プログラムの類似性定義のためのネットワーク表現

7L-3

大阪大学産業科学研究所
今中武　上原邦昭　豊田順一

1．はじめに

近年、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わ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、今後多様化する要求仕様に対し、予め全ての部品を準備することはほとんど不可能である。このような問題を解決する方法として、プログラム合成システムに学習メカニズムを組み込むことが考えられる。学習メカニズムを組み込めば、プログラム合成を行なうにつれて、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、読解していく間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、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。類似プログラムからの学習で問題となるのは、プログラムのどの部分を見て類似しているとみなすか、どのような情報を抽出すればよいかということである。以上のような考え方に基づいて、本稿ではプログラムの類似部分を発見する過程を計算機上で実現するために、プログラムのネットワーク表現を提案する。ネットワーク表現はプログラムの入出力仕様、機能、構造、動作などを詳細に表わしたものである。

2．プログラムの類似性

プログラムの類似性は、プログラムの仕様が類似している場合だけでなく、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 の関係を求めるプログラム

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

と、mother の関係を用いて grandmother の関係を求めるプログラム

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

は、プログラムの動作などを比較するまでもなく互いに類似していると判断することができる。これは2つのプログラムの入出力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、リスト中の特定の要素を調べる member プログラム

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

は、プログラムの入出力仕様、機能仕様に共通点が少なく、類似していると考えることはできない。しかし、再帰呼び出しに伴う入力リストの取扱いなどの点からはプログラムの構造が類似していると考えることができる。このように、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。

3．プログラムのネットワーク表現

プログラムのネットワーク表現は、大きく仕様と構造の2つの部分に分かれる。

・仕様

仕様は、プログラムの行なう仕事を人間が認識する形で表現したもので、どのような入力でどのような出力が得られるのか、ということを主に表している。たとえば、第1引数 X が第2引数 Y のリスト中に含まれているかどうかを調べる member(X,Y) のネットワーク表現では、第1引数が第2引数の要素と関係 '＝' で結ばれる。

・構造

プログラムの構造は、プログラムリストに沿った動作手順の流れを示したものである。たとえば、member プログラム（図1）では、処理1単位を示す処理ノード ◎ の並びとなっている。以下にネットワーク表現のノード、アークの属性の種類について主なものを述べる。

☆アークの属性

アークの属性について主なものをいくつかあげて説明する。has, ako, is などについては、他論文でも多く用いられているために、説明を省略する。

・in … データ入力が必要なノードから、入力データのノードを指し示す。たとえば、図1の最初の処理では、第1引数と第2引数の頭部とが等しいかどうか判定するために、第1引数と第2引数の頭部のノードから判定処理を行うノードに in アークがのびている。

・out … データ出力が必要なノードから、出力データのノードに向かうアークで、組み込み述語の出力などに用いる。

・eq … eq アークで結ばれた2つのノードは同一の属性を表わし、等価である。2つのノードは方向性を持たないので、アークの双方向に矢印を付ける。

・inc … inc アークで指し示されたノードは、もう1方のノードに含まれることを示す。たとえば図1では、第2引数はリストであり、アトムの要素を持ち、第1引数のアトムと等しいアトムを含むことを inc アークで表わしている。

・true(fail) … 処理ノード（◎）の間を結び、プログラムの動作が真(偽)である時の次の動作を指し示す。

・call … プログラムの呼び出しの時に、呼び出すプログラムノードを指し示す。たとえば、図1では再帰のノードが member プログラムノードと call アークで結ばれている。

・in1,in2, ... (out,hasについても同様) … in,out,has アークは、'減算' などの演算のノードが結びついていることがあり、結びついているもう一方のノード間で順序関係が解っていなければならない。そのときは in,out,has の後ろに数字をつけ、順序関係を定義する。

☆ノードの属性

・プログラムノード（●）… プログラム1単位を表わすノー

Network Representation for Definition of Similarity between Prolog Programs

Takeshi IMANAKA, Kuniaki UEHARA, Jun'ichi TOYODA

The Instituite of Scientific and Industrial Research, OSAKA Univ.

ドとして用いられ、● で表わす。ネットワークの起点であり、入出力変数を表わすノードと has アークで結ばれる。

・処理を表わすノード (◎) … プログラムの構造部で用い、プログラムの1動作または1述語を ◎ で表わす。このノードからは、必ず true または fail のアークが1本ずつ延び、同じ ◎ ノードと結ばれる。

・オペレータを表わすノード … 処理ノード ◎ と has アークで結ばれたノードに ako アークで結びつく。"判定","複写","代入","検索","再帰" などがある。このノードでの入出力データは in,out アークで他の引数と結ばれる。

・"すべて"の概念を表わすノード (破線丸) … "すべて" の概念が必要なとき、inc アークとともに用いる。例えば、入力された2つのリスト A,B を結合してリスト C に出力する append(A,B,C) のネットワーク表現で、「第3引数の要素は第1、第2引数の要素を "全て" 含む」という概念を表わすために波線丸が用いられる。

以上の書式にしたがって、具体的な Prolog プログラムのネットワーク表現を行なった。プログラムは文献[2]に掲載されているプログラムのうち、リスト処理を行なう50個のプログラムについてネットワーク表現を行なった。以下に member プログラムのネットワーク表現を示す。

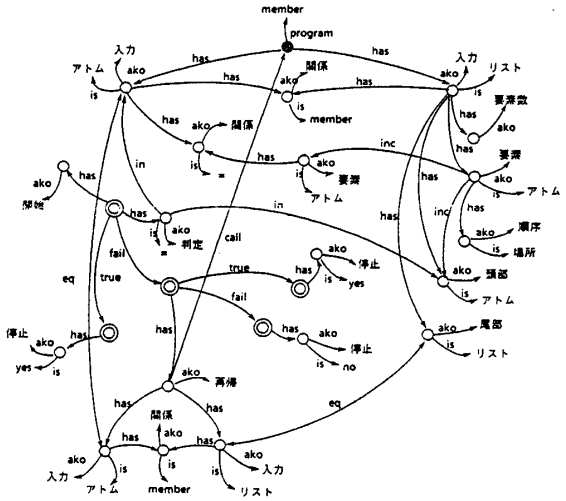

図1　プログラムのネットワーク表現

4 プログラムの類似部分の抽出

前章で述べた Prolog プログラムについてネットワーク表現を行ない、それらのプログラムから類似部分を手作業で抽出した。その中から具体例を示す。図2は、append, member プログラムの共通部分を抽出したものである。2つのプログラムは、再帰処理に伴う入力リストの扱いが類似していることが示されている。類似部分を抽出した結果を整理するため、図3のようにプログラムのグループ分けを行なった。図中では、プログラムに通し番号を、共有されている類似部分にアルファベットをそれぞれ付けている。この図から、多くのプログラムが多種の類似部分を共有していることが判る。

5 考察

プログラムのネットワーク表現で仕様を書く場合、人によって仕様部分のネットワーク表現が異なるという問題が生じる。たとえば、2つの入力リスト A,B を結合して出力リスト C を得る append(A,B,C) プログラムの仕様部分は、"A の要素のリスト中での位置は C でも等しく、B の要素のリスト中での位置は C では A の要素数だけ後ろになる" と "入力

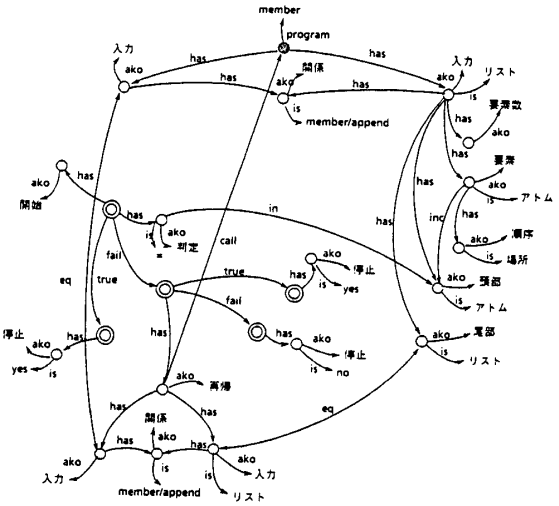

図2　プログラムの類似部分

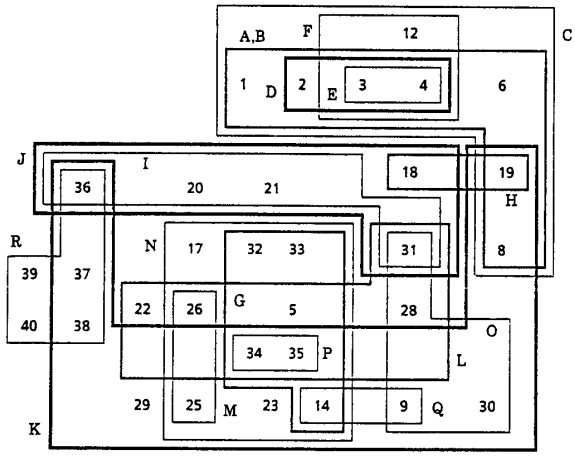

図3　類似部分によるプログラムのグループ分け

された2つのリスト A,B を結合してリスト C に出力する" という2つの書き方ができる。後者の仕様をネットワーク表現すると、'リストの結合' というノードを新たに用いなければならない。このように、新たなノードを無制限に用意すれば、あらゆる言葉に対して用意することになり、現実的ではない。我々は、ノードの種類を有限にするために、仕様のネットワーク表現を数学的な表現に限定している。

また、類似部分を抽出する際には、単純に共通部分を取り出したが、人間が学習する場合は、類似部分を様々な場合に当てはめることができるように抽象化して記憶している。したがって、類似部分を抽出した場合、類似部分それぞれを一般化することが重要となる。このようなプログラミング技術の学習過程をモデル化する試みについては[1]で述べている。

謝辞

ネットワーク表現を実際のプログラムに適用する際に手伝って頂いた本学工学部修士課程の岩田昌也氏に感謝します。

参考文献

[1]今中武, 上原邦昭, 豊田順一：類推、帰納の概念を用いたプログラミング知識の学習メカニズム, 情報処理学会, 知識工学と人工知能研究会報告, 59-13 (1988).

[2] W.F.Clocksin and C.S.Mellish: Programming in Prolog, Springer-Verlag Berlin Heidelberg (1981).