プログラミングに関する認知科学的研究(1)

松　原　康　夫

Cognitive Scientific Study of Programming (1)

Yasuo MATSUBARA

Computer programming is one of the most highly intellectual human activities. How a man produces a program, however, has not been investigated vastly.

Despite of efforts to automate the programming, it has not been successful except for the cases of very small and restricted problems.

Investigations of human programming process may contribute not only to automatic programming but also to pedagogy of programming. In this paper, the human process of programming is considered from the cognitive scientific point of view.

1．はじめに

プログラミング，つまりコンピュータが実行するプログラムを作成する作業は，人間の知的活動の中でも最っとも高度なものの一つである。しかしながら，その活動がどのような過程を経て行われているかに関しては，殆んど研究されていないようである。

他の代表的な知的活動としては，目で視たものを理解する画像認識と，言語を理解したり文章を書いたりする自然言語処理がある。これら及びこれらに関連した分野については多くの研究がなされている。この事は，人間の情報処理活動の最も基本的なものであることからすれば当然かも知れない。

ここで，プログラミングという知的活動をとりあげるのには，以下に示すような，いくつかの理由がある。

(1) プログラミングは，他の知的活動とは異った面があるため，他の活動の研究からは得られない知見の得られる可能性がある。

例えば，プログラミングは特に問題解決的な面が強いことや，人間の思考過程を客観化するという特異な性質がある。

(2) プログラミングを自動化しようとする試みはなされているが，極めて小さい問題についてしか実現されておらず，少くとも現在のところ，人間にしかできない作業である。

(3) 工学的な立場から，プログラミングの効率化，プログラムの品質の向上が画られているが，決定的な方法論は存在しないようである。これは，人間的要素を研究しなければ解決できない問題である。

(4) 効果的な，プログラミングの学習法又は教育法が確立していない。又，プログラミングに対する個人の適性というものがあるかどうか，あるとすれば何に起因するか解っていない。

本論文では，認知科学的な立場から，人間のプログラミング過程を研究する。認知科学とは，人間の知的活動を総合的な立場から研究するものであるが，詳しくは文献［1］等を参照されたい。

又，本研究の応用的目標としては，次のようなものが考えられる。第一には，人間と同様な方法でプログラムを構成するシステムを構築することである。又，第二には，教育への応用である[2]。ピアジェ[3]の影響を受けた，パパートのLOGO[4]や，水道方式[5]による算数教育は，人間の知的過程の分析を教育面に応用した良い例である。本論文では，これらと同様に，プログラミング過程の分析からプログラミング教育上有益な知見を得ることをも意図している。

2．プログラムに関する研究

次に，これまでのプログラムに関する研究について振り返ってみよう。

これまでのプログラム又はソフトウェアに関する研究には，理論的か実際的かという点から考えても，非常に広いスペクトルの範囲があるように思われる。もっとも理論的な方面としては，いわゆるプログラムの理論やスコット理論のように，主としてプログラムの数学的性質を調べる分野があげられる。逆にもっとも実際的な分野として，いわゆるソフトウェア工学やソフトウェアの品質管理等がある。ソフトウェア工学では，単に短い時間でプログラムを作成することよりも，むしろいかに理解し易すく，メンテナンス可能なプログラムを作成するかについて，実際的な方法を考えるものである。又，これと関連して作成されたプログラムの品質を如何にして評価し，又管理するかという研究が最近成されつつある。これは，プログラム又はソフトウェアを一つの商品として捉える，もっとも実際的な立場に立つものである。

又，プログラムを作成するに当って、先ずどのような問題を解決するのであるかを，正確に記述することが肝要となる。そのために，プログラムの目的又はプログラムの仕様を厳密に記述する，形式的仕様記述法の研究がある。これは，ソフトウェア工学よりはかなり理論よりの所に位置付けられる。

これよりもさらに理論よりの分野として，プログラムの正当性や停止性を示す方法論がある。ただし，厳密には，与えられた仕様記述に関して，プログラムが正しいかどうかを議論しなければならないので，形式的仕様記述法とは離すことのできない関係にある。しかしこれらの方法論は，多くの労力を要するので，現実に広く使われるには致っていない。

ソフトウェア工学では，人間がいろいろの手法又はツールを駆使して，良いプログラムを作ろうとするものである。これに対して，コンピュータ乃至プログラミング言語の方を人間に近づけようとする試みがある。広い意味では人工知能全般がこれに概当することになるが，より現実的なものとして，超高級言語に関する研究がある。これは，特定の応用分野を限定して，その中で良く使われる高級な機能を，言語の基本機能として用意するものである。パーソナルコンピュータで良く使われる簡易言語は，これと同様の考え方を含んでいると言えるだろう。

又，仕様記述そのもの，又はそれに非常に近いものをそのまま実行させる，言い換えれば，直接実行可能な形で仕様記述を行う，という考え方から，ロジックプログラミングが研究されている。

最っとも理想的な形としては，仕様記述から自動的にプログラムができることである。その方向の研究も成されているが，現在のところ極めて簡単な問題についてだけしか成功していない。又，仕様記述を正確に作成するという作業が，人間には困難なことである，という問題点も存在する。

以上で概観したように，種々の方面からプログラム又はソフトウェアに関する研究がなされているが，プログラミング効率やプログラムの品質を飛躍的に向上させるような方法論は見出されていない。

ここで一つの疑問が生じる。それは，人間

は何故プログラムを作ることができるのか，という疑問である。もしその仕組が解明されれば，それを機械化することもあながち夢ではなくなり，従ってプログラミング効率や品質を飛躍的に向上させることも可能となるであろう。

3．教育上の立場から

ここで，日本におけるコンピュータ又は情報処理に関する教育を考えてみよう。最大の問題点は，小学校，中学校，高校において全く教育されていないことであろう。このことによって，大学への進学者は，知識の殆んど無いままに進路の選択をしなければならない。従って大学で情報関連のコースを選んだときに，始めて自分に適性があるかないかを知ることになるのである。

又，大学でコンピュータを教育する側から見た場合，全くの初歩から始めなければならないことになる。これは，数学で言えば，もっとも基本となる数の概念や加減乗除等のレベルから，大学で始めることに相当する。国語で言えば，ひらがなの読み書きから始めるようなものである。このことにより，大学教育に適わしい，高度な知識や訓練を与える時間が不足しがちとなる。

教育の中で最っとも細心の注意を要するのが入門教育であろう。小学校の低学年で，読み書きや四則演算を懇切丁寧に，時間をかけて教えているように，大学といえどコンピュータの入門教育には種々の工夫が必要と考えられる。

これは，コンピュータやプログラムに関連した，思考の枠組が全く頭の中に存在しないことによる。そこに，新しく枠組を学生一人一人の内部に構築させなければならないのである。

この場合，プログラミングにおいて使われるのと非常に近い思考の枠組を既に培っている学生は，比較的容易にプログラミング能力を修得するであろう。又，全体としてそれに近い枠組が無くても，その部分として使い得るような枠組を豊富に持ち合わせている学生はそれだけ有利である。

一般に，理科系と文化系の学生を比較して，理科系の学生の方がプログラミングの修得が容易であると言われるのは，主にこのような理由によるのかも知れない。特に数学ができる人の方が，できない人よりも有利と言われるのは，数学的な思考の枠組のなかに，プログラミングに必要な枠組の材料として使えるものがたくさんあるからであろう。(但し数学的な思考に対する適性と，アルゴリズム的な思考に対する適性とは必らずしも一致しないとする意見もある[7]。)

しかし，このことは理科系の学生に比して文化系の学生が，プログラミングに関する適性が小さいことを意味しない。それは単に，それまでに修得した思考の枠組の材料の違いであるにすぎない。

例えば，文化系の学生にプログラミングを教える際に，配列の使い方を教えることの困難を指摘する人がある。つまり理科系では，ベクトルやマトリクスに親和性のある学生が多いのに対して，文化系の学生はそうでない。このことから配列を教えることが困難になるのであるという指摘である。

この事は，筆者にもある程度事実であるようにも思われるが，だからといって文化系の学生が配列を使ったプログラミングを修得できないわけではない。改めてそれに相当する枠組を頭の中に形成してやればよいだけの話しである。

ここで，理科系の学生を教育してきた筆者の経験を振り返ってみよう。

まず言えることは，理科系の学生であっても相当のパーセンテージで落ちこぼれる人がいたことである。とくに，計算機環境としてバッチ処理しか使えなかった時期はひどい状況であった。中には四年間殆んど計算機に触

れもしないで卒業したものもあった程である。それが，一つの学年の人数（60人）と同程度の端末機を有するTSSになってから，事態は画期的に改善された事は事実である。

しかし，それでも約3割の学生は卒業時でも次のようなレベルであったようである。つまり，簡単な演習用のプログラムは作るが，ある目的を持った問題を与えられたときに，それを解決するためのプログラムを自分で作る気がしない。

このような状況が生じた最大の原因（決してこれだけではないが）は，筆者の見るところでは，入門教育の軽視にあったようである。つまり，他の教科が，高校までの知識を前提としていて，その場で完全に理解できなくても後で取り返しがつくのと同列に考えていた嫌いがある。実は，先に述べたように，小学校で読み書きや四則演算を教えるのと同じような手間と工夫が必要なのではないだろうか。

プログラミングにおいて使われている思考の枠組がどのようなものであるかについて，少しでも明らかになれば，それは教育方法に反映させてプラスとなるはずである。又逆に，このような入門教育の場において，学生の理解しにくい点を知ることは，プログラミングのためにどんな思考の枠組が使われているかを明らかにする手掛りとなる。

4．理科系と文化系

前章でも一部触れたが，ここでは理科系と文化系の学生が、コンピュータを学ぶ上での容易さに差を与えるような条件の違いについて考察しよう。

一つには，教える側が無意識の中に，理科系の素養を前提とする教え方をしがちであることが挙げられる。又，プログラミング言語も科学技術計算のために開発されたものが使われることが多い。もっとも，文化系の素養を前提とした，文化系の学生が興味を持ちそうな入門用のうまい題材が殆んど無いことも事実であろう。この点に関しては今後工夫していく必要があるだろう。

もう一つは，プログラムの抽象性の問題がある。つまりプログラムというものは，日常の世界から相当の距離を置いたものだということである。理科系の人間は，このような日常の意味から離れた世界での思考に，文化系の人間より親しんでいるだろう。

しかし逆に，理科系の人間にとって注意しなければならないこともある。それは，プログラムを構成している，変数や代入文等は単なる符号ではなく，むしろ記号としての性質を持つことを忘れがちである。つまり変数等は，プログラムの外部又は内部の何かの対象を指し示したり，象徴するものである。

実際，プログラムが何をするものであるかを理解するためには，変数等が何を表わすかを考える必要がある。

とくに，教える側に立ったときも，各変数が何を表わしているかを，常に考えさせるようにするべきであろう。このような観点から見ると，問題の方からトップダウンに考えさせるという教え方は妥当性がある。

5．プログラミングに必要な知識構造

本章では，プログラミングをする能力というものを，それに必要な知識構造という点から考察する。

プログラミング能力のある人間が，自分のやり方を振り返る機会として，初心者を教育することがある。この場合，もっとも基本的と思われる事柄から始めて，次第に高度な事柄へと進む。その順序付けは，大部分の人が大筋で同意できるようなものであろう。大体以下のようになるだろう。

1．代入文と変数
2．逐次処理
3．入出力
4．条件分岐
5．ループ

6．サブルーチン
7．配列等のデータ構造
8．再帰呼び出し

この順序は全順序というわけではなく，むしろ半順序で表わすべきかも知れない。例えば1～3，については一緒に教えるか又は逆の順序で教える場合もあるだろう。

次に，各段階を教えるに当って，普通の学生が使用できる，よく似た思考の枠組が存在するかどうかについて考察したい。

最初に，代入と変数であるが，変数という言葉が数学に出て来て，似てはいるが意味が異るので混乱を引き起し易い。又，代入文という訳語に問題がある。assignment は割り当てとか割り付けと訳すべきかも知れない。この点は教育上工夫を要するところであり，筆者は箱に中身を入れるという比喩を使うことにしている。

代入文にしても入出力文にしても，高校までの知識の枠組で，これらと似たものは余り無いと考えられる。従って現在の所，教育法としては，繰り返し説明することによって，これらを実行したときの状態変化を，頭の中で思い浮かべられるようにするしかない。

又ここで，初心者にとって疑問となるのは，変数名として何を使っても良いことや，変数に入力文で入れた値を代入文で使えること等であり，説明する必要がある。やはり，単にどうなるかだけでなく，どう使うかをも並行して教える必要がある。

逐次処理の中でも単なる直列処理に関しては，頭の中に枠組が十分に構築されていると思われる。文章は一方向に読むものであるし，何かの計画表というものは全て同じ構造を持っている。

BASIC を教える場合，場合分けに関しては少し混乱するらしい。つまり，条件が成り立つときには指定された文番号に分岐し，そうでないときは次の文に移動する，という考え方が余り自然には感じられないらしい。やはり構造的な IF THEN ELSE の方が自然と考えられる。単なる代入文の直列処理が理解できる者でも，IF 文が入ってくると一時的に混乱するようである。しかし，これは比較的容易に克服できるようである。

ところで，IF 文の働きを理解できた学生は，容易に場合分けのプログラムを書くことができるであろうか。この点に関しては個人差が現われるようである。勘の良い学生であると，特に教えなくとも使うことができるが，そうでない学生には，どのようにして場合分けのプログラムを書くかを教える必要がありそうである。

ループになると，さらに困難を感じるらしい。このような考え方は，高校までの教科には少くとも明らさまには出てこないことによると考えられる。強いて例を挙げれば，楽譜に繰り返しがあることと，子供のゲームであるすごろくに繰り返しがある程度である。

筆者がループを教えているとき，次のような経験をした。ある変数に正整数を入力し，これが正である限り1ずつ引いて行くようなループを作る。そして，そのループの中で，PRINT “＊”； を行う。このプログラムの各文の意味を一つ一つ説明したが，学生にはこれが何をするプログラムかわからない。そこで，これは入力した数だけ＊を一行に表示するプログラムである旨を説明すると，そこで始めて納得したようである。

この経験は大変示唆的である。一つ一つの文の働きだけが解っても，プログラムを理解したことにならないことは言うまでもない。しかし，経験を積んだ者ならば，この程度のプログラムは，見ただけで理解できるはずである。つまり経験者の場合には，与えられた回数だけある動作を繰り返えすためには，一重のループを使えばよい，等といったことが一つのフレームとなって知識の枠組の中に入っているのである。多分，経験者の知識の中には，このような目的と手段の対がたくさん

貯えられているのであろう。従って初心者教育では，このような基本的フレームを，学生が自分の頭の中に構築できるような問題の出し方をするべきである。

水道方式に見られるように，単純なものが把握できてから複合的なものに進むという考え方によれば，単純な一重のループが完全に理解できてから二重ループに進む必要がある。経験者には，一重のループを構成するブロックに，別のループを埋め込む，という入れ子の考え方は極めて自然なものであるが，初心者教育においては，これを前提とする訳にはいかない。

次にサブルーチンであるが，教育上の問題としては，題材として使われる非常に小さいプログラムでは，サブルーチン化する必然性を余り感じないことかも知れない。

しかし，サブルーチンを教えることはそれ程困難ではないようである。初心者にも比較的容易に把握できる概念のようである。

プログラミング言語として，BASIC の最っとも良くない点の一つは，サブルーチンを文番号で呼ぶことである。この点に関しては名前で呼ぶ FORTRAN の方がましである。又 CALL を書かない ALGOL 系の言語の方がすぐれている。又 LOGO[4]では，手続きの名前を指定することによって，その内容を確認したり，編集することができるという便利な点がある。

配列については，先に述べたように，ベクトルやマトリクスに親しみを持つ学生の方が把握し易いであろう。しかし，これらに親しみを持たない学生を教えるには，他の似たものを利用する必要があろう。例えば，ビルの階層とか，番号の付いた室，番地あるいは表等が，日常生活に現われる良い例である。つまり複数の容れ物が一列に並んでいて，番号が付けられているものである。このような材料を使って問題を作ることは教育効果を上げるかも知れない。

又，配列を使うためには，ループがないと意味のある処理が殆んどできないので，少くとも一重のループを完全にマスターしてから配列を教えるべきであることは，多くの人の一致した意見であろう。

最後に再帰呼び出しであるが，現在ではループより後に教えるのが常識となっているようである。再帰を教えるためには，問題自体が再帰的な構造を持ち，従って再帰を使った方がプログラムを簡潔にできるような問題を使って教えるべきであろう。

人間の考え方として，ループと再帰とどちらが理解し易いかは，簡単に言える問題ではない。問題によって再帰を使った方が解り易い場合とそうでない場合があることは確かである。しかし，LOGO では，一定回数同じことを繰り返す repeat 以外に，繰り返しに関する制御構造を用意しておらず，初心者から再帰による考え方を推奨しているようである

ループと再帰に関しては，中学生を対象とした実験で，再帰よりも先にループを教えるべきだという研究がある[6]。

6．プログラミングの過程

本章では，具体的な問題に即して，どのような過程でプログラムが作られるかを考察する。但し，ここでは言語として BASIC を使うことにする。

最初に最っとも簡単な次のような問題を考える。

問題１：２つの数を入力して，その和を出力するプログラムを作りなさい。

このような問題が与えられたとき，人間はプログラムが入力，処理，出力という３つのステップから構成されるという最っとも単純なフレームを思い浮かべるだろう。さらに下位構造として，入力とは何かの変数に外部から値をセットすることであり，処理は代入文で実現し，出力は変数の値をディスプレイに表示又はプリンタに印字することであるとい

う知識が呼び起こされるであろう。又，入出力にはどんな文が使われるか等も思い起こされる。

処理に関しては，単なる代入文から，条件分岐，ループ，サブルーチンを組み合わせたものまであり得る。

人間は，以上の問題をこのようなフレームに対応させた形で解釈していくことになる。とくに経験者は容易にこのプログラムを書くことができる。そして例えば変数としてX，Yを用い，Z＝X＋Y という代入文を書くかも知れない。

ここで，変数名を選ぶのにも，人間らしい傾向があることが予想される。最初はX，Y，Z 等から選び，それで足りなくなればU，V，W 等から選ぶというのが妥当なところかも知れない。但し，問題に変数名あるいは数量の名前等が指定されていれば，もちろん同じ名前を使うことになる。

又，Z＝X＋Y という代入文を書かずに，出力文に直接X＋Yと書くかも知れない。これは短いプログラムを書くか，整ったプログラムを書くかの選択であり，経験者でも時によって異る選択をするだろう。

問題の中で，二つの数という表現がある。これに対して人間は暗黙の中に単精度の変数で表わせる数を仮定している。しかし問題によっては整数又は倍精度の変数を選ぶだろう。これはフレーム理論ではデフォールト値として単精度の変数が選ばれることに相当する。しかしここにも構造が存在することに注意する必要がある。すなわち変数として，単精度，倍精度，整数の各型が数を表わすのに使うことができるという知識が必要となる。

又，これらの3つの型でも表現できないときには，文字型あるいは配列を使って数を表現する必要がある。この辺の知識になると，初心者と経験者では差がついてくる。初心者では3つの型で表現できないときはお手上げになるであろう。

経験を積んだ者にとっても，配列を利用して数を表現するという考え方は，かなり高度な知識に属するだろう。その内容としては，第一に，配列を使えば大きな数が表現できるという判断が必要である。第二には配列を使ってどう数を表現するかという表現方法の問題がある。第三には，表現された数に対する演算や操作をどうするかといった問題がある。これらの判断は，より高度なプログラミングの能力に属するものであり，この段階で取り扱うのは適当ではないだろう。

では次の問題を考えよう。

問題2：正整数nを入力して，1からnまでの数を順に表示するプログラムを書きなさい。

このプログラムでは，nを表わすために，何らかの変数が必要である。一番使われそうな変数名はNであろう。何かの事情でNが使えないときは，NN とか N1 又は N0 等を使うのが一般的であろう。

この問題では，nを入力するという記述の後に，表示するという記述がくるので，処理の順序と一致している。しかしこの問題は次のようにも記述できるはずである。

問題2：1からnまでの数を順に表示するプログラムを書きなさい。但しnは正整数であり，キーボードから入力するものとする。

このように記述されては，入力と表示の順が逆となってしまう。これを正しく解釈するためには，nを使う前にnが入力される必要がある，というルールを使う必要がある。但し，このルールはもっと一般的なルールから導き出すようにすべきかも知れない。

次に，1からnまで順に表示する，という記述であるが，この場合1刻みであることはデフォールト値としてよいだろう。又，1からnまでという表現は，順にという表現と一致する。

1からnまで，何かを繰り返えすということは，非常に基本的な概念であるため，大抵

のプログラミング言語でFOR文やDO文等，そのための制御文を用意している。だから単にプログラムができればよい，という場合には，自動的にFOR文を割り当てればよい。

しかし，それではループを理解したことにはならない。教育の場においては，敢えて最初はFOR文を教えないことがある。ここでもFOR文を使わないでループを構成することを考えてみよう。

最っとも原始的な考え方は次のようになるだろう。

始めに1を表示する。次に2を表示する。

ここまでくると，2を表示することと1を表示することは大変よく似ていることに気づく。その差は1と2の差であり，一つの変数（仮りにXとする）を使って，その値を1増やせばよい。つまりPRINT Xの後でX=X+1を行えば，次に再びPRINT Xを行うことになる。

これ以降もX=X+1とPRINT Xが繰り返えされることが判かる。ここに到って，同じ文を何回も書かずに，X=X+1の後でGOTO文を書いて前に戻せば良いことが判る。従って次のようになる。

```
10  PRINT    X
20  X=X+1
30  GOTO  10
```

しかし，これでは無限ループになるので，途中に，ループから抜け出す文を入れる必要がある。ここで初心者は，単なるGOTO文をどこかに入れることを考えるかも知れない。それではループが成り立たないので，条件文が必要なことに気づく。

IF文を入れる場所は，三箇所程度考えられる。第一は先頭であり，第二はPRINT Xの後である。第三はGOTO文の前となる。この辺の選択は，それぞれの効果を試してみてその結果によって選択することになる。

とくに第三の場合には，最後のGOTO文そのものを条件分岐に書き換えても同じ効果を得られることに気が付くかも知れない。

以上のようにしてループを構成するためには，前に戻るGOTO文によって，繰り返しを行うループが構成できるのであるという知識は最低限必要かも知れない。

同じ問題を，経験者が解く場合には，最初から一つの，ループに関するフレームに当てはめて考えるのであろう。

それでは，そのフレームはどのような構造を持つだろうか。ループにもいくつかのタイプがある。while do型のもの，repeat until型のもの，そしてループの途中から抜け出るもの等がある。上の問題のループは，条件文の位置によっていずれのタイプのものにもなる。又，このループでは，ある回の実行と，次の回の実行の差が，単純なX=X+1という増加文によって実現されている。しかし他のループでは，もっと複雑な処理によって，次の回の実行との差が実現される場合もある。このような区別の方が，抜ける位置による区別よりも重要かも知れない。

さらに，ループに関するフレームの具えるべき情報は，そのループの実行の引き起こす効果に関するものであろう。

上のプログラムでは，単に出力だけが効果として残る。他の例としては，ある変数の値に累積的に結果が得られることもあるし，一つの配列の内容に，その効果が残される場合もある。

このような，多岐に渡る効果を一般的に表わす方法として，一階の述語論理が使われることが多い。しかし，これは余りにも一般的でありすぎるように思える。人間が，ループに限らず，プログラムの効果を理解するのには、もっと多種の概念を，その場合に応じて使っているように思われる。

今ここで，それらを明らかにすることはできないが，一つの重要な枠組として，線型の離散構造とでも呼ぶべきものがあるように思う。これは0からnまでの整数の集合といっ

てもよい。時によってnは∞のこともある。この構造は時系列の中にも現われるし，空間的にも現われる。

ループの実行においては，繰り返しの範囲が何回か繰り返えされるが，これは時系列上の線型離散構造である。又，問題2のプログラムで効果として現われる表示は，空間的に現われた，線型離散構造である。

7．むすび

本論文では，プログラミングという，人間の一つの知的過程に関して，認知科学的な立場から検討を加えた。

プログラミングの認知科学として，考察すべき問題は数多く存在するが，本論文で取り上げたのはその中の極く一部である。しかも本論文ではそれぞれの問題に対して，確固とした結論めいたものを提示していない。むしろ，いろいろな問題点の存在を明らかにしたといえるだろう。

方法としては，教育上の経験と，自分自身がプログラミングを行うときにどうするか，という内省によるところが大きい。

心理学においては，科学であろうとする立場から，このような内省という方法を極力避けて，あくまで客観的な証拠に依拠した議論を行おうとする。最終的に科学として確立するためには，確かにその段階に到る必要がある。

しかしながら，認知科学ではその段階よりもはるかに前の段階にある。従って，具体的な事実に照らしてモデルを検証するというよりも，具体的な事実をヒントとしてモデルを構築することの方が，現在の認知科学の主な仕事であるように思われる。

そのモデルの性格として，第一に人間の強力な知性を説明できること，そして第二にはコンピュータのプログラムとして具体化できることが要求される。

今後の課題として，さらに多くの問題点を指摘するとともに，プログラミング過程をより具体的に分析し，コンピュータに乗せることができる形にモデルを近づけていくことが必要である。

謝　　辞

ピアジェの文献に関してお世話になった，田中裕次先生並びに川上善郎先生に感謝します。又，教え方の良い例を示された前田英明先生と，水道方式について指摘いただいた三澤常男先生に感謝します。

参考文献

(1) D. A. Norman ed.:"Perspective on Cognitive Science", Ablex Publishers Company (1981)
佐伯　胖監訳："認知科学の展望"産業図書(1984)

(2) 佐伯　胖："コンピュータと教育"，岩波新書(1986)

(3) Jean Piaget:"Six Études de Psychologie", Éditions Gonthier (1964)
滝沢武久訳："思考の心理学"，みすず書房(1968)

(4) S. Papert:"Mindstorms－Children, Computers, and Powerful Ideas", Basic Books (1980)
奥村貴世子訳："マインドストーム―子供，コンピュータ，そして強力なアイデア"，未来社(1982)

(5) 遠山　啓，銀林　浩編："水道方式入門―整数編―"，国土社 (1971)

(6) 安西祐一郎："認知科学と人工知能"，bit 1月号 (1986)，PP.24－29，共立出版

(7) D. E. Knuth:"Algorithmic Thinking and Mathematical Thinking", American Mathematical Monthly, March (1985) PP. 170－181
一松　信訳："算法的思考と数学的思考"，bit 4月号 (1986)，PP. 4－16，共立出版