3W-3

意味記憶システムＩＸ
－知識表現言語ＩＸＬのルール・コンパイラの開発－

樋口哲也、古谷立美、半田剣一、楠本博之、国分明男
（電総研）

１．　はじめに

意味ネットワークは基本的には宣言的知識の図的表現であり、手続き的知識の取り扱いには別の枠組みを用意する必要がある。意味ネットワークに基づく知識表現言語ＩＸＬでは、述語論理の節形式を用いている。

ＩＸＬで記述した手続き処理は、知識ベースを格納する意味ネットワークマシンＩＸＭ上で実行するが、そのためには節形式で書かれた手続き的知識を解釈して、ＩＸＭマシン上で並列処理可能な命令列を生成しなくてはならない。

この手続き的知識表現からＩＸＭマシン命令列への変換をここではＩＸＬルールコンパイラと呼ぶ。数値演算を含む基本的な手続き処理を実現するためのＩＸＬルールコンパイラが完成したので本稿ではその概要について説明する。

２．　ＩＸＬでの手続き表現

ＩＸＬは意味ネットワーク処理用の述語をPrologに付加したPrologのスーパーセットである。

ＩＸＬには、ＩＸＬコマンドと呼ぶ宣言的知識に対応する記述単位がある。ＩＸＬの手続き的知識は、このＩＸＬコマンドを節形式の中に書くことによって表現できる。

述語論理で祖父を求めるためのルールはたとえば次のように書ける：

祖父(X, Z):- 父(X, Y), 父(Y, Z).

これをＩＸＬで書くと

```
asst(grand_father, X, Z):-
 asst(father, X, Y), asst(father, Y, Z). (1)
```

となる。ここでasst(father, X, Y)は一つのＩＸＬコマンドであり、宣言的知識の記述に対応している。このようにＩＸＬでは手続き的知識のルールが、節形式のボディにＩＸＬコマンドを記述することにより表現される。

いま［太郎の祖父は誰か？］という質問が来たとすると、この質問は

asst(grand_father, 太郎, Z)　となり、（１）のルールが起動される。そして、このルールは概略的に言えば

asst(father, 太郎, Y)

asst(father, Y, Z)

の二つのＩＸＬコマンドの実行により、Ｚの解が求まる。

ＩＸＬの手続き実行はProlog処理系上で現在サポートされているが、これはあくまでも逐次処理であり、最終的には解の全解探索を行うＩＸＭマシンでの並列処理を目標としている。

しかしその並列処理のためには上記二つのＩＸＬコマンドだけでなく、全解探索を行うためのＩＸＭマシン用の命令列も生成する必要があり、それがＩＸＬルールコンパイラの主たる目的となっている。

３．　ＩＸＭマシン上での手続き処理

ここではまずＩＸＭマシン上でのＩＸＬコマンドの実行方法について述べたあと、それを含む、一つのルール全体の実行方法について述べる。

3.1　ＩＸＭマシン上でのＩＸＬコマンドの実行

ＩＸＭマシン上でのＩＸＬコマンドの処理はマーカビットに基づいている。意味ネットワークの各ノードは、ＩＸＭマシン上でそれぞれマーカビットと呼ぶ１ビットのフラグを複数個持っており検索の途中結果を保持する。これを用いてある条件を満たすノード群を識別できたりする。ノードは具体的には連想メモリ中のワードに保持され、マーカビットはそのワード内の一領域を占める。

(1)のルールのボディの第一番目のＩＸＬコマンドが　asst(father, 太郎, Y)　として実行される場合、解となる変数Ｙの具体例のノード、つまり太郎のノードから fatherのリンクをたどって到達できるノードは、その識別のために特定のマーカビットをセットする。たとえばＹに当たるノードのマーカビット１番をセットする。

(1)の２番目のＩＸＬコマンドであるasst(father, Y, Z)は、最初のＩＸＬコマンドで求めたＹをもとにして、Ｚに当たるノードを求めなくてはならない。これはＹが共通変数として現れているからである。（直観的にいえばＹは太郎の父であるから、太郎の祖父ＺはＹの父として検索する）。求めたＺのノードには例えばマーカビットの２番をセットして、これを識別する。

この例からわかるようにＩＸＬルールコンパイラの最初の役割は、(1)のような手続きルールを解釈してＩＸＬコマンドをボディから抽出し、解の探索のために変数にマーカビット番号を割り当

Rule Compiler for the Knowledge Representation Language IXL
Tetsuya HIGUCHI
Electrotechnical Laboratory

ててＩＸＬコマンドのアーギュメントとすることである。

3.2 手続き処理用のＩＸＭマシン命令の生成

(1)の手続きルールは、ＩＸＭマシン上での実行結果として、Ｚに対応するノードと太郎のノードとの間に［祖父］のリンクを張ることで完了する。従って上記二つのＩＸＬコマンドだけでなく、リンク張りの命令やリンク張りの相手の認定のための命令もＩＸＭマシン上で実行する必要があり、ＩＸＬルールコンパイラはそれら(1)のルールには直接現れてはいない命令を生成する必要がある。ここでは、そういった命令の必要性を説明するため、具体例をあげる。

(1)のルール実行は、意味ネットワークの面からみると、図１のような［解のパターン］を意味ネットワーク上で探索し、太郎のノードと解との間に新たなリンクを張ることに対応する。解のパターンの検索は、太郎のノードから父のリンクを逆たどりして最後にＺのノードに至るが、祖父のリンクを張るためにはＺのノードにおいて太郎のリンクのＩＤがわかっていなければならない。従ってリンクをたどる間に太郎のＩＤ情報も一緒に送る命令を生成する必要がある。このリンクたどり用の命令群をtree-traverse命令と呼ぶ。

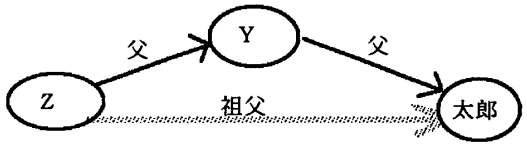

図１　解のパターン

(1)のルールをコンパイルして得られるＩＸＭマシン実行用の命令列は図２のとおりである。

```
initial(太郎、1)
asst2(father, 1, 3)
asst2(father, 3, 2)
transmit(1, father, bwd, 3)
relay(3, father, bwd, 2)
connect(1, grand_father, 2)
```

図２　(1)のルールのコンパイル結果

命令中に現れる文字列は、実際にはＩＸＭマシン上でそのノードが割り付けられる物理アドレスとなる。

initial命令は探索の起点となる太郎のノードを見つけ、そのノードのマーカビット１番をセットする命令である。続く二つのasst2は(1)のルール内の二つのＩＸＬコマンドに対応する。transmit, relayはいずれもtree-traverse命令である。transmit(1, father, bwd, 3)では、マーカビット１番の立っているノード（つまり太郎のノード）から、そのＩＤを、fatherリンクを逆（bwd;backwardの意）たどりして到達するノード（ここにはマーカビットの３番が立っていなくてはならない）に送る。この命令が実行された段階で図１のＹのノードに太郎のＩＤが到達している。次に実行する relay(3, father, bwd, 2)命令ではやはりfatherリンクを逆たどりして太郎のＩＤを次（この場合はＺ）に中継する。最後のconnect(1, grand_father, 2)命令では、送られて来た太郎のＩＤとマーカビットの２番が立つノードとの間に祖父のルールを張る。これは具体的には新らしい連想メモリのワードの登録となる。

(1)のルールがasst(grand_father, 太郎, Z)として呼ばれるときは以上のとおりであるが、並列度は低い。しかしasst(grand_father, X, Z)として呼ぶときは意味ネットワーク内のすべての祖父関係を見つけだす。このときのコンパイルコードは図２からinitial命令を取り除いたものにほぼ等しい。各命令が意味ネットワーク内の対象すべてに対してＳＩＭＤ的に実行されるので、このときの並列性は対象が多いだけかなり高くなる。

4．　数値演算

ＩＸＬのルールには数値演算も記述でき、ＩＸＭマシンでの実行コードが生成される。演算の種類は基本的な加減乗除、比較演算に今のところ限っている。演算は意味ネットワークの各部分で並列実行可能であるが、３で述べたリンク張りのときと同様、演算のオペランドの認定のためにtree-traverse命令も生成する必要がある。

5．　おわりに

以上のようにＩＸＬルールコンパイラの処理には、ルールボディ部からのＩＸＬコマンドの抽出の他に、ノード間での新たなリンク生成、そのための経路情報の抽出、数値演算のための命令生成などが含まれている。

現在Prologで記述したＩＸＬルールコンパイラの第１版が完成している。今後の課題はＩＸＭマシン上での実動化であり、tree-traverse命令やconnect命令など実際に意味ネットワークをたどってリンクを張る命令群の効率的な実装である。

また手続きの中でまた別の手続きを呼んだとき（再帰呼び出しも含む）のマーカビットの管理も重要な問題であり、演繹データベースを意味ネットワークで実現する際の課題となる。

末筆ながら本研究の機会を与えられた柏木電総研所長、御指導頂く棟上情報アーキテクチャ部長、管理工学研究所の小島氏、大林氏に感謝する。