ( 長さ１( list1 ) ) {          ← 停止条件
                return ( [ ] );           ← 初期化
        }

        head = 先頭要素 ( list1 );        ← 前処理
        tail = 残り ( list1 );

        list2 = 末尾削除 ( tail );        ← 再帰呼び出し

        list3 = 連結 ( head, list2 );     ← 後処理
        return ( list3 );
}
```

■リストの末尾置換を行うＣの再帰プログラム

初期化の違い

末尾置換プログラム

```
リスト　末尾置換( list1 )

{

        変数はリスト型;

        if ( 長さ１( list1 ) ) {          ← 停止条件
                return ( [z] );           ← 初期化
        }

        head = 先頭要素 ( list1 );        ← 前処理
        tail = 残り ( list1 );

        list2 = 末尾置換 ( tail );        ← 再帰呼び出し

        list3 = 連結 ( head, list2 );     ← 後処理
        return ( list3 );
}
```

１１ページ

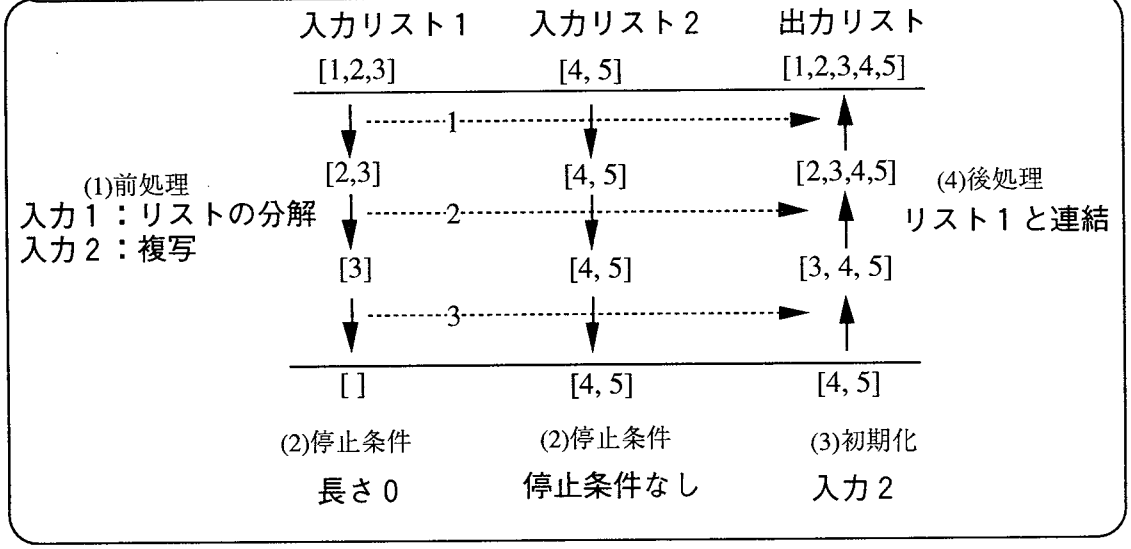
■２つの入力リストの結合を行うＣの再帰プロプログラムの振舞い

結合プログラムの入出力例

入力１	入力２		出力
[1,2,3]	[4,5]	結合 →	[1,2,3,4,5]

この問題では，２つの入力があるので，前処理と停止条件が下図のように２つあります．

結合プログラムの振舞い



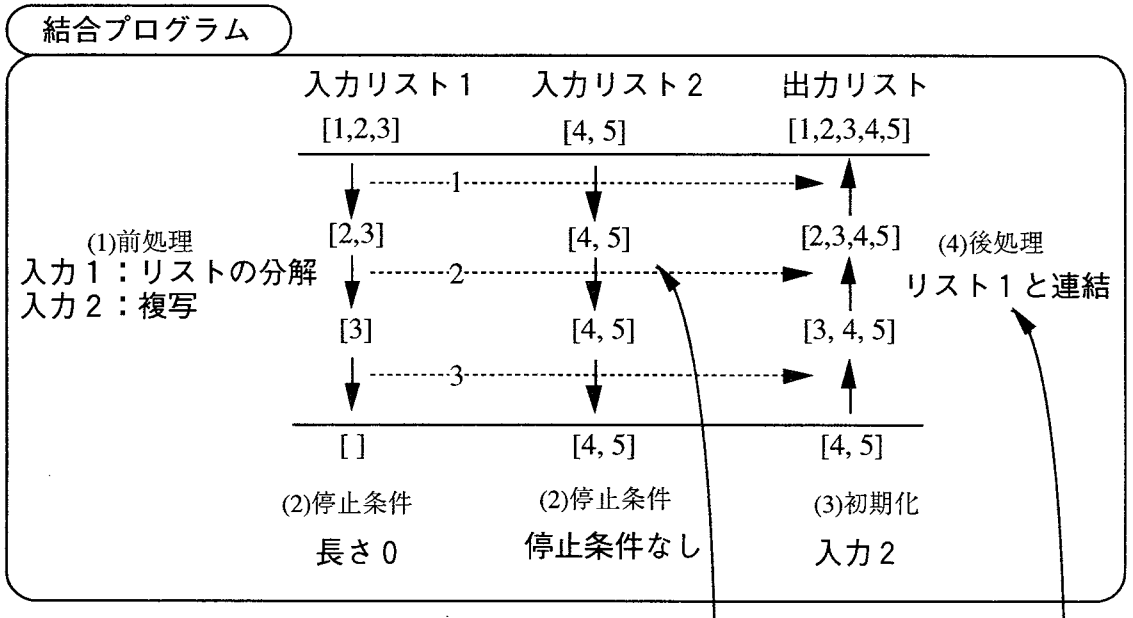
（１）前処理　入力１：入力リスト１を分解して先頭要素を１つずつ取り出す．
入力２：入力リスト２に対して手を加えず，複写する．

（２）停止条件　入力１：リスト１の長さが０になるまで，前処理を繰り返す．
入力２：リスト２を停止条件とはしない．

（３）初期化　入力リスト２で出力を初期化する．

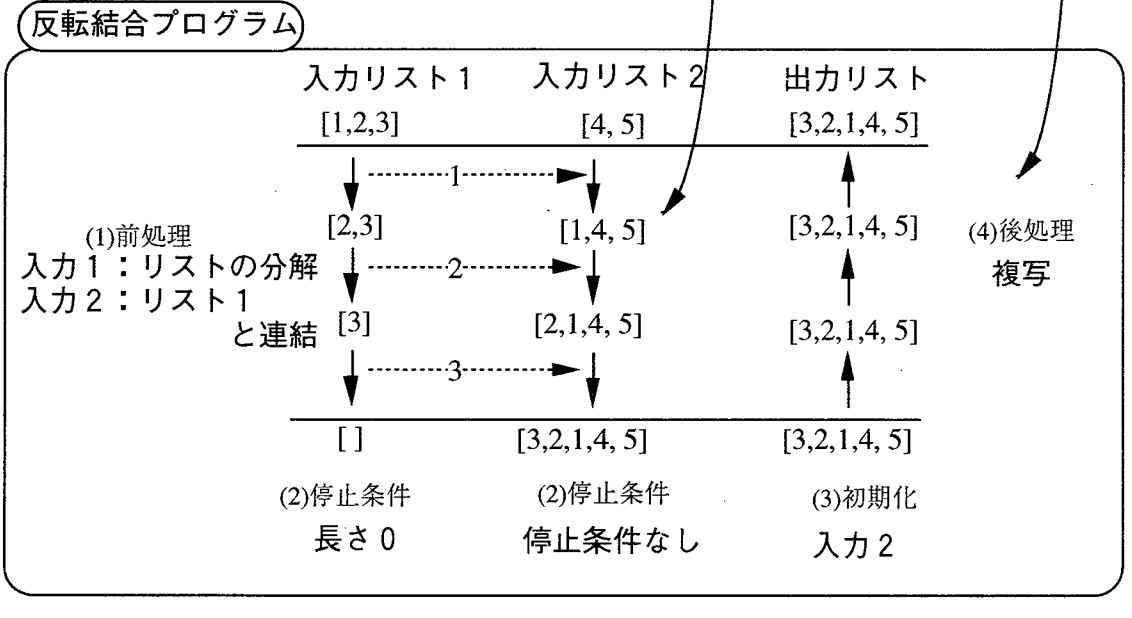
（４）後処理　入力リスト１の先頭要素と出力リストを連結する．

■２つの入力リストの結合を行うＣの再帰プロプログラムの振舞い



入力１で取り出した先頭要素を出力　で連結するとリスト１の順序は保存され，
入力２で連結するとリスト１の順序は反転する．

■２つの入力リストの反転結合を行うＣの再帰プロプログラムの振舞い



１３ページ

■２つのリストの結合を行うＣの再帰プログラム

結合プログラム

```
リスト　結合( list1, list2 )

{
        変数はリスト型;

        if ( 長さ０( list1 ) ) {          ―― 停止条件
                return ( list2 );          ―― 初期化
        }

        head = 先頭要素 ( list1 );         ―― 前処理
        tail = 残り ( list1 );

        list3 = 結合 ( tail, list2 );      ―― 再帰呼び出し

        list4 = 連結 ( head, list3);       ―― 後処理
        return ( list4 );
}
```

■入力リスト１の要素を反転して入力リスト２と結合するＣの再帰プログラム

上のプログラムを参考にして，反転結合を行うＣの再帰プログラムを作成して下さい．

１４ページ

■２つのリストの結合を行うＣの再帰プログラム

結合プログラム

```
リスト　結合( list1, list2 )

{
        変数はリスト型;

        if ( 長さ 0 ( list1 ) ) {
                return ( list2 );
        }

        head = 先頭要素 ( list1 );
        tail = 残り ( list1 );

        list3 = 結合 ( tail, list2 );

        list4 = 連結 ( head, list3);
        return ( list4 );
}
```

停止条件
初期化
前処理
再帰呼び出し
後処理

■２つのリストの反転結合を行うＣの再帰プログラム

反転結合プログラム

```
リスト　反転結合( list1, list2 )

{
        変数はリスト型;

        if ( 長さ 0 ( list1 ) ) {
                return ( list2 );
        }

        head = 先頭要素 ( list1 );
        list3 = 連結 ( head, list2);

        list4 = 反転結合 ( tail, list3 );

        return ( list4 );
}
```

相違点
停止条件
初期化
前処理
再帰呼び出し
後処理

１５ページ

■「末尾挿入」プログラムを書いて下さい．

制限時間　５分．　残り時間が画面に表示されています．時間まで挑戦して下さい．

末尾挿入プログラムの入出力例

入力		出力
[1,2,3]	末尾挿入 →	[1,2,3, z]

１６ページ

■「末尾削除-反転結合」プログラムを書いて下さい．

制限時間　５分．　残り時間が画面に表示されています．時間まで挑戦して下さい．

末尾削除-反転結合プログラムの入出力例

入力１	入力２		出力
[1,2,3]	[4,5]	末尾削除-反転結合 →	[3,2,4,5]

１７ページ

■「対称結合」プログラムを書いて下さい．

制限時間　５分．　残り時間が画面に表示されています．時間まで挑戦して下さい．

対称結合プログラムの入出力例

入力１	入力２		出力
[1,2,3]	[4,5]	対称結合 →	[1,2,3,3,2,1,4,5]