知識工学と人工知能　59－13
（１９８８．７．７）

類推、帰納の概念を導入した プログラミング知識の学習メカニズム

今中　武　　上原邦昭　　豊田順一
大阪大学産業科学研究所

プログラマはプログラムを作成、読解していくうちに、プログラム間の類似性を発見し、共通の有用な情報を抽象化してプログラミング知識を学習する。本報告では、類推、帰納の概念を用いてプログラマの学習過程をモデル化する手法を示した。本モデルの特徴は、我々がプログラムを理解するときに注目する視点(プログラムの仕様、データ構造など)に対応させて、いくつかのサブモデルを導入していることである。これにより、仕様だけでなく構造などが類似しているプログラムに対しても、プログラマが類似性を発見し、学習するといったことを本モデルで説明できる。

A Model of Learning Process by use of Analogical reasoning and Inductive reasoning

Takeshi IMANAKA, Kuniaki UEHARA and Jun'ichi TOYODA
The Institute of Scientific and Industrial Research
Osaka University, 8-1 Mihogaoka, Ibaraki 567, Japan

Programmers can get programming knowledge during the practice of programming. Programmers retrieve similar programs by analogical reasoning, and extract the programing strategy from the similar programs by inductive reasoning. From this viewpoint, we have developed a model which can be viewed as expertise transfer of the programming knowledge. Our model is based on analogical reasoning and inductive reasoning. Because, The feature of this model is containing some sub-models which correspond to programmer's views of programs. Sub-models are used to handle similarity between program specifications, program structures etc. This model can be applied to expert systems which synthesize or understand programs.

1 はじめに

近年、プログラム合成の研究ではプログラムのモジュールやサブルーチンを部品と見なし、これらの部品を組み合せて要求仕様に合う新たなプログラムを合成する手法が研究されている。しかしながら、今後ますます多様化すると考えられる要求仕様に対し、予め全ての部品を準備することはほとんど不可能になるといった問題が生じる。このような問題を解決するための方法として、プログラム合成システムに学習メカニズムを組み込むといったことが考えられる。学習メカニズムを組み込めば、システムが自動的にプログラム部品の数を増やし、多様化した要求に答えることができる。

一方、人間は多くのプログラムを作成、読解していくうちにプログラム作成のノウハウを学習している。これは新たなプログラムの作成時に、あるいは読解時に過去に作成した類似プログラムを想起し、何らかの有用な情報を抽出することができるためである[1]。このように、類似プログラムから有用な情報を取り出す操作は類推の過程である。また、類似した複数のプログラムを想起すると、それらのプログラムに共通の有用な性質を抽象化し、プログラミング知識として修得することができる。これは帰納推論による学習の過程である。したがって、人間の行っている学習メカニズムは類推、帰納推論の概念を用いてモデル化することができる。このモデル化が成功すれば、先に述べたようにプログラム合成に応用することができる。また、熟練のプログラマが他人の作成したプログラムを理解することができるように、プログラム理解システムにも応用可能である。

以上のような考え方に従って、我々は類似したプログラムの発見と、プログラムの一般化に基づく学習をモデル化している。本モデルはPrologプログラミングの学習を対象としており、プログラミング技術を獲得するためにプログラムの構造や仕様などに対応した4種類のサブモデルを導入している。このため、プログラムの仕様が類似している場合だけでなく、プログラムの構造が類似している場合なども学習の対象とすることができるようになっている。以下、2章では本モデル化で用いるサブモデルの必要性について述べ、3章では各モデルについて詳しく説明する。また、4章では4つのサブモデルを用いた学習過程について例をあげて説明する。

2 プログラミングにおける類推、帰納

プログラマはプログラムを目にしたときに、文字だけを単純に認識するのではなく、プログラムがどういった働きをするものなのか、各変数は何を表し、プログラムの動作にどうかかわり合っているのか等、複数の見方をする。したがって、プログラマの発見する類似性は、プログラムの仕様が類似している場合だけでなく、プログラムの構造が類似している場合もある。たとえば、fatherの関係からgrandfatherの関係を導きだすプログラム

grandfather(X, Y) ← father(X, Z), father(Z, Y).

と、motherの関係からgrandmotherの関係を導きだすプログラム

grandmother(X, Y) ← mother(X, Z), mother(Z, Y).

は、プログラムの動作などを比較するまでもなく類似しているものと判断することができる。これは2つのプログラムの仕様が類似しているためである。一方、2つのリストを結合して出力する append プログラム

append([], X, X).
append([A|B], C, [A|D]) ← append(B, C, D).

と、リスト中に特定のアトムが存在するかどうかを調べる member プログラム

member(X, [X|_]).
member(X, [Y|Z]) ← member(X, Z).

は、プログラムの仕様からは共通点が少なく類似していない。しかしながら、再帰呼び出しに伴う入力リストの扱いなどの点でプログラムの構造は類似している。

以上のように、仕様や構造など複数の見地からプログラムの類似性が発見されるために、プログラマの学習過程のモデル化には、プログラムを理解する時の複数の見地に対応させていくつかのサブモデルを用意する必要がある。したがって、本モデル化手法では(1)現実関係モデル、(2)Prolog 関係モデル、(3)プログラム関係モデル、(4)プログラム構造モデルをサブモデルとして導入している。これらの4つのサブモデルは、それぞれ(1)プログラムが扱っている現実世界の事物、関係はどういった構造を持っているのか、(2)プログラム中では現実世界の事物がどう扱われているのか、(3)プログラムが計算できる関係や事物はどういうものなのか、(4)プログラムの構造はどうなっているのかを表したものである。

3 プログラムを表すための4つのサブモデル

3.1 現実関係モデル

プログラムを理解、あるいは作成する時には、そのプログラムで扱う現実世界の事物がどう関わり合っているのかを知っていなければならない。たとえば、学生の成績の平均点を計算するプログラムを理解するためには、成績と平均点がどういった関係にあるのかを知る必要がある。特に、Prolog においては述語名に事物間の関係を、引数に事物を割り当てることが多く、現実世界の知識を関係モデルで表しておくと都合がよい。現実関係モデルの例を図1に示す。事物がノードに、事物間の関係がアークに対応づけられている。また、アークにつけられたラベルは、アークで結ばれたノード間に成り立っている関係の名前を示している。(1)は、関係'先祖'が関係'父'、'母'を繰り返した関係と等価であることを示している。(2)は、事物'a'と集合'b'の間の関係'子供達'が成り立つためには、事物'a'と集合'b'の要素の間に'父'(または'子')、あるいは'母'(または'子')の関係が成り立っていなければならないことを示す。また、(3)は、線形集合の大小関係を示している。この関係

は、2つの線形集合$[a_1, a_2, \cdots, a_n]$, $[b_1, b_2, \cdots, b_n]$の任意のm番目の要素(a_m, b_m)を比較して、$a_m < b_m$ならば$[a_1, a_2, \cdots, a_n]$で表される集合の方が$[b_1, b_2, \cdots, b_n]$で表される集合よりも小さいことを示している。以上のような現実関係モデルで表される知識は、プログラムと独立して存在するものであり、学習における背景知識に相当している。

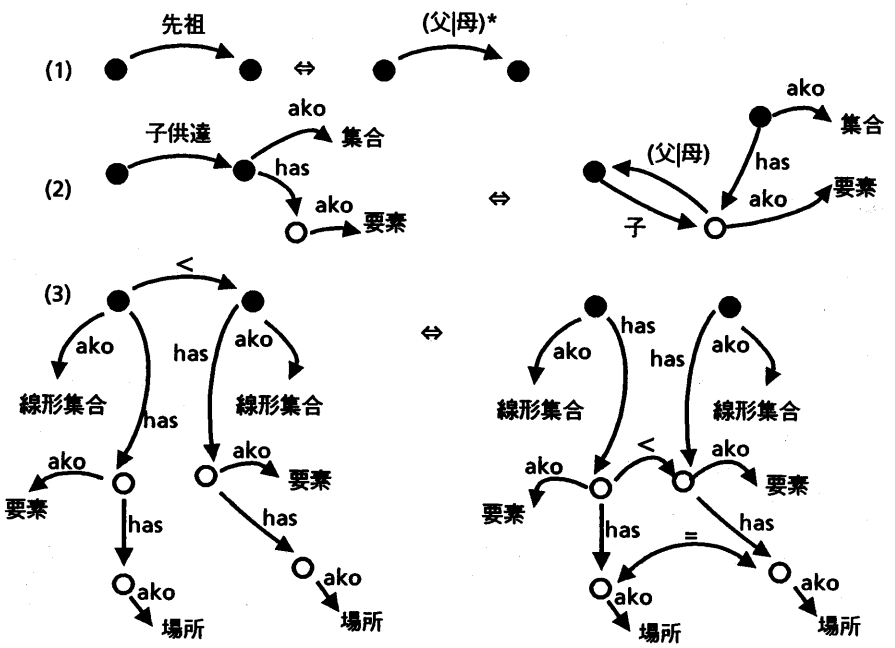

図1　現実関係モデル

3.2　Prolog関係モデル

現実関係モデルで表される事物の関係をプログラムで計算する際には、一旦Prologで扱えるデータ構造を用いて表現し直さなければならない。たとえば、図1の線形集合の大小関係をPrologプログラムで計算するために、線形集合をリスト構造に、線形集合の要素をリスト要素に写す必要がある。Prolog関係モデルは、Prologの持つデータ構造を予め記述したものである。これは、プログラマが持っているPrologのシンタックスに関する知識に相当している。たとえば、図2はリスト構造とリスト要素の関係やリストの持っている属性を表しており、1)リストが要素を持っていること、2)リスト要素には場所の順序があること、3)要素は通常有限で要素数を数えることができることなどが示されている。

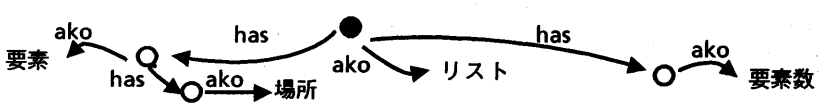

図2　Prolog関係モデル

3.3 プログラム関係モデル

Prologプログラムが計算できる関係は、プログラム中で用いられている関係のみの組合せで表すことができる。たとえば、入力変数'X'、出力変数'Y'に対して、'a(X,Z), b(Z,Y)'というプログラムは、'X'と関係'a'が成り立つ事物'Z'が存在し、その事物'Z'と関係'b'を持っている事物'Y'を求めていることになる。この関係を表したものがプログラム関係モデルであり、図3のようなモデルで表現することができる。同様に、2つの入力リスト(入

図3 プログラム関係モデル

力1、入力2)に対して1つの出力リスト(出力1)を求めるappendプログラムのプログラム関係モデルは図4のようになる。プログラム中に"X=[A|B]"の形式の述語が存在する場合は、変数'X'に対応するノードを頭部'A'と尾部'B'に分割しているために、破線を用いて分割していることを示している。この図からappendプログラムが、"入力1が空で入力2と出力1が等しい"場合、または"入力1が空でなく、入力1の頭部が出力1の頭部と等しく、入力1の尾部と入力2と出力1の間に関係appendが成り立っている"場合に成立する関係を計算していることがわかる。

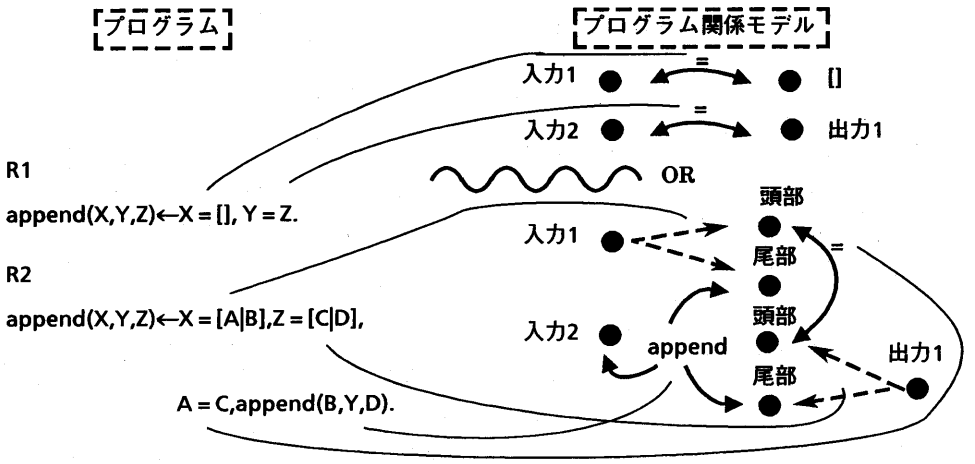

図4 appendプログラムのプログラム関係モデル

3.4 プログラム構造モデル

プログラマは、Prologプログラムを見た時に、複雑に組み合わされている手続き間でデータがどのように受け渡されているのかといった点に注目する[2]。Prologで用いられるデータの受渡しは(1)連続(2)並列(3)分岐に分類でき、これらの3つのパターンがどう組み

合っているのかを図式化したものがプログラム構造モデルである。プログラム構造モデルは、コーディングの際のプログラム作法に対応している。(1)連続は、述語が連言の関係で並んでおり、かつ述語の実行によって得られた出力結果が次の述語実行時の入力として用いられるときに対応している。たとえば、'a(X, Y), b(Y, Z)' において変数 'Y' が述語 'a' の出力で、述語 'b' の実行時に入力として用いられる場合には、述語 'a' と述語 'b' は連続の関係である。この場合、述語 'a' を実行しなければ述語 'b' を実行することはできない。(2)並列は、述語が連言の関係で並んでおり、かつ述語間で連続の関係がない場合である。たとえば、'a(X), b(Y,Z)' では、データの受渡しがないために並列であり、述語 'a' 'b' の実行順序は可換である。(3)分岐は、(1),(2)と異なり厳密にはデータの受渡しではないが、

a(A,B) ← b(A).
a(A,B) ← c(A),d(B).

なるプログラムを考えた場合、データの受渡しについては1つ目のルールと2つ目のルールが完全に独立であり、この独立関係を表すためのものである。したがって、2つのルールの条件部にある 'b(A)' と 'c(A), d(B)' は分岐関係である。プログラム構造モデルの例として append プログラムの構造モデルを図5に示す。プログラム構造モデルは図に示したように、append プログラムの各引数に入出力の種別を書き込み、"入力＝出力" になった場合は代入、"入力＝入力" になった場合は判定といった操作名をつける。また、"入力＝[A|B]"、"出力＝[C|D]" はそれぞれリスト構造の分割、併合を表す。ただし、"出力＝[C|D]" の場合は併合するものの値 'C', 'D' が決まってから処理をするものと考えるために、変数 'C', 'D' に代入処理を行う述語を捜し出し、その処理よりも後ろに移動させる。たとえば、append プログラムのルール (R2)の条件部にある "Z＝[C|D]" は、入出力の種別を書き込んだ時に "出力＝[C|D]" になる併合処理であり、図5のプログラム構造モデルに示したように変数 'C', 'D' に値を代入する処理、"A＝C", "append(B, Y, D)" よりも後ろに併合処理を移動させる。以上のように変換した上で、先の連続、並列、分岐の関係に従ってモデルを構成する。

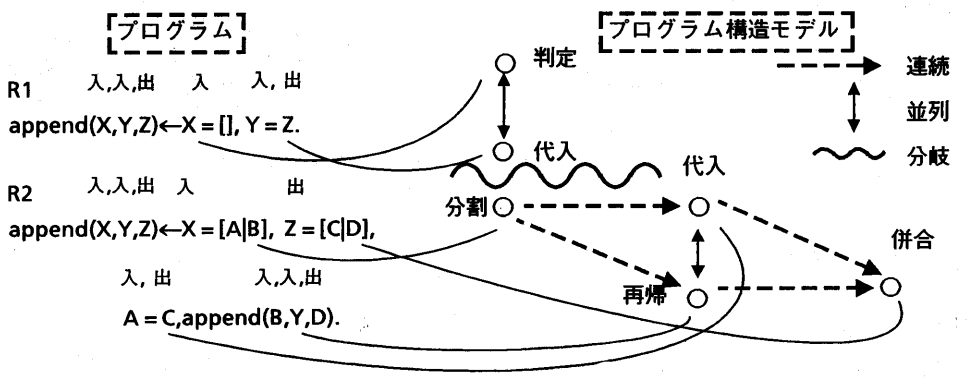

図5 append プログラムのプログラム構造モデル

4 類推、帰納を用いた学習過程のモデル

4.1 現実関係モデルとProlog関係モデルを用いた類推と帰納

Prologプログラムが計算できる関係は、Prologのデータ構造間の関係に限られるが、我々は、この関係をあたかも現実世界の事物間の関係と見なしている。たとえば、grandfatherのプログラム中で'f','a','t','h','e','r'といった6つの英文字で表された関係を現実世界の関係'父'であると見なしている。したがって、Prologのデータ構造では、同一の関係を計算していても、現実関係との対応づけによっては全く類似していない事物間の関係を計算しているものと見なすことができる。たとえば、迷路探索のプログラムで、ある地点Aとある地点Bの間に道がある関係を'f(A, B)'と表していたと仮定する。この時、AさんとBさんが兄弟であることを'f(A, B)'と表せば迷路探索のプログラムで兄弟の関係を求めることができる。これは、迷路探索で求める関係と兄弟の関係は、それぞれ'道'、'兄弟'といった現実関係モデル上の関係をProlog関係モデルの'f(A,B)'に対応づければ、Prolog関係モデル上で同様の構造を持った関係と見なせるためである。さらに、隣を表す関係から、隣の隣を求めるプログラム、友人を表す関係から友人の友人を求めるプログラムなどが、同様のデータ構造を割当てれば、ほぼ同一のプログラムで計算できる。したがって、このようなプログラムは類似しているものと見なすことができるために、1つのグループにまとめて類似部分を抽象化する。

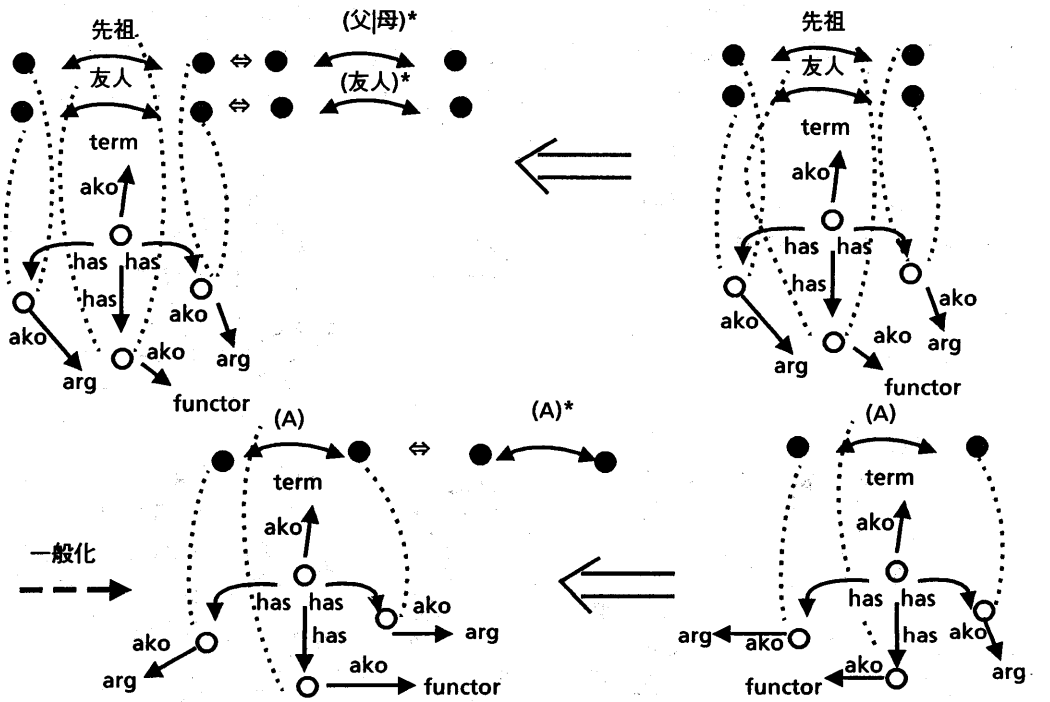

図6 現実関係モデルでの知識獲得例

図6は、'友人'、'先祖'といった関係が持っている構造とそのデータ構造の割当て方(図中の点線)に類似性が発見されて、それらを一般化して知識を獲得している様子を示している。図中の記号'⇐'は、これらのプログラムが矢印の右側の関係を用いて左側の関係を計算するプログラムであることを示している。この知識は、1)関係Aが関係Aの繰り返しであること、2)関係Aを満たす事物を関係Aから求めていること、3)事物をそれぞれ引数に割当て、関係を述語に割当てていることの3点が、これらのプログラムの類似点であることを表している。

さらに、複雑な例として、木構造をボトムアップに辿るプログラムを考える。このプログラムで、親のノード'a'に対して子のノード'$b_1,b_2,\cdots,b_n$'が存在することを'f(a, [b_1, b_2, $\cdots$, b_n])'と表しているものと仮定する。このとき、図7の'⇐'の右辺に示すように図1の現実関係モデルの子供達関係と木構造の関係'f'との間に類似性が発見され、子供達関係をProlog関係モデル上で'f(親,[$子_1$,$子_2$,$\cdots$,$子_n$])'と対応づければ、木構造のボトムアップ探索のプログラムが先祖を求めるプログラムとなる。したがって、2つのプログラムは類似しており、図6の例と同様の知識獲得が可能である。

これらの例から解るように、同じ先祖を求めるプログラムでも、現実関係モデル上の事物や関係をProlog関係モデル上のデータ構造に写像するときに、'term'を使うのかリストを使うのかで迷路探索のプログラムと類似したり木構造のボトムアップ探索のプログラムと類似したりする。実際に先祖を辿るプログラムを作る場合でも、プログラマによってデータ構造の割当て方が異なり、それによってプログラムの作り方も異なるはずである。これは、プログラマ毎に現実関係モデルやProlog関係モデルにおいて発見する類似プログラムが異なり、異なった類似プログラムを抽象化して学習しているためである。

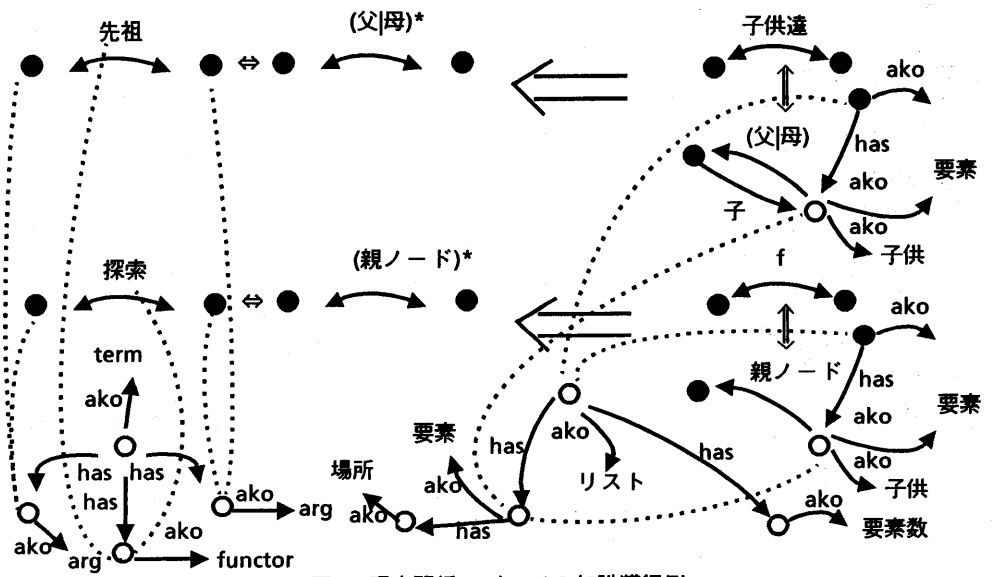

図7　現実関係モデルでの知識獲得例

4.2 プログラム関係モデルとプログラム構造モデルにおける類推と帰納

プログラム関係モデルにおいて類似している2つのプログラムを発見すると、その類似部分の関係を取り出し、プログラムの構造と共に蓄積する。たとえば、'change'の関係で規定された法則に従って、入力文を書き換えて出力するalterプログラム

```
alter(A,B)←A = [],B = [].
alter(A,B)←A = [H|T],B = [X|Y],change(H,X),alter(T,Y).
change(A,B)←A = you,B = i.
change(A,B)←A = are,B = [am,not].
```

のプログラム関係モデルは図8に示すようになり、図4のappendプログラムと類似している。したがって、appendプログラムとalterプログラムが計算できる関係の構造が類似している。両者が共に計算できる入出力の関係は、"入力が[]ならば出力が直ちに決まること"、"appendでは入出力集合の先頭が'='、alterでは'change'を満たせば残りは関係を保存すること"、また"計算している関係を持っている2つの入出力はリストであること"の3点が共通している。したがって、このような条件を満足する関係を計算するプログラムの構造は、appendプログラムとalterプログラムの構造の類似部分と同様になる。appendプログラムとalterプログラムから得られたプログラム関係モデルとプログラム構造モデルを図9に示す。図は、プログラム関係モデルで表されるような関係を計算するためには、プログラム構造モデルで表されている構造を持つプログラムを用いればよいことを表す。

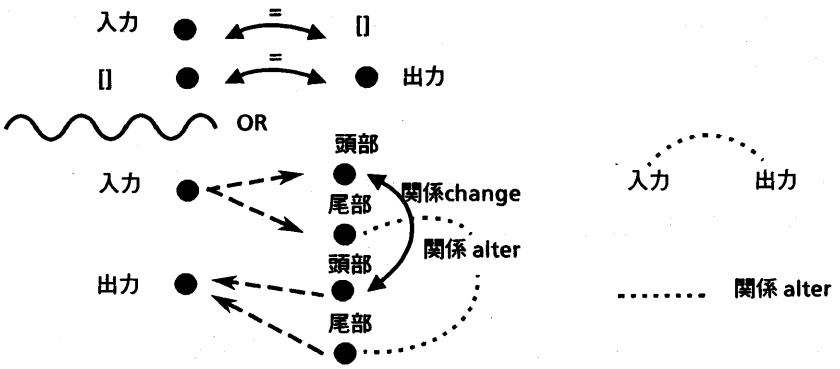

図8 alterプログラムのプログラム関係モデル

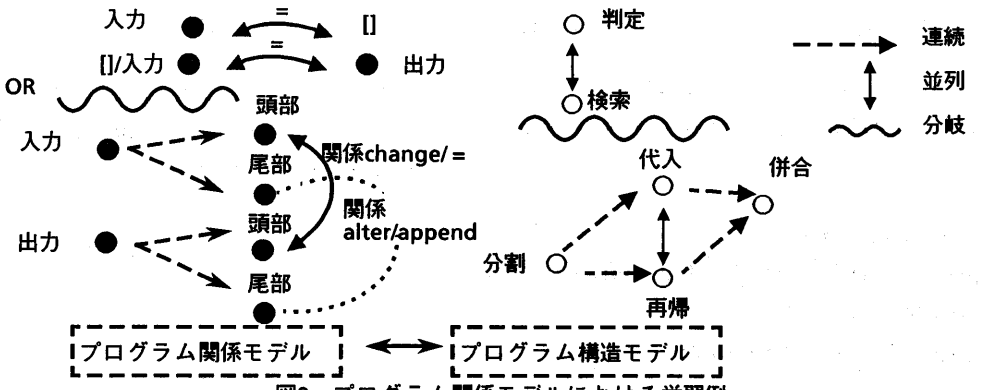

図9　プログラム関係モデルにおける学習例

5　まとめ

プログラムの類似性を4つのモデルを用いて発見し、類似プログラムの共通部分を抽象化して学習する過程のモデル化について述べた。本モデルの特徴は4つのサブモデルを導入していることである。以下にサブモデルの導入により得られた2つの結果を示す。

1) 現実関係モデルとProlog関係モデルの学習過程のモデル化により、プログラマによってデータ構造の割当て方が異なったり、それによって作成するプログラムが異なったりすることを説明できる。

2) プログラム関係モデルとプログラム構造モデルにより、現実関係モデルでの仕様が全く異なるプログラム(appendとalterなど)でもプログラマが学習するといった事実が説明できる。

現在、本モデル化手法について、さらに検討を進めており、計算機上でのモデル実現に着手している。

謝辞

本報告で述べたモデルの有効性を、実際のPrologプログラムを用いて検討して頂いた大阪大学工学部の岩田昌也氏に深謝します。

参考文献

[1]Carbonel, J. G.: Derivational Analogy: A Theory of Reconstructive Problem Solving and Expertise Acquisition, In: Michalski, R. S., Carbonell, J. G., and Mitchell, T. M. (Eds.), Machine Learning Vo.2: An Artificial Intelligence Approach (1986).

[2]Rosendahl, M., Mankwald, K. P.: Analysis of Programs by Reduction of their Structure, In: Goos, G. and Hartmains, J.(Eds.), Graph-Grammers and Their Application to Computer Science and Biology, Lecture Notes in Computer Science, Springer-Verlag, pp.409-417(1979).