知識工学と人工知能　45－4
（1986・3・10）

日本語文によるLISP関数の自動生成

池谷幸喜、重永　実　（山梨大学　工学部）

1　はじめに

自動プログラミングは、ソフトウェア生産の重要な手段として実用化されてきている。しかし、人間の思考としてのプログラミングにおいて重要となる問題解析（抽象的な問題表現をプログラミング領域の問題に解釈し直す行為）は十分に扱われてこなかった。これはソフトウェア生産の完全自動化を阻んでいる一つの原因となっている。そこで、問題解析機能を組み入れることにより、ユーザへの依存を減らし、自動プログラミングを完全なものに近づけることができる。問題解析は、与えられた（自然言語で記述された）問題をプログラミング領域に依存した意味表現に解釈するという行為と見ることができる。したがって、適切な問題の意味表現の構造を決めることが重要な問題となる。

一般に、プログラムの仕様記述（機能記述）は自然言語文で書かれる。記述は各ステップ毎を自然言語文で書き換える低レベル（具体的）な表現方法から、入出力とその関係を述べた高レベル（抽象的）なものまで様々である。また、プログラム生成（ソフトウェア生産）では、どのモジュールが使用できるのか、どのようなプログラム制御構造が適当であるのかを自動的に決定することが重要な問題である。低レベルのプログラム仕様（実際のプログラムに近い表現）ならば、決定の自動化は容易である。しかし、抽象的な高レベルの仕様記述では、モジュール、制御構造の決定は既に述べたように仕様の意味解釈という知的な行為が必要になってくるため、問題はそれほど簡単ではない。いい換えれば、抽象的な仕様記述を如何にどれだけ具体的な（プログラムに近い）表現に翻訳できるかという問題になる。仕様記述言語、述語論理式等による仕様記述や、人間と機械との対話(dialogue)から引き出された抽象的なプログラム表現（プログラムネット）を目的言語で書かれたプログラムに変換するという方法等が試され、実用化されてきている[4]。

本研究では、LISP入門書[2],[3] 程度の問題を日本語文で記述し、その文章を構文解析、意味解析して適切な意味表現を作り出し、その意味表現をLISP言語で書かれたプログラムに変換することを試みている。

2　システムの概要

本システムは自然言語処理部とプログラム生成部から成る。自然言語処理部は、入力された日本語文を構文解析、意味解析して、格構造表現された意味ネット（Semantic Network）を生成する。そして生成された意味ネットはプログラム生成部によって、LISP1.9仕様のプログラムに変換される。両部に利用される知識は、日本語に関する文法（構文）情報・各単語の意味情報と（LISP）プログラミングに関する知識との二つに大別される。

2．1　構文解析[10]

ローマ字によって分かち書きされた入力文を句単位（ほぼ文節と同じ）に解析し（句表現の作成）、各句間の係り受け関係（構文木）を求める。一つの文に対して複数個の句表現の可能性があるが、意味的整合性が得られるもののみに限定される。また、一つの句表現からも複数個の係り受け関係が得られるが、係り受け距離が最短の構文木のみを採用する。

２．２　意味解析[6][7][8][9][10]

2.1 で得られた全ての句について、句の性質を解析し、構文木に表わされた係り受け関係を処理して各句を意味的に結びつけ、格構造表現された意味ネットを生成する。意味的に等価で表現が異なる文も、統一された意味ネット（等価な意味表現）に変換される。統一された意味ネットは名詞ノードと動詞ノードのみで構成され、特に動詞ノードは主動詞として意味ネットの根のノードのみに現われる。これは文中の［動詞→名詞］修飾を［名詞→名詞］修飾に変換しているためである。例えば次の三つの節は何れも、［リスト］ノードが［数値］ノードに［ELEMENT （要素)]アークによって繋げられた意味ネットとなる。

（1）要素が数値であるリスト　（2）数値からなるリスト　（3）数値のリスト

係り受けの処理をした結果は、格構造表現で句番号をアイデンティファイヤーにした属性リストを使って記憶されている。しかし、この形では意味を部分的に捉えたり、連続した節や句を任意に取り出すことが困難である。そこで、連想リスト（リスト表現）に変換し、このリストを意味解析の結果とする。意味ネットは、句番号に対応するノード（Ｎｎ）と修飾関係を表わすアークからなり、ノード番号がヘッドの連想リストである。

２．３　プログラム生成

自然言語で記述されたプログラムモジュールの仕様の意味ネットには、次のような形態的な特性がある。

◎　モジュールの入出力に対応する単語は意味ネット上の特定の箇所に現われる。出力は主動詞の格（特に対象格）に、入力は主動詞の別の格または最終修飾語に対応する。

動詞のSemantic Marker によって出力に対応する格を、さらに意味ネットを辿って最終ノード（葉）を見つけることにより入力を決定できる。複数の制御構造やモジュールによって合成できるプログラムの意味ネット上での結合状態は、図2.1 に示すように、夫々の制御構造やモジュール間の関係と階層的に対応する。左の図はモジュールの鋳型（一つの機能を表わした意味ネット）毎に意味ネットを３つ(SN1,SN2,SN3) に分けたものである。MVは主動詞をNnは名詞を表わすノードである。MVから出ているアークTA1 は動詞の対象格を表わし、名詞ノード間のアークは名詞間の修飾関係を表わす。右の図はプログラムの構造を鋳型ごとに分けたもので、ST1,ST2,ST3 は夫々鋳型SN1、SN2、SN3 に対応するプログラム断片である。意味ネットの鋳型の階層関係とプログラムの階層関係に注意されたい（3.4 節参照）。この特性を使って、与えられた問題に必要なモジュールや制御構造を取り出し、夫々を合成してプログラムを生成していく。

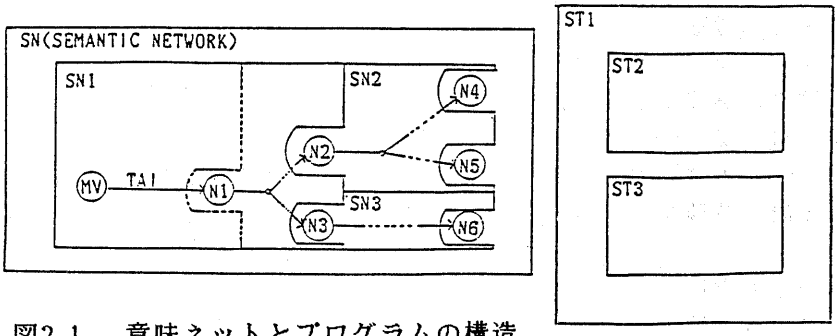


図2.1　意味ネットとプログラムの構造

プログラム生成のための規則はプロダクション・ルールのかたちで記述されており、生成は順次ルールを適用してゆくことにより自動的に行なわれる[1] 。次にその型を示す。

(1) [SN] → [PROGRAM]　(2) [SN] → [PROGRAM] + [sub-SN] ＊
(3) [SN] → [sub-SN] ＊　(4) [SN] → [MODIFY PROGRAM]　（＊：一回以上可）

[SN]は、マッチングを取るべき意味ネット（基本関数、プログラムモジュール、制御構造が持つ仕様文の意味ネット：鋳型）と各ノードやアークに関する条件を持つ。[PROGRAM] は、LISP1.9 の基本関数、プログラムモジュール（細分化できないプログラム）、特殊な制御構造を知識に持ち、プログラムを生成する。(1) のルールでは、意味ネットは直接プログラムに変換される。入力文の意味ネットは鋳型に完全にマッチングするか、または部分的にマッチングすればよい。後者の場合、マッチングしない部分は得られたプログラムの入力部を表わす意味ネットとして再び試される。すなわち、プログラム本体とその入力のコーディングをする。(2) では、基本的には(1) のルールと同じであるが、(1) の持つ行為の他に、生成に必要なプログラム要素を求めるための下位の意味ネット（[sub-SN] ）を設定する。(3) では、意味ネットは幾つかの下位意味ネットの組み合わせとして表現されたり、異なった問題に対応する意味ネットに変換される。(4) はプログラムの生成はせず、現在生成中のプログラムの部分的な修正指示を与える。(1) ～(4) 以外にも、妥当なデータ構造の設定、割付けを行なうルール、特殊な名詞ノード（特定の計算式や定義を持つ名詞のノード）を処理するためのルールなどがある。これらのルールは(1) →(2) →(4) →(3) の順に適用されていく。これは、与えられた問題を既存の基本関数やモジュールで合成し、無駄な意味ネットの展開や変換を防ぐためである。各ルールの要素は全てLISP関数によって構成されている。ルール記述用の各関数は作業用領域(WM)から必要な情報を取り出し、固有の作業をして、WMの内容を更新する。WMは３つの領域（WM1,WM2,WM3 ）から成る（4.1 参照）。

２．４　知識源

知識源には、不変的に使われる知識（長期記憶）、文章処理とプログラム生成の過程で得られる情報（中期、短期記憶）がある。不変的知識は、システムに予め用意されたおり、(1) 構文情報、意味情報を記述した品詞別単語辞書、(2)活用語の語尾変化とその意味を活用語尾ごとに記述した活用語語尾辞書、(3)品詞間の隣接関係を表わした構文状態遷移ネットワーク、(4)名詞句間の係り受けに対して、その処理方法を記述した辞書、(5)動詞の意味(Semantic Marker) 別にその処理方法を記述した辞書、(6)構文解析、意味解析した結果をプログラムに変換するための知識がある。また、処理過程で生じた情報は、システム自身が作り出し利用する。意味解析の結果（各文の意味ネット）が中期記憶に入り、短期記憶にはプログラムの生成で得られる様々な情報が逐次追加、変更、消去されてゆく。

３　プログラムモジュールの仕様と意味ネット

一般にプログラムの仕様書（機能記述）は（動詞／目的語型の）自然言語文で書くことができ、ここでは日本語文で記述する。この仕様は最も高レベルの抽象的表現、すなわち入出力の関係を表わした記述である。LISP基本関数やプログラムモジュールの機能を意味ネットで表現した結果、その意味ネットには次のような特性があることがわかった。

◎　モジュールの入出力に対応する単語（入出力を代表する名詞）は意味ネット上で特定の箇所に現われる。すなわち、出力は主動詞の格（特に対象格）、入力はその格の最終修飾語または別の格さらにはその格の最終修飾語に対応する。

３．１　仕様文の意味ネット

特定の動詞（取り除く、加えるなど）を除いて、必ず対象格がモジュールの出力に対応する。このことを示したのが図3.1 である。この例では入力は対象格の最終修飾語に対応

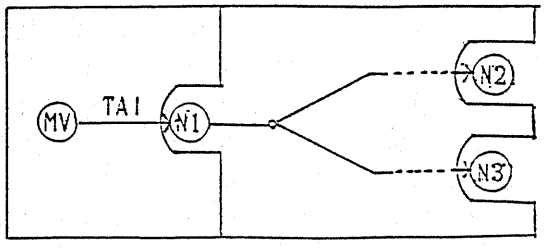
している。すると、主動詞ノード（MV）と対象格アーク(TAI）を取り除いた意味ネット（  ）を、モジュールの入出力を明確にした仕様の意味ネットと見ることができる。この図では、凸の部分が出力を、凹の部分が入力を意味する。このことは、主動詞MVはモジュールの仕様においては補助的な役割しか果たさないことを意味する。

図3.1 仕様文の意味ネット

対象格以外に別の格をもつ動詞では、個々に特別な解釈が必要である。例えば、動詞”取り出す”は対象格と源泉格をもち(”～から～を取り出す”)、夫々出力と入力に対応する。仕様文の主動詞にこの動詞が使われている場合には、次の２通りの解釈が可能である。

1.源泉格に現われる単語を対象語の所有者とすることで、対象格のみをもつ動詞に換えることができる。

2.仕様の意味ネットの一つの型に加える（動詞を省略しない意味ネットとする）。

本システム内では両方の解釈を採用しており、その動詞が使われる状況に応じてどちらか一方に決定する。

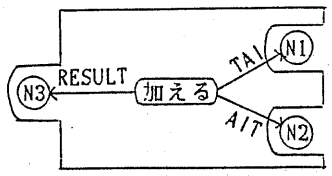
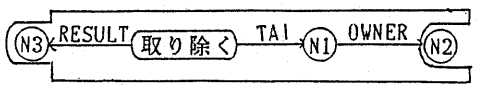


図3.2 動詞”加える”の意味ネット

1.



2.

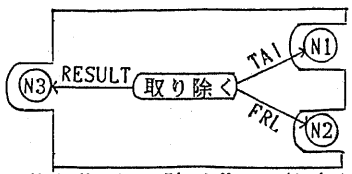


図3.3 動詞”取り除く”の意味ネット

また、出力に相当する格を持たない動詞や”取り出す”と同じ格構造を持ち、且つどの格も出力を示さない動詞は適当な解釈を加えることにより  型の意味ネットになる。

結局、仕様文の意味ネットは入力と出力が明確にわかる  型に統一できる。このことは、意味ネットから既存の基本関数やモジュールさらには特殊な制御構造を使って、要求され

たプログラムを階層的(top-down)に生成できることの強い裏付けとなった。

３．２　副作用を目的とするモジュール（または関数）

値を返すこと（出力）が本来の働きであるが、副作用(side effect)を目的とするモジュールがある。変数への代入や置き換えを行なうモジュール（または関数）がその例である。このようなモジュールの仕様文の意味ネットでは出力に相当する格はなく、対象格が入力を示す。そこで仕様を次のように変え、入力ー出力を持ったモジュールに強制的に書き換える。◎　代入あるいは置き換えた値を出力する。

３．３　（特殊な）制御構造の仕様

ここで言う制御構造とは、構造自身に特別な意味を持った制御構造である。そこで、制御構造は夫々に特別な仕様を持っていると考え、その仕様も意味ネットを用いて表現する。制御構造の仕様の意味ネットも、基本関数やモジュールと同様な形態的な特徴を有している。すなわち、◇型の意味ネットとなり、入出力部が明確に示される。しかし、入力部が他の制御構造の出力部に組み込まれたり、または出力部に他の制御構造の入力部が組み込んだりする。これはモジュール結合のうちデータ結合[5] と呼ばれるものの一種である（入出力のデータ型が同じ）。制御構造には、前者の性質を持つものと後者の性質を持つものがある。制御構造と他の制御構造やモジュールとの合成にはデータ結合を用いる。

３．４　基本関数、モジュール、制御構造を使ったプログラム合成

LISPの基本関数はLISPプログラムのプリミティブであり、モジュールや制御構造を組み立てる。ここでは、プログラム生成の効率化を図って、必要欠くべからざるモジュールや制御構造はプログラムの最小単位（プリミティブ）として取り扱う。基本関数、モジュール、制御構造の仕様は、3.1 ～3.3 節で説明したように、◇型の意味ネット（今後、鋳型と呼ぶ）で表現できる。基本的には問題文の意味ネットから適当な鋳型を取り出し、それらを逆に合成して、プログラム生成を行なう。すなわち、◇型の意味ネット（鋳型）を問題文の意味ネットにはめ込みながら（根節すなわち主動詞ノードからマッチングさせながら）、適切な関数、モジュール、さらには制御構造を決定しプログラムを生成していく。

次に、この基本的な合成法がモジュールや制御構造の合成にも適用できることを説明する。種種の問題文の意味ネットとそのプログラムの構造との関係を解析した結果、意味ネットープログラム間に次のような特性（図2.1 参照）があることが分かった。

(a) 鋳型が問題文の意味ネットの上位（主動詞に近い部分）にある制御構造はプログラムの上位の構造となる。

(b) 連結した２つの意味ネット間で、出力＝入力関係で結合された二つの制御構造は、一方の制御構造が他方を組み入れた入れ子型の構造になる。

(c) 上位の制御構造に出力＝入力関係で結合された複数の制御構造は、結合順に並列に並んだ構造になる。

モジュール間の合成は、関数合成と同様に、夫々の入力ー出力を結合させながら行なう。モジュールは、入力部（＊：一回以上繰り返し可）・入力→出力を実行する計算部・出力部からなる。入力部は計算に必要な入力を返す他のモジュールや（基本）関数、さらには出力を持つ制御構造（3.3 節参照）から構成される。問題文の意味ネットが、このモジュールの鋳型で終わる場合には、入力部は各入力を代表する変数（最終的に関数の引数部の局所変数）に置き換わる。計算部は出力を計算するプログラムであり、入力部と同様な要素で構成される。出力部はモジュールの型に合わせて、計算された値を返す基本関数(RETURN,EXIT-LOOP,EXIT-FOR等）で表現される。

一方、制御構造間の合成はデータ結合を使用しなければならない。制御構造はその構造

によって２種類に分けられる。

(a) 出力部はモジュールのように働く（出力値を返す）が、入力ー計算部がデータ結合を必要とするもの。

(b) 入力部はモジュール同様各要素から構成されるが、出力部がデータ結合を必要とするもの。

４　プログラム生成

問題文の意味ネットからプログラム断片（LISPの基本関数、モジュール、特別な制御構造）の鋳型（意味ネット）を階層的に取り出し、各断片を再合成すればプログラムを生成できることを示した。これは、問題文の意味ネットが鋳型の合成体（鋳型が入力ー出力によって繋がれたもの）であるときのみ有効である。言い換えれば、意味ネット内に必要なプログラム断片の鋳型が明確に現われている場合のみ可能である。問題文の意味ネットが常にこのような形態であるなら、プログラム変換規則は、2.3 節で示した[SN]→[PROGRAM] 型の規則だけで十分である。しかし、明確に表現された問題文は少なく、抽象的なまたは暗黙的な言い回わしから適切なプログラム制御構造や仕様可能なモジュールを導出しなければならない。また、bottom-up 生成はプログラムが階層的に決定できない場合、例えば問題文の意味ネットの中で修飾語がプログラムの上位の制御構造になるような時（後述）に使われる。

上位のプログラム構造が決定されると、次はその入力部の生成に移る（関数では引数に当たる）。これは、プログラム生成における一般的な手続きであり、変換規則の中にimplicitに現われる。

４．１　作業用領域(Working Memory)

プログラム変換規則はプロダクションルールで記述されており、プログラム生成は各ルールを評価することで行なわれると述べた。ルールの各要素は意味情報（プログラム生成に必要な知識）を引数に持つLISP関数で記述されている。プログラム生成過程で作り出される情報は各関数が参照できるように、共通領域に入っている。この領域を作業用領域(WM)とよぶ。作業用領域は３つに分割されており(WM1,WM2,WM3) 、夫々の領域には生成過程で生じる情報が用途別に記憶されていく。

(1) ＷＭ１　主に意味ネットに関する情報を蓄える。ＷＭ１にはマッチング（探索）すべき問題文の意味ネット(SN)、探索開始ノード番号(PS)、鋳型と意味ネットとの対応表(CSM、ノードのLEAF(S) またはROOT(F) の別、探索し残した意味ネット）が入る。

(2) ＷＭ２　プログラム生成に必要な情報を蓄える。プログラムの階層番号(LV)、プログラム構造の形態(SYL、実行型(AS,ES) 、制御型(CS)) 、生成中のプログラム断片(PP)、局所変数リスト(L-VARS)が入る。

(3) ＷＭ３　プログラム生成を制御するための情報を蓄える。上位構造と下位構造との結合情報(SA 、結合情報の他にプログラムの修正、追加指示も含まれる) 、作業領域番号(WMN,UWMN ：スタック番号) 、ルール番号(RN,適用された変換規則の番号) 、プログラム断片と意味ネットとの対応表(CSP、階層番号と入力と出力に対応するノード番号の組) 、入力対応表(CI 、変数名またはプログラム構造番号と対応する単語名の組、また変数に値が束縛されている時にはその値も入る) 、出力対応表(CO)が入る。入力対応表は、完成されたプログラム（モジュール）を本体に持つ関数の引数を決定するのに使われる。

プログラム生成は、作業領域を初期化して始められる。作業領域リストには、作業領域（WM1,WM2,WM3 のリスト）がルールが適用されていくごとに積まれていく。

４．２　プログラム変換規則(SYN:RULES)

図4.1 にルールの記述形式を示す。LHS は条件部を、RHS は実行部を示している。各要素はLISP関数で記述され、ルールの適用はLHS 部の各関数（関数が条件を示す）を評価し全て真であるとき（全条件が満たされた場合）、RHS 部の関数が評価されて行なわれる。

```
（ＲＵＬＥｉ　　（ＬＨＳ　　（@関数名　引数１．．．引数ｍ）
                              （@関数名　引数１．．．引数ｎ）
                                        ⋮
                  ）
                  （ＲＨＳ　　（@関数名　引数１．．．引数ｋ）
                                        ⋮
）                ）
```

図4.1　辞書SYN:RULES のルール記述形式

４．３　変換規則の適用

各ルールは記入の優先順位を持ち、それに従って辞書SYN:RULES に入っている。したがって、ルールの適用はその順序に従う。優先順位は、2.3 節でも述べたように、(A) →(B) →(D) →(C) である。同じ型のルール群の中にも優先順位があり、同じ条件を含んでいる場合には、条件数の多い方を優先させる。また、生成を制御するためのルールは、(A) ～(D) のルールが不適当な時のみ評価されればよく、したがって(C) の後に置かれる。

ルールは記入順に先頭から試されてゆく。全てのルールが不適であれば、ユーザに表現を変えて新しい問題文を入力し直してもらう。一方、あるルールが条件が満たされ適用された後は、再び先頭のルールから評価を始める。

以上はプログラム変換規則適用のための制御方法である。したがって、システムのプログラム生成を行なう部分は、プロダクション・ルールを制御するPSI(Production System Interpreter)と見ることができる。

４．４　異なった意味解釈による制御構造の違い

ここでは、対象となっている用語も修飾語も等しいにもかかわらず、その用語の意味ネットでの位置によって、プログラムの制御構造が異なってくる場合を説明する。

例１　数値リストからなるリストの各要素に含まれる数値の和を求める。

例２　数値リストからなるリストの各要素に含まれる数値の和のリストを作る。

例１は、リストに含まれる全ての数値の和を求める（入力：(1 2 3) (4 5 6))、出力：21）と解釈でき、一方例２はリストに含まれる数値リストごとに数値の和を求めリストにしていく（入力：((1 2 3) (4 5 6))、出力：(6 15)）と解釈できる。この違いは、名詞”和”が動詞の対象格に現われる（直接和を求める）のか、または他の名詞の修飾語として現われる（上位プログラムの入力になる）のかによる。例１の場合、”～の和”を求める構造は”～に含まれる数値”を求める構造を入れ子にする。ところが、例２の場合、”～の和”を求める構造が”リストに含まれる数値リスト”を取り出す構造の入れ子（数値リストを取り出す度にそこに含まれる数値の和を求める構造）になる。”和”は（作るべき）リストの要素となることにより、複数個存在する可能性があることから夫々の数値リストに含まれる数値の和であると解釈できる（”～和の積を求める”でも同様である）。

したがって、対象となっている用語の出現箇所（動詞の格あるいは他の修飾語）によってルールを分けて作る必要がある。

５　おわりに

本研究は、プログラムの機能を与える問題文の適切な意味表現の決定とその意味表現をプログラムに変換する規則の構築にあった。

意味表現は、問題文に含まれる各用語の意味関係が明確に現われ、入力－出力関係が適切に存在する表現、すなわち意味ネットが使われた。意味ネットは動詞とその格を中心に修飾語をその周りに張り巡らした構造であり、その構造はプログラムの構造と階層性において非常によく一致する。また、一般にプログラムの仕様を記述するのに動詞／目的語型（格表現構造）の自然言語文が使われることから、適切な意味表現と言える。しかし、与えられた問題の解析、すなわち問題解析が今後プログラムの自動生成において必要不可欠な要素になると考えられることから、現在の意味ネットでは意味表現が不十分である。言い換えれば、問題の全体的な意味を捉えることができない（bottom-up なプログラム生成が困難になる）。そこで、プログラム構造との相関性を考慮しながら（意味ネットの基本的な特長が失われないように）、意味ネットによる意味表現を改良していかなければならない。

プログラム変換規則は、意味ネットに依存した構成になっており、意味関係の記述に従って組み立てられていく。また、プログラム（関数、モジュール、特殊な制御構造）の機能（仕様）は意味ネットで記述し、与えられた問題文と基本的には直接に結合されていて、意味ネットの構造に従ってプログラムに変換される。したがって、階層的なプログラム生成を行なう規則は容易に作れ、その構造も比較的単純である。しかし、bottom-up 生成を行なう規則を作ることはその反面非常に困難になる。また、bottom-up な生成法は、より複雑なプログラムやプログラムの効率化を図ってゆくのに必要なものである。そこで、bottom-up 生成を円滑にかつ無駄なく行なうために、プログラムの機能のみでなく機能実行の論理を表わすことができるプログラム記述（各ステートメント間の関係、制御構造、データ構造を明確に表わしたプログラムのネット）を考えなければならない。すなわち、問題文の意味ネットとプログラムとの間に、別のプログラム記述を作り、意味ネットはそのプログラム記述へ変換して、さらにそれを目的の言語によるプログラムに変換することを考える必要がある。これにより、プログラム変換規則は、意味ネットに依存しない形で作ることが可能になる。また、規則の一般化（チャンク化）も容易に行なえる。

ここで使われた意味ネットとプログラム変換規則は、プログラムの階層的な自動生成という目的に対しては十分に対処できる。より高級な問題解析やbottom-up 生成を織り込んだより完全な自動プログラミングシステムを完成するためには、上記の問題点を解決していかなければならない。

参考文献

[1] John R.Anderson 、Robert Farrel 、Ron Sauers：”Learning to Program in LISP” Carnegie-Mellon University (1984). [2] P.H.Winston 、B.K.P Horn:”LISP”(1981). [白井良明、安部憲広訳：LISP、培風館 (1982)]. [3] 黒川利明：” LISP入門” 、培風館 (1982). [4] E.Feingenbaum：” Handbook of Artificial Intelligence ” (1984). [5] Glenford J.Myers ：”COMPOSITE/STRUCTURED DESIGN” (1978).[国友義久、伊藤武夫訳:「ソフトウェアの複合／構造化計画」,近代科学社 (1979). [6]日栄編集所:”要解口語文法”、日栄社 (1971). [7]久野暲：” 日本語文法研究” 、大修館書店 (1973). [8]草薙裕、南不二雄、中野洋、吉田夏彦：文法と意味Ⅱ、朝倉日本語新講座４(1985). [9] 杉野憲作：” 音声理解システムのためのMOPsの利用による名詞予測に関する研究” 、山梨大学計算機科学科修士論文(1983). [10]池谷幸喜：日本語によるLISPプログラミング、山梨大学計算機科学科卒業論文(1984).