EC84－25

知識工学的手法を用いた図形入出力プログラム作成支援システムの構築法について

A Method of Constructing Knowledge Based Programming Assistance System for Graphics Input-Output Programs

吉田　敦　　田中　英彦　　元岡　達
A. Yoshida , H. Tanaka , T. Moto-oka
東京大学　工学部
Faculty of Engineering , University of Tokyo

1. はじめに

問題解決に必要な種々の知識を、知識ベースに格納し、推論機構により、これらの知識を利用して、特定の問題領域における種々の問題解決を行なうシステムを、知識ベースシステムという。このようなシステムとしては、エキスパート・システムと呼ばれるものがある。これらのシステムは、プロダクションルール、あるいはフレームと呼ばれる形式で、特定の応用領域に関する知識を持つもので、これまでに、医療診断や化学構造分析、地質構造分析等を対象領域とするものが試作されてきた。[1], [2]

最近の動向として、これらのシステムを、新たな応用分野、例えば、計算機システムの配置の決定や、論理回路のCADなどにも用いようとする研究が行なわれている。しかし、これらのシステムの構築に対しては、いまだ確立された方法論がなく、それを確立するための研究もほとんど行なわれていない。あえて言うなら、知識ベース・システムの構築には、行きあたりばったりの方法が取られているのが現状である。

そこで、本研究では、知識ベースシステム構築の方法論を検討するため、問題領域を、簡単な二次元図形入出力プログラムの作成支援とした知識ベースシステムの試作を行なってみた。ここで試作されたシステムは、プログラム作成支援システムとしては必ずしも、完璧なものとは言えないが、このシステムの試作を通して、知識ベースシステムの構築法について検討を行なってみた。

ここでは、第2章で、プログラム作成のモデリングを、第3章で、試作したプログラム作成支援システムの構成、第4章でこのシステムの動作とその例、第5章で考察、検討事項を述べる。

2. プログラミングに対するモデル

特定の問題解決システムを構築する際に、まず、その問題に対するモデルを構築することが必要なことは、言うまでもないことである。ここでは、ステップ数にして数十ないし数百行程度のプログラムの作成を支援するという問題についてのモデルをまず考えてみた。

このような、比較的小規模のプログラムの場合も、プログラムの作成は、要求の仕様化、仕様の詳細化、コーディングの、3つのステップに分けて考えることができる。このとき、各ステップは、それぞれ以下のような作業になる。

要求の仕様化は、プログラマが、作成したいプログラムで実現する仕事を、仕事を実現するための手順として表現することである。但し、この段階では、まだ具体的な個々の処理までは考えないことにする。このときに用いる知識としては、一般にプログラムを作るのに必要な知識のほかに、そのプログラムの対象とする応用領域に関する知識が、ある程度必要になる。

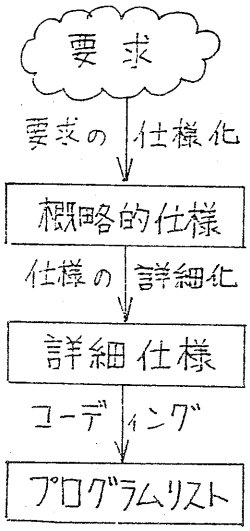
仕様の詳細化は、ここで作られた概略的な仕様を、より詳細な仕様へと、段階的に置き変えていくことである。この操作は、必ずしも、トップダウンに行なわれるとは限らない。

ここでは、概略的な仕様の他に、作成されるプログラムの用いるデータに関する記述が必要である。ここで用いられる知識は、プログラミング一般で用いる知識、作成されるプログラムが対象とする応用領域に関する知識である。また、プログラムの実行環境で利用可能なライブラリやサブルーチン・パッケージのなかのサブルーチンに関する記述も、参照されることが多い。その理由は、一般的に言ってこれらのプログラム・パーツの利用が、プログラミングの効率を良くするからである。仕様の詳細化は、仕様の個々の記述の全てが、プログラミング言語のステートメントと対応できるようになるまで続けられると考えてよい。

ここで作られた仕様を特定言語のプログラムに変換する作業が、コーディングである。ここでは、仕様の各要素を、対

応するステートメントに変換するための規則（特定の言語の文法規則，言語特有の制約など）が用いられる。また、効率よいプログラムを作り出すための、経験的な知識も、利用されることがあると考えられる。このようなモデルに基いた、プログラム作成の様子を図１に示す。

図１．プログラミングのモデル



通常、これらの作業の間には、仕様の検証や、検出された誤りを修正するための作業の手戻りがはいってくるはずである。ここでは、モデルを簡単化するために、各段階で用いられる知識が正確なものなら、作られる仕様やプログラムは正しいものとして、検証や、修正のための手戻りは一切考えなかった。

３．　システムの構成

３．１　システム設計の方針

第２章で述べたプログラミングのモデルに基いて、実用的な、しかし、比較的小規模なプログラムの作成を支援するシステムを試作してみた。（［５］，［６］）ここでは、まずこのシステムの設計の方針について述べる。

対象となるプログラムの範囲は、グラフィックス・サブルーチン・パッケージを用いた、２次元図形入出力プログラムとした。その理由は次のとおりである。

① プログラム・パーツとしてのサブルーチンの記述内容を検討する。このことは、プログラム・パーツの再利用の方法を検討するうえで、重要な課題の一つである。

② 図形の入出力技法として、数多くのものが知られているが、これらの技法のうち、問題に適したものの選択は、ある程度、経験的知識を要する。

③ 図形の入出力は、ＣＡＤシステム等における高度のマン・マシンインタフェースを提供するものとして、利用度が高い。従って、その意味でも、実用性の高いシステムとなる。

ここで、このように、作成されるプログラムの応用領域を限定した理由は、次のとおりである。応用領域を特に限定しない場合、応用領域に固有の知識は、プログラム作成支援システムには実装されていない。従って、作成されるプログラムの仕様のうちそのプログラムが対象とする応用領域に固有の部分についての仕様は、プログラマが、細かく指定しなければならない。この場合、プログラマが、作成したいプログラムに対して与える仕様の記述量は、かなり膨大なものとなる恐れがある。場合によっては、実際のプログラム・コードより長くなることもありうる。そこで、応用領域を限定し、その応用領域に固有の知識をプログラム作成支援システムに持たせることで、プログラムの仕様のうち、応用領域に固有の部分の記述量を減らすことが考えられる。このとき、作成されるプログラムの対象とする応用領域固有の知識により、仕様の詳細化手順の探索効率を向上させることが期待できる。この場合、プログラム作成支援システムの汎用性は損われるように見えるが、プログラム作成に必要な知識を、一般のプログラムに必要な知識と、特定の応用領域向けのプログラムのみに必要な知識とに分け、作成されるプログラムの対象とする問題に応じて、後者を交換できるようにすれば、ある程度の汎用性は、確保できる。尚、対象とする言語は、ここではとりあえずＦＯＲＴＲＡＮとしたが、これも、プログラミング言語への変換ルールを他のプログラミング言語向きのものに交換することにより、特定のプログラミング言語に捉われないシステムができる。

プログラマから、プログラム作成支援システムへのプログラムの仕様の入力は、語彙および構文の限定された英文によることにした。ここで、特定の仕様記述言語を用いなかった理由は、次のとおりである。

特定の仕様記述言語を用いる場合、そのための仕様記述言語の設計が必要になる。しかし、仕様記述言語の設計は、一般に難しく、仕様記述言語の設計如何によっては、システムの拡張性を著しく損う恐れがある。また、特定の仕様記述言語の習得が、ユーザの負担となることが多い。以上の点で、特定の仕様記述言語の利用は得策ではないと判断した。

仕様の詳細化は、抽象的な仕様から、より詳細な仕様への変換であると考えられる。従って、このための変換規則を、プログラム作成支援のシステムの知識として与えることにした。知識の表現形式としては、プロダクション・ルールが適していると考えた。なぜなら、プログラムの仕様の詳細化においての適切な詳細化の手段の選択は、そのときの、種々の条件を参照して、複数の候補のなかから、条件に適合することと考えられ、プロダクション・ルールで表現するのに向いている。また、プロダクション、ルールは、実装も簡単で、しかも、追加・修正が比較的容易だからである。この場合、条件の部分は、抽象的な仕様で、行動の部分は詳細化を行なう環境、詳細化された仕様になる。つまり、以下のような形式で表現される。

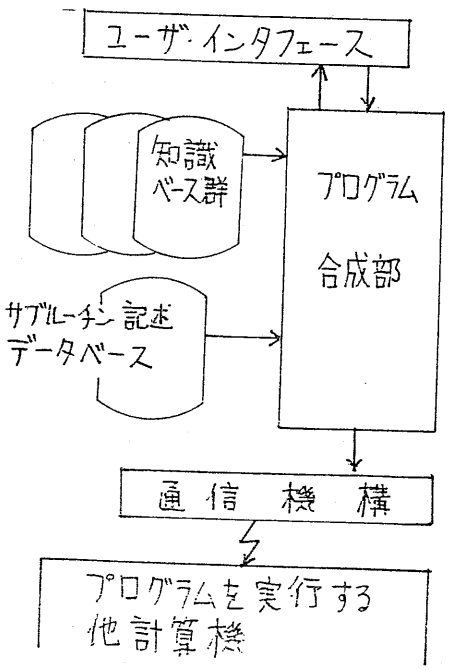
if＜抽象的な仕様＞then＜条件＞，＜詳細化された仕様＞．

このような形式で表現される知識および、それを使って処理をするシステム全体は、論理型プログラミング言語PROLOG（C－Prolog ）で記述した。
このとき、上のプロダクション・ルールは次のような形で表現される。
＜抽象的な仕様＞：－＜条件＞，！，
＜詳細化の処理＞．
システムは、VAX11／730上に実装され、VAX／UNIX上のC－Prolog のもとで動作する。

3．2　システムの構成

本システムの構成を図2に示す。本システムは、ユーザ・インタフェース、プログラム合成部、知識ベース群、サブルーチン記述データベース、通信機構の5つの要素からからなる。

図2　システムの構成



ユーザ・インタフェースは、ユーザの入力するコマンドの解析、ユーザへの応答メッセージの作成などをおこなう。
プログラム合成部は、ユーザから与えられた、作成したいプログラムに関する情報をもとに、まず、プログラムの概略仕様を生成し、次にそれをもとに、詳細仕様を作成し、最終的に、詳細仕様からプログラム・コードを生成する。詳しい動作の内容については、第4章で説明する。

知識ベース群には、仕様の作成・詳細化、プログラム・コードの生成に必要な知識が、異なる種類の知識毎に独立に扱えるような形で格納されている。これらの知識の分類および、その内容については、次節で述べる。
サブルーチン記述データベースには、各種のサブルーチン・パッケージ中のサブルーチンに関する記述が格納されている。この記述内容についても、後に説明する。
本システムでは、異なる計算機の上で実行されるプログラムの作成を仮定している。従って、作成されたプログラムを、それを実行する計算機に転送するための機能が必要となる。この機能を提供するものとして、通信機構を設けてある。さらに、知識ベースが、複数の計算機上に分散して存在する場合、他の計算機の中にある知識ベース中の知識を参照する時にも、この通信機構が利用できる。
本システムは、現在VAX11／730の上で稼動中である。現在の所、サイズは、C－Prolog で5000行程度の大きさである。本システムでは、GSPCが提案した、グラフィックス・サブルーチン・パッケージの標準化案であるCORE［3］に準拠して、当研究室で開発したサブルーチン・パッケージMT－COREを用いた2次元図形入出力プログラムが作成可能で、FORTRANのステートメントにして数十ステップ程度の大きさのものを作成可能である。この制約は、主に、プログラム作成支援システムが実装される計算機のメモリのサイズによるもので、メモリ・サイズがもっととれるなら、作成できるプログラムのサイズも、より大きくなるはずである。

3．3　システムの持つ知識

第2章で述べたように、プログラム作成には、作成されるプログラムの対象とする応用領域に依存しない知識と、応用領域に依存する知識がある。ここでは、前者をプログラミング一般に関する知識、後者を応用領域固有の知識と呼んでいる。この他に、プログラムの仕様から、特定のプログラミング言語でのプログラム・コードを生成するのに、そのプログラミング言語への変換規則が必要であり、これを、その言語に対する言語固有の知識と呼ぶ。
既に述べたように、本研究では、プログラムの作成を、概略仕様の作成、概略仕様からの詳細仕様の生成、詳細仕様のプログラム・コードへの変換から成ると見なしている。従って、プログラム作成支援システムに実装するべき知識としては、これらの変換のために必要な変換規則や、それらの変換規則選択のためのパラメータの推論規則が必要になる。後の、システムの拡張を行なう時のことを考えて、これらの知識を、作成されるプログラムの対象とする特定の応用領域によるもの（応用領域固有の知識）と、そうでないもの（プログラミング一般に関する知識）に分け、これらの知識を独立して扱えるようにした。
上で述べたことから、プログラミング一般に関する知識は、作成されるプログラムの仕様のうち、特定の応用領域とは無

関係な部分を詳細化するための変換規則、および、規則選択のための補助的な規則となる。このような知識の例を図3－a に示す。

図3-a　プログラミング一般に関する知識

```
refinement([subst,Var1,Var2]):-
        is_defined(Var2),
        !,
        make_dspec1([subst,Var1,Var2]).

refinement([subst,Var1,Var2]):-
        !,
        insertdef(Var2),
        make_dspec1([subst,Var1,Var2]).

refinement([read,Data]):-!,
        refinement([prompt]),
        refinement([readinputdevice,
                    Data,Inputdev]),
        refinement([errorcheck,Data]).
```

例に示された知識のうち、上は、変数への代入に関する一般的な知識で、変数を代入するときには、代入されるほうの変数は、何らかの値を持つべきことを示している。下は、ユーザからの入力に対する一般的知識であり、ユーザからの入力の前に、プロンプティング・メッセージの出力が、入力の後に、その入力に対するエラー・チェックが必要であることを示している。

次に、応用領域固有の知識は、作成されるプログラムの仕様のうち、特定の応用領域に依存する部分を詳細化するための変換規則、および、規則選択のための補助的な規則となる。ここでは、プログラムの仕様のうち、図形の入出力に関係する部分の詳細化のための変換規則、これらの規則を選択するためのパラメータを推論する規則が実装される。これらの知識の例を図3－b に示す。

図3-b　図形入出力に固有の知識

```
refinement([specify,[Varx,Vary]]):-!,
        varrole(Varx,[specify,position]),
        varrole(Vary,[specify,position]),
        typeofdisplay(raster),!,
        refinement([readlocator,Varx,Vary,
                    dragging]).

refinement([draw,polygon,Xarray,Yarray]):-!,
        variables(Xarray,_,[S],_,_),
        variables(Yarray,_,[S],_,_),
        !,
        makearraylast(Xarray,S,Xarrays),
        makearraylast(Yarray,S,Yarrays),
        refinement([call,movea,
                    [Xarrays,Yarrays]]),
        refinement([call,pllina,
                    [Xarray,Yarray,S]]).
```

上の知識は、グラフィック入力技法の選択に関する知識で、シンボル等の位置を指定する場合で表示装置が、ラスタ・スキャン型のものなら、ドラッギングと呼ばれる入力技法を選択することを示している。下の知識は、配列で現わされる、座標のデータで多角形の表示を行なうための方法を示す知識である。

言語固有の知識としては、FORTRANに関する知識を実装した。ここで実装されているものは、大部分が、詳細化された仕様の各部分から、FORTRANのステートメントへの変換規則である。その例を図3－c に示す。この例では、条件反復の一種であるwhile －doループとその中身をFORTRANのステートメントの系列に変換するための規則を示す。

図3-c　実装言語固有の知識

```
fortran_conv([while_do,Cond,Statelist]):-
        make_fortcond(Cond,Fcond),
        make_notcond(Fcond,Result),
        generate_label(Lab1),
        writeln(['  ',Lab1,'  ',continue),
        generate_label(Lab2),
        writeln(['       ',if,Result,go,to,Lab2]),
        fortranconv0(Statelist),
        writeln(['       ',go,to,Lab1]),
        writeln([' ',Lab2,'       ',continue]).
```

これらの知識は、グラフィック・サブルーチン・パッケージを用いた簡単な図形入出力プログラムの2～3の例を考え、それらのプログラムの作成に必要な知識を探しだし、リストアップして、これをプロダクション・ルールで表現することを考えて、さらにそれをPROLOGの述語として実装するためHorn-Clause の形に直したものである。各種類毎の知識の数は、次のとおりである。

① プログラミング一般に関する知識　199個
② 応用領域固有の知識　250個
③ 言語固有の知識　62個

さらに多くの例題を考えていくと、新しい知識の追加が必要になる場合が生じることは、十分考えうることである。このときの知識の追加については、第5章で述べる。

3．4　システムの参照するデータ

第2章で述べたように、プログラムを作成する時には、そのプログラムで用いる、ライブラリや、サブルーチン・パッケージのなかのサブルーチンに関する情報が必要になる。それは、プログラム作成の効率を良くするためには、通常、各種のプログラム・パーツの利用が行なわれるからであり、このようなプログラム・パーツとしてよく利用されるのが、各種のライブラリやサブルーチン・パッケージのなかのサブルーチンだからである。この情報をプログラム作成支援システムに持たせることは、プログラミングの効率をあげるという効果をもたらすほかに、プログラマが、作成したいプログラムのなかで使う必要になるライブラリや、サブルーチン・パッケージのなかのサブルーチンに関する詳細事項を知っている必要がないという点でプログラマの負担を軽くしている。

これらの情報は、作成されるプログラムの仕様を詳細化するための知識により参照される。具体的には、ある一つの機能（例えば、多角形を表示する、入力装置を初期化する、等）を実現するサブルーチンを、これらの情報を手掛りに探し、そのようなサブルーチンがみつかれば、要求される機能は、そのサブルーチンを呼び出す手続きとして実現するといったような形で用いられる。

本研究においては、ライブラリや、サブルーチン・パッケージのなかのサブルーチンの記述として、次のようなものを

考え、実装した。

① サブルーチンの名前
② 引数に関する記述（仮引数名，型，役割，制限）
③ サブルーチンの機能
④ サブルーチンの実行可能条件

これらは、すべて、プログラム作成支援システムの内部表現であり、ユーザから直接にはアクセスできない。図4に、サブルーチンに関する記述の例を示す。

図4 サブルーチンに関する記述

```
subdesc(rdloc,
        [[locnum,int,
         [specify,number,of,locator],
         range(1,1)],
         [xinput,real,
          [return,ndc,coordinate,x,of,locator],
          []],
         [yinput,real,
          [return,ndc,coordinate,y,of,locator],
          []]],
        [putval(locatorx,xinput),
         putval(locatory,yinput)],
        [state(corestate,init),
         state(vsstate,[init,select]),
         device_stste([5,locnum],
                      [init,enable])]).
```

例に示されるのは、MT－COREのサブルーチンの 1つで、ロケータ入力装置の現在持つ座標値を読むものである。名前は、RDLOCで、引数として、xinput（実数），yinput（実数），locnum（整数）をとり、xinput，yinputにロケータの持つ座標値がはいる。locnumはロケータの番号を指定する。これは、COREが初期化され、視表面が初期化・選択され、指定するロケータ装置が初期化・enableされていないと使えない。

4． システムの動作

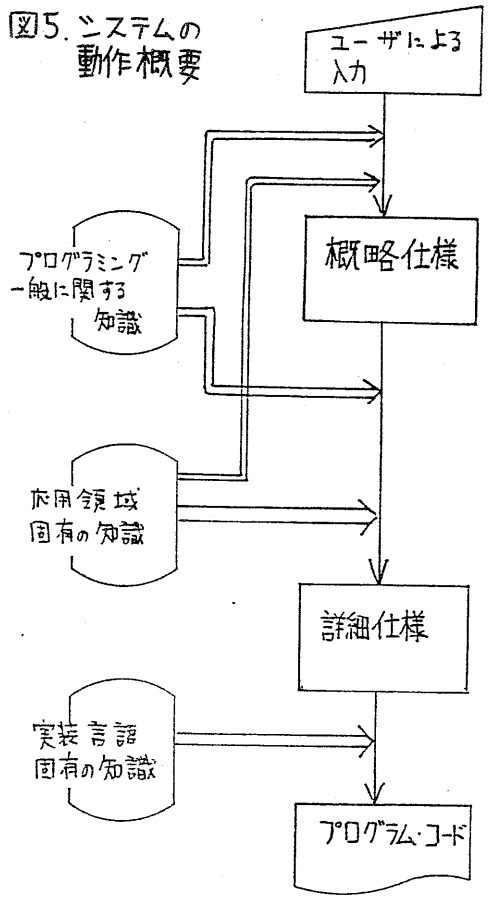
次に、本システムの動作についての説明を行なう。本章では、まず、動作の概略を説明し、その次に、具体的な例題に沿って、システムの動作の説明を行なっていく。この例において、前章までに述べていなかった、ユーザの入力や、プログラムの概略仕様、詳細仕様のシステム内部表現および、プログラムで用いられるデータに関する記述のシステム内部表現についても説明していく。

4．1 動作の概要

図5に、本システムの動作の概要を示す。この図において、実線の矢印は、本システムにおけるプログラム作成支援の処理の流れ、二重線の矢印は、前章で述べた三種類の知識の処理の各段階への作用を示す。システムの動作は、次のようなステップで行なわれる。

1. 質疑応答形式による、データに関する記述の入力
2. 英文による、プログラムの概略仕様の入力
3. 概略仕様への、不足部分の追加
4. 概略仕様からの、詳細仕様の生成
5. 詳細仕様からの、プログラム・コードの生成

図5. システムの動作概要



以下、これらの各ステップについての説明を行なう。

1. 質疑応答形式による、データに関する記述の入力

ここで言うデータとは、作成されるプログラムのなかで用いられるデータのことである。データとしては、テーブル型データ構造および、その階層化されたものと、単一のデータ項目、および、定数項を考えている。階層化された構造の場合、各階層には、テーブルは一つと制限している。ここで、データに関する記述の内容について説明する。テーブル型データ構造については、次のようなことが記述される。

① データ構造の識別名・各階層の識別名・各階層ごとのデータの最大数
② 各階層に含まれる変数の名前
③ 各変数に関する記述（変数名・型・配列としてのサイズ・役割・値の範囲）

単一のデータ項目については、変数名、型、役割、値の範囲を記述する。

定数項については、定数の名前、定数の値、定数の型、定数の役割を記述する。

図6にデータ構造の例と、それに対する記述の例を示す。図6-aで、データ構造の図を、図6-bで、これに対する本システムでの内部表現を示す。 これらの記述を作るための情報を、質疑応答形式で、システムに与えていく。具体的な質疑応答のようすは、後ほど、図7にて示す。

2. 英文による、プログラムの概略仕様の入力

データに関する記述の入力の次に、プログラムの概略的な仕様を、語彙・構文の限定された英文で入力する。入力される英文は、動詞＋目的節＋補語節、動詞＋目的節＋形容詞節、動詞＋目的節＋副詞節のいづれかの構文をもつ命令文の形をとる。動詞としては基本的なものを、20語ほど用意した。このほか、普通名詞が、約60語、副詞が約10語、形容詞が約10語、その他、接続詞や前置詞が約20語ほど登録されている。未登録の動詞がはいっていたり、誤った文法形式の文がはいると、仕様の詳細化の際、仕様の再入力を要求される。図8に入力される英文の例を示す。

3. 概略仕様の不足部分の追加

ユーザの指定した概略仕様で、明らかに不足している部分を、システムが検出し、適当な位置に挿入する。ここに到って、始めて、概略仕様ができたことになる。

4. 概略仕様からの、詳細仕様の生成

プログラムの概略仕様、プログラムで用いるデータに関する記述を用いて、そのプログラムに対する詳細仕様を生成する。ここで、各種のライブラリや、サブルーチン・パッケージのなかのサブルーチンに関する記述で、本システムのサブルーチン記述データベースに登録されているものを参照する。本システムにおいては、概略仕様に対し、段階的に変換規則を適用していき、詳細仕様を得ていく（図10）が、この時、必要なパラメータ等で、明示されていないものについては、システムの持つ知識で推論するか、ユーザに対して問い合せをすることによって決定していく。従って、詳細化は、対話的に進められる。

5. プログラム・コードの生成

4. で最終的に得られた仕様が、詳細仕様である。これは、プログラミング言語の１ステートメント、或いは、複数のステートメントの、意味のある系列に対応するステートメントの系列をリストの形で表現したものである。詳細仕様の基本要素として、代入、分岐、条件分岐、反復、条件反復を用意した。このうち、（条件）分岐や（条件）反復は、ネスティングが可能であるようにした。この仕様で記述されているプログラムの実際の動作のシーケンスは、この基本要素の並びで表わされる。これらの基本要素は、ＦＯＲＴＲＡＮやＰＡＳＣＡＬ等の手続き型のプログラミング言語への変換を想定して決めてある。図11に、その例を示す。

このような形で記述された詳細仕様を、特定言語のプログラムに変換する時に、その実装言語固有の知識を用いる。このようにして生成されたプログラムは、特定言語のソース・ファイルをユーザに名前を指定してもらい、この名前で作成し、そのファイルに出力される。もし、作成されたプログラムが、他計算機の上で実行するものならば、ユーザは、システムに対し、プログラム転送のコマンドを発し、目的とする計算機へソース・ファイルを転送することができる。

次節で、例題に対する動作の説明を行なう。

４．２　例題に対する動作

図形入出力プログラムの例として、文献［４］の、家具配置のデザイン設計プログラムをあげてみた。ここでは、問題を単純化するため、以下の様な仮定を置き、プログラムの仕様も、文献［４］のものよりも、簡単にしてある。

1.　家具のシンボルは全部で６種類あるが、これを表示するためのサブルーチンが既に、作成されるプログラムを実行する計算機の上に与えられているものとする。これは、シンボルコード、位置、角度を与えると、家具のシンボルを指定された位置に、指定された角度で表示するものである。このサブルーチンに関する記述が、プログラム作成支援システムのサブルーチン記述データベースに登録されていると仮定する。

2.　家具シンボル表示の為のデータは、既に、作成されるプログラムが実行される計算機のファイルの中にあり、作成されるプログラムでは、このファイルを読んできて使うものとする。

3.　シンボルの位置、表示されているシンボルの指定は、ロケータ型入力装置にて行ない、他の指定は、キーボードから行なうことにする。

このプログラムでは、図６-aに示されるようなデータを用いて、処理を行なう。データに関する記述は、図６-bのようになる。

図６-a　例題プログラムで用いるデータ

	symbol	xposi	yposi	rotate	visibl
1	1	4.0	4.0	0.0	true
2	2	13.0	8.0	180.0	true
3	1	15.0	13.0	180.0	true
≈	≈	≈	≈	≈	≈
20					

図６-b　上のデータに関する記述

```
d_str(furnitures,[furniture],[20]).
tables(furniture,[symbol,xposi,yposi,rotate,visibl]).
variables(symbol,integer,[20],
          [specify,sort,of,the,furniture],
          range(0,20)).
variables(xposi,real,[20],
          [specify,position,x,of,the,furniture],
          range(0.0,30.0)).
variables(yposi,real,[20],
          [specify,position,y,of,the,furniture],
          range(0.0,30.0)).
variables(rotate,real,[20],
          [specify,direction,of,the,furniture],
          range(0.0,360.0)).
variables(visibl,logical,[20],
          [state,the,furniture,is,seen,or,not],
          [true,false]).

variables(nfurn,integer,[],
          [specify,the,number,of,the,furniture],
          range(0,20)).
variables(icomm,integer,[],
          [specify,the,command,selected],
          range(1,6)).
```

この記述を得る為の、システムとユーザとの対話の一部を、図7に示す。データに関する記述の入力は、一問一答形式によりシステムにガイドされて進められていく。

図7　データに関する記述の入力

```
Enter Data name
¦: furnitures.
Is the data furnitures srhtructured?
¦: yes.
Enter names of the layer of furnitures.
¦: [furniture].
Enter maximum data size for furniture.
¦: 20.
Enter specification data of furniture.
¦: [symbol,xposi,yposi,rotate,visibl].
Enter type of the data : symbol.
¦ integer.
Enter role of the data : symbol.
¦ specify sort of the furniture.
Enter range of the data value : symbol.
¦: [1,6].
```

このプログラムでは、次のような処理をメニュー選択形式で選び、実行したのち、再び、メニュー選択にもどるということを、処理の終了が選択されるまで繰り返す。

a 家具配置の、データ構造furnituresに基く表示
b 家具配置図へのタイトルづけ
c データ構造furnituresへのデータの追加
d データ構造furnituresからのデータの削除
e 表示領域の変更と表示
(f 処理の終了)

これを、概略仕様として入力したものが図8である。

図8　ユーザの入力した概略仕様

```
¦: Select one of follows.
¦: Draw a picture by furnitures.
¦: Specify the title of the picture.
¦: Add data_of_a_symbol to furnitures.
¦: Delete data_of_a_symbol from furnitures
¦: Change the display area of the picture.
¦: selection end.
¦: end.
```

しかし、ユーザの入力した、この仕様は不完全である。例えば、メニュー選択の項目の中に、処理の終了が入っていない。また、家具の表示に用いるデータfurnituresを最初、どこから持ってくるのかが不明確である。また、グラフィックスの初期化も指定されていない。このような不備を、システム側で補って、概略仕様を完成させるのが、概略仕様への、不足部分の追加である。

ここでは、データ構造furnituresによる表示が指定されているので、グラフィックスの初期化と、データ構造furnituresの、ファイルからの読み込みあるいはキーボードからの読み込みが必要である。(前者は、図形入出力固有の知識、後者は、プログラミング一般に関する知識により、指示される。)そこで、グラフィックスの初期化、さらに、データ構造furnituresが既にファイルのなかにあることから、データ構造furnituresのファイルからの読み込みを、前に追加する。さらに、furnituresに対するデータの追加・削除が指定されているので、処理の最後に、データ構造furnituresの、ファイルへの書き込みが追加(データ・アクセスに関するプログラミング一般に関する知識により指示される。)、さらに、グラフィックスの初期化がこの時点で仕様に組み込まれているので、図形入出力固有の知識により、仕様の末尾にグラフィックスの終了が付け足される。また、コマンドの選択の実装に用いられる、プログラミング一般に関する知識により、メニュー項目に、処理の終了が追加される。これらの処理は、システム側で行なわれる。以上のような操作が、システムで行なわれるので、ユーザの与える仕様は、若干の不完全性が許される。なお、furnituresのあるファイル名については、ユーザに問い合わせが来る。以上のようにしてできた概略仕様は、図9のようになる。

図9　システムで補正した概略仕様

```
[initialize,graphics,
 [bufferd,synchronous,2d,none]]
[read,furnitures,from,file001]
[menu,menu000,
 [draw,specify,add,
  delete,change,exit]]
[draw,picture,furniture]
[specify,
 [the,title,of,the,picture]]
[add,data_of_a_symbol,furnitures]
[delete,data_of_a_symbol,furnitures]
[change,
 [the,display,area,of,the,picture]]
[exit,menu000]
[write,furnitures,file001]
[terminate,graphics,
 [bufferd,synchronous,2d,none]]
[end]
```

概略仕様からの詳細仕様の生成

次に、図6-bに示す、データ構造に関する記述と、図9に示す、プログラムの概略仕様をもとに、詳細仕様を生成する。

この処理は、ユーザとの対話を交えて行なわれる。また、概略仕様を構成する仕様の要素毎に、要素の並んでいる順番に処理を進めていく。これら全部について説明するのは、ページ数の都合上無理があるので、ここでは、そのなかの一つ、[add , a-data-of-a-symbol, furnitures]の詳細化について説明する。この様子を、図10に示す。

まず、データ構造に対するデータの追加に関する、一般的知識(データの追加にあたっては、まずデータの追加の余裕があるかをチェックし、余裕があれば、追加するべきデータを指定し、これらを追加する。)が利用され、図10の①に示すような詳細化が行なわれる。ここでは、データ構造が、固定サイズのテーブルで拡張はしない(デフォルト・ケース)と仮定されるので、データテーブルに、これ以上のデータの追加のための余裕がないときの処理は、データの追加をしないで戻ることにする。(これは、このシステムでのデフォルト・ケースである。)

次に、システムはデータを追加する余裕がない（no－room）を、データ構造に関する記述（データのサイズは20まで）から、家具シンボルの数が20を越えた時と判断し、それに対する条件の記述を生成する。また、追加されるデータは、データ構造の記述から、シンボルコードsymbol、シンボルの位置xposi,yposi 、シンボルの角度rotate、可視性visiblであることがわかり、データ項目の指定も、これらの指定に置き変わる。また、データの追加は、特に指定がないので、データ・テーブルの最後に追加されるという操作に置き変えられる。以上の決定は、プログラミング一般に関する知識により、システムが下しており、ユーザは、これらの指定をしなくてすんでいる。これによる詳細化は、図10の②で示される。

次の詳細化のステップでは、各々のデータの指定のしかたを決める。この決定は、システムからユーザに問い合わせを発し、ユーザが答を返すことで行なわれる。ここでは、xposi,yposi はロケータで、その他はキーボードで指定されることにする。（人間とのインタフェースに関しては、なるべくユーザに決めさせるという方針を、本システムではとっている。）これにより、図10の③に示す詳細化が行なわれる。

ここからは、図10-cの下線部の詳細化を次に説明する。（図10-cの仕様全部についての説明はしない。）

このレベルからは、図形入出力に関する、固有の知識の役割が、大きくなってくる。まず、ロケータ装置を使用するためには、ロケータ装置が初期化され、enableになっていなければならない。ここでは、これらの条件がまだ満たされてないので、ロケータ装置の初期化、ロケータ装置のenableが、それぞれ、グラフィックスの初期化の直後、ロケータ装置使用開始の直前に追加される。これらの決定は、図形入出力固有の知識により、システムのなかで下される。また、ロケータ装置からの入力は、座標変換の必要があるので、読み込みの後に座標変換が追加される。これも、図形入出力固有の知識による判断である。このときの詳細化の様子を、図10の④に示す。

図10　段階的な詳細化の様子

```
[add,data_of_a_symbol,furnitures]
  a)                      ① V
[if,no_room,then,return]
[specify,dataitems,for,symbol]
[add,dataspecified,furnitures]
  b)                      ② V
[if_then,[nfurn,gt,20],return]
[specify,symbol]
[specify,[xposi,yposi]]]
[specify,rotate]
[specify,visibl]
[subst,nfurn,nfurn+1]
[addlast,[symbol,xposi,yposi,rotate,visibl]]
  c)                      ③ V
[if_then,[nfurn,ge,20],[go,to,menu000]]
[read,symbol]
[read,locator,[xposi,yposi],normal]  ----------
[read,rotate]                                  ④
[subst,visibl,'true']                          |
[subst,nfurn,nfurn+1]                          |
[addlast,[symbol,xposi,yposi,rotate,visibl]]   |
```

図10の続き

```
                                               |
                                               V
[initialize,locator]   ..... inserted before
  :                          reading data from file
  :
[enable,locator]
[wait,locator_input_value,[ix,iy]]
[convert_ndc_to_wc,[ix,iy],[xinput,yinput]]
```

以上のように、プログラミング一般に関する知識、応用領域（図形入出力）固有の知識を用いて、ユーザとの対話を行ないながら、仕様の詳細化を進めていく。以上のような詳細化を進めていくことにより、［add，．．．］で指定される仕様の要素に対して、最終的に図11に示すような、詳細仕様が得られる。このような詳細仕様の記述は、ユーザからは、直接には見えない。

図11　図10によりできる詳細仕様

```
[if_then,[[nfurn,ge,20]],
         [[write,'FULL DATA !'],
          [go,to,menu000]]]
[write,'ENTER SYMBOL 1:CHAIR 2:DESK
 3:BOOKCASE 4:LOWDIVIDE
 5:PARTITION 6:PLANT']
[read,isymb]
[if_then,[[isymb,lt,1],or,[isymb,gt,6]],
 [[read,isymb]]]
[call,enadv,[5,1]]
[call,enadv,[3,1]]
[call,awblc2,[1,ibtn,300.0,ix,iy]]
[call,ndctow,[ix,iy,xinput,yinput]]
[call,disadv,[3,1]]
[call,disadv,[5,1]]
[write,'ENTER DIRECTION OF FURNITURE']
[read,dirinp]
[if_then,[[dirinp,lt,0.0],or,
         [dirinp,gt,360.0]],
         [[read,dirinp]]]
[subst,nfurn,nfurn+1]
[subst,symbol,[nfurn],isymb]
[subst,xposi,[nfurn],xinput]
[subst,yposi,[nfurn],yinput]
[subst,rotate,[nfurn],dirinp]
[subst,visibl,[nfurn,'true']
[call,furnit,
 [isymb,xinput,yinput,dirinp]]
[go,to,menu000]
```

また、特定の機能を果たすサブルーチン（例えば、ロケータ装置の起動、座標変換など）の探索と選択は、システム側で行なってくれるので、ユーザは、個々のサブルーチンとして何があるか、どんな機能を持つか知らなくてもよい。但し、サブルーチン・パッケージや、ライブラリの選択は、ユーザに、一部任せてあるので、ユーザは、サブルーチン・パッケージやライブラリとして、何が使えるかを知っている必要がある。

詳細仕様から、特定のプログラミング言語への変換は、自動的に行なわれる。本システムでは、FORTRANプログラム・コードへの変換を行なう。変換は、全く自動的に行なわれるので、ユーザは、特定のプログラミング言語の文法を全く知らなくてもよい。但し、プログラムを実装する言語の

選択は、一部はユーザの責任で行なうので、ユーザは、どんな言語が使えるかを知る必要がある。図11に示した詳細仕様からは、図12に示されるようなプログラム・コードが生成される。

図12　図11の詳細仕様に対するプログラム

```
      if (nfurn .ge. 20) then 200
210   write (6,220)
220   format (' ENTER SYMBOL 1:CHAIR 2:DESK
     c  3:BOOKCASE 4:LOWDIVIDE
     c  5:PARTITION 6:PLANT')
      read (5,230) isymb
230   format (i1)
      if ((isymb .lt. 1) .or. (isymb .gt. 6)
     c  goto 210
      call enadv (5,1)
      call enadv (3,1)
      call awblc2 (1,ibtn,300.0,ix,iy)
      call ndctow (ix,iy,xinput,yinput)
      call disadv (3,1)
      call disadv (5,1)
240   write (6,250)
250   format ('ENTER DIRECTION OF FURNITURE')
      read (6,270) dirinp
      if ((dirinp .lt. 0.0) .or.
     c (dirinp .gt. 360.0))
     c goto 240
      nfurn = nfurn + 1
      symbol(nfurn) = isymb
      xposi(nfurn) = xinput
      yposi(nfurn) = yinput
      rotate(nfurn) = dirinp
      visibl(nfurn) = .true.
      call furnit (isymb,xinput,yinput,dirinp)
      goto 100
200   write (6,260)
260   format ('FULL DATA !')
      goto 100
```

なお、生成されたコードは、VAX／UNIX上のFORTRAN77のコンパイラにかけてみた所、文法上の誤りは検出されなかったので、文法上の誤りは、全くないといってよいと思われる。

5.　考察と検討

5．1　プログラム作成支援について

ここで、本システムを、プログラミング支援システムとして見た場合の考察と検討を行なってみる。

本システムでは、プログラムの作成の際に、そのプログラムで用いるデータの記述と、プログラムで行なう処理の大体の流れを指定するだけで、あとは、仕様の詳細化の時に、システムから発せられる質問に答えるだけでよい。これだけなら、一見、従来までに開発されてきたソフトウェア作成支援システムに比べて優れている点はあまり無いように見える。

しかし、知識を用いることにより、従来のシステムと比較して、次のような点で便利になっている。

まず、従来のシステムでは、プログラムの動作に対する仕様の記述には、曖昧さや、不完全な記述は許されなかった。従って、プログラマは、最初から、明確な仕様を考えて、それから仕様を記述しなければならなかった。しかし、通常は、最初からプログラムの明確な仕様を決めることは、困難なことである。これに対し、本システムでは、最初の仕様記述での曖昧性や不完全性をある程度許し、完全な仕様は、システムの持つ種々の知識で補っていくようにした。これにより、最初から仕様を明確にする必要性から、プログラマを解放できると思われる。

また、仕様の詳細化の段階で、詳細手順の選択に必要なパラメータを、或る程度システムの持つ知識で推論させることにより、パラメータの決定の負担を軽減させている。また、手順を決定するための基準も、単に、詳細化される仕様だけを基準とするのではなく、プログラムの実行される環境や、経験的な知識も、基準として用いることができる。これは、従来のソフトウェア作成支援システムでは、ほとんど実現不可能な機能であると思われる。また、このような、手順選択の基準の柔軟性も、実際の、プログラムに対する仕様の詳細化の際に欠かすことのできない要素であると思われる。

なお、仕様の入力に、簡単な英文を用いることで、特定の仕様記述言語を全く知らない人にも、簡単に使えるシステムを構築できたと考えられる。自然言語処理の研究が進み、さまざまの技法が実用化されれば、より高度なマン・マシンインタフェースが提供できると期待される。

従来のソフトウェア作成支援システムでは、システムの拡張をする時に、大幅な改良が必要であった。これに対し、本システムのように、プロダクション・システムの形をとると、システムの拡張性は、かなり改善される。従来のシステムでは、制御部分の細かな改良が必要であったのに対し、本システムでは、単に知識を追加するだけで、制御の部分は、ほとんど手を加えなくてもよい。但し、知識の追加にあたっては、何らかの支援ツールが必要だと思われる。

なお、本システムのように、知識ベース型のシステムにすると、従来のシステムでは表現しにくい、経験的知識が容易に組み込め、効率のよいシステムを構築できることが期待される。

また、本システムで採用した、サブルーチンに関する記述も、プログラム作成の際に参照するデータとしては、十分実用になっていると思われる。今後は、この記述が、プログラムの検証や、説明などに対しても、実用になるかどうかの検討が必要であろう。

但し、知識ベース・システムには、次節で述べるような問題点がある。また、本システムは、比較的小規模な実用プログラムの作成支援を対象としており、ソフトウェア工学で一般に対象とする大規模プログラムの開発支援に、本システムの構築で考えたプログラミングのモデルやプログラム作成支援システムの構築方法がそのまま通用するかどうかははなはだ疑問である。

5．2　知識ベース・システムとしての問題点

本システムの試作にあたって、知識ベース・システムの構

築という観点から問題になったことをあげてみる。近年、知識工学は、一種のブームになっている。すなわち、さまざまな応用領域に対していろいろな知識ベース・システムの開発を行なっていこうとする気運が見られるが、始めに述べたように、これらのシステムの開発には、いまだしっかりとした方法論が示されていない。

本システムにおいても、実は、このような反省事項があてはまる。つまり、システムの構築にあたって、これといった方法論を立てて行なったわけではなく、他のシステム同様、職人芸的な方法に頼っていた。つまり、適当なモデルを考え、具体的な例題を設定して、そのなかから、必要な知識が何であるかを考え、これを、プロダクション・ルールで表現したものを、PROLOGの述語として表現するために、Horn Clause の形に直してはシステムの知識として追加していた。

このため、新しい例題を行ない、新たな知識が必要とわかった時に、知識を追加する際、デバッグにかなりの手間をとられた。具体的には、新たな知識をHorn Clause で表現した時、そのサブ・ゴールに、システムにないものをいれ、そのため、余計にHorn Clause をさらに追加するはめになったりした。新しい例題を見つけては、これに対して、知識ベース・システムをテストし、必要となる新たな知識を追加していくというパラダイムは、知識ベース・システムの場合、避け難いものと思われる。しかし、本研究のような方法では、極めて効率が悪くなる。

従って、このような知識ベース・システムの構築には、確立された方法論と、知識を追加するための、その方法論に基づいたツールが必要であると言ってよい。このようなツールの機能としては、知識ベースで解決したい問題に対するモデルを構築・修正する機能、知識ベース・システムの基本的な知識を保持する機能、ユーザの入力をHorn Clause に翻訳する機能、追加された知識の正当性や冗長性をチェックする機能が必要である。

5．3　今後の課題

本研究で試作したプログラム作成支援システムは、今後さらに、知識の追加や修正が必要であり、このために、前述のようなツールの開発が必要である。また、実用的な、プログラム作成支援システムにするために、プログラムの修正・保守の機能が必要である。また、プログラム・パーツの再利用の観点からも、ユーザの作ったサブルーチンの記述を追加する機能も必要である。

なお、本システムは、一種のプロダクション・システムの形をとっているが、他の知識表現を用いたシステムについても、検討してみる必要があるかも知れない。

これらの機能を実現するために、第2章で考えたモデルの修正も必要になることも考えられる。これらのモデル修正も支援できるような知識ベースの開発支援ツールが必要であり、このようなツールの開発のためにも、知識ベース・システムの構築法の確立が望まれる。

謝辞

本研究を行なうにあたり、貴重な御助言を下さいました、元岡教授ならびに田中助教授、そして、研究室の方々に感謝いたします。

<<参考文献>>

[1] Stanford Univ．編：”Heuristic Programming Project ´80” pp1～78

[2] P. H. Winston, R. H. Broun：”Artificial Intelligence An MIT Perspective” 1979 MIT Press

[3] Wright T：”Status Report of the Graphics Standards Planning Committee” SIGGRAPH Quarterly Vol13 No3 Aug.1979

[4] R, D. Bergern, P. R. Bono , J. D. Foley ”Graphics Programming Using the CORE System ”. ACM Computer Surveys Vol 10 No 4 . Dec.1978

[5] 吉田、田中、元岡：「知識工学の手法を用いたソフトウェア作成支援システム」，第27回情報処理学会全国大会4B－9 pp 529～ 530

[6] 吉田、田中、元岡：「プロダクションルールを用いたプログラム作成支援システムに関する考察と検討」，第28回情報処理学会全国大会3G－9 pp1055～1056

盛光印刷所