総　論

1. 自動プログラミング

1.2 人工知能からみた自動プログラミング†

大須賀　節　雄††

1. はじめに

現実の社会において人の行う情報の処理方式とコンピュータによる処理方式とは本質において異なり，この橋渡しの役割がプログラマに課せられていた．その仕事の性質からプログラミングは本来困難をともなう作業であり，まず仕様記述を行い，それをプログラムに変換するという手順が標準的とされてきた．前段における難しさは現実の問題からプログラム化のための完全な仕様を作成することにあり[1]，これはプログラム開発の際の最大の困難とも言えるが，問題の形式的仕様が与えられた後段でも，仕様から特定のコンピュータ環境にあったプログラムに至るまでのギャップは大きく，困難な作業をともなう．特にここでは効率的に最適なプログラムを生成する難しさがある．現在のソフトウェア開発はどちらも人手によってなされている部分が多く，自動化の主たる動機は開発コストや生産性の改善にあったが，要求される情報処理の量，規模ならびに複雑さが急速に増大し，すでにこのような人手による方式で対応できる限界に近づいている．今後は情報の管理技術や処理技術に全く新しい方式を開発し，プログラミングにおける自動化を極力進めないかぎり，一層大型化するニーズに対応すること自体が困難になることも予想される．本稿ではこの問題をプログラミングにおける人工知能導入の可能性という立場から考えてみる．ただし，この問題はこれまでに提案されてきた各種システム[9)~12),14),17)]に多かれ少なかれ取り入れられ，その報告も多数なされている．またプログラミングにおける各種アプローチを広義のプログラム変換と捉え，そのかなり網羅的な解説が文献16)でなされている．さらに知識の利用に関して，優れた解説記事が本特集にも含まれているので，重複は避け，本稿では知識処理をさまざまな問題に適用してみている立場から，筆者らの考え方を述べてみたい．特に従来の自動プログラミングの研究は仕様が与えられて以後の変換に関するもの，あるいは完全な仕様を書き下す方法や言語に関するものが多いが，本稿ではプログラム開発支援として仕様作成を含む全過程の自動化あるいはそのコンピュータ自身による支援の可能性について考えてみたい．

2. 自動プログラミングとは

自動プログラミングに関する議論には，要求分析，仕様記述，自然言語によるプログラミング，論理，部品化，プロトタイプ，モデルなどの諸概念，ソフトウェア工学に基づく各種開発手法や変換手法，さらに人工知能が加わって賑やかである[1),13),16)]．現在問われているのはプログラム開発の総合的な意味での改善であるから，ここでは自動プログラミングの範囲をそこまで広げて考察する．

一口に自動プログラミングといっても，たとえば科学計算用のたかだか数回の使用で使命が終わるプログラムと，事務処理システムのように一定の環境のもとで繰り返し用いられるプログラムでは評価基準が異なる．前者は一般のユーザが問題解決のためのプログラムを，いかに早く，容易に作るかに重点が置かれるが，後者は生成されたプログラムの効率が重視されるので専門職としてのプログラマの仕事とされ，プログラミングの容易さという面の比重は相対的に低かった．しかし今日の自動プログラミングの重視はこの傾向への反省であり，後者においてもソフトウェア開発の早さ，容易さの面を重視する傾向を表している．したがって自動プログラミングの基本的方法論としては両者の相違は減少し，開発過程での重点の置き方や程度，あるいは開発支援システムの規模の差になってゆくものと考えられる．

次に自動プログラミングをコンピュータ支援による

† A View to Automatic Programming from Artificial Intelligence by Setsuo OHSUGA (Institute of Interdisciplinary Research, Faculty of Engineering, The University of Tokyo).

†† 東京大学工学部境界領域研究施設

プログラム開発環境として考える．ユーザにとってコンピュータは目的をより良く達成するための手段であるから，もし目的達成のために，経済性まで含めた総合的判断のもとで，より優れた手段があれば，それによって代替されるものである．しかし現実には情報処理の手段として能力の点でコンピュータに対抗できるもの，というよりコンピュータが目標とするのは人をおいて他になく，理想は人と同じように機能するコンピュータを開発することである．環境としての自動プログラミングシステムの究極の目的もそこにあるものとしよう．この面で自動プログラミングの意味を明確にするために，情報処理の機能とそれに依存して定まる手順について，人とコンピュータを比較してみることも必要である．

人が目的を達成するために他の人に仕事を依頼する場合，まずよりよいコミュニケーションが重要である．目的を達成するためにはまず相手に何をどのように伝達すれば良いかが問題であって，しかもできるだけ容易なものであることが望ましい．人間と一口に言っても，相手が状況を理解している人と新人とではコミュニケーションの労力は異なるし，自国語を理解しない外国人の場合も異なるであろう．この理想的な形態は，良く状況を理解した，優れた能力をもつ人の場合であることは言うまでもない．この状況は相手がコンピュータであっても同じである．コンピュータが上記の理想的な状況と同様にユーザの目的を理解し，それを達成するように行動すること，そこに至るための人の努力が最小であることが望ましい．

このような理想とするシステムに近づくために，少なくとも2つの要素が重要である．1つは人が要求や目的を記述するために利用できる優れた手段，すなわち広義の言語，もう1つは相手が状況をどこまで理解しているかの程度，すなわち知識である．この2要素は関連はあるが独立の問題である．

さらに，コンピュータがこれらの機能を備えているという前提の下で，プログラムという対象を作り上げてゆく手順や方法，利用可能な手段の利用法などプログラム開発を容易にするための方法論を確立する必要がある．その上で，上記機能を効果的に使ってこのような方法論を実現するためのシステム構成を求める必要がある．

最後に，自動プログラミングは何を目指すべきかを考察することによって問題をさらに具体的にしてゆきたい．コンピュータを使いやすくすることが目標であるなら，一部のコンピュータの専門家のみでなく，少なくともコンピュータを必要とする程度の問題意識をもった人はすべてユーザになる．しかしユーザにとっては問題の解を得ることが目的であってコンピュータはそのための手段であるから，できることならプログラミングに煩わされることなく目的を達成することが理想的である．このようなシステムとして期待されているのが知識処理であり，システム自身が内部で自動的にプログラムを生成することにより，ユーザをプログラミングから開放することを目指している（図-1）．このためのシステムは多数のプログラム群を部品としてもつほかに，各プログラムが行う処理の内容を表す知識，問題解決のための知識を，前項で述べた入力解析の知識とともにもち，これら知識に基づいて，与えられた問題を解いて結果を求めるように部品プログラムを組み合わせて実行する．

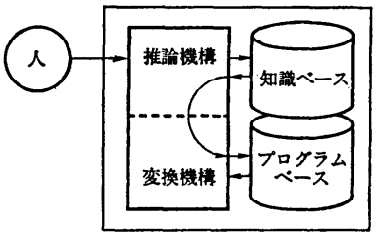


図-1 知的問題解決システム

しかし，このような形態のままでは自動プログラミングではない．自動プログラミングと言うときはコンピュータが目的とする機能を果たすプログラムを生成し，出力することとする．上記知識処理方式でも，単に問題の解のみでなく，それを求めるためにシステムが生成した手順を出力として得られるようにすれば自動プログラミングシステムとなる．後に述べるように，設計・開発向きの知識処理は本質的に自動プログラミングを指向しており，この方式は今後の自動プログラミングの重要な方向を示している．

プログラミングには，問題を記述するために人が通常用いる言語と，計算機言語という全く異なったセマンティクスをもつ2つの言語間の翻訳が含まれるが，一般ユーザ向けの知識処理については，問題の記述を解釈して，要求機能を果たすプログラムを部品プログラムを用いて構成する方式にするのが実現性が高い．したがって，この方式を効果的に利用するには，良い部品プログラム群を用意する必要がある．プログラミングを容易にするために，部品プログラムの集合は実

世界で問題を表現する上で必要な基本概念に対応するものをすべて含んでいることが必要である．部品としては，コンピュータ機能に基づいて定義される概念（たとえばソーティング）は領域によらない共通部品となるが，それ以外は問題領域に応じて必要な概念が異なるので，領域依存である．しかし，狭い領域に依存するのではなく，事務処理分野，化学分野，機械分野などのような大きな領域が単位となって，ちょうど各分野でこれまで便覧や規程書といったものが作られてきたのに対応して，概念の整理とプログラムの部品化が行われることが望ましい．このような性格から，何を部品とするかは領域ごとに実世界で問題定義に用いる言語によって判断されよう．しかる後，これら領域ごとに作られた部品プログラムが同一システム内に集大成されることによって，複数領域にまたがる大型問題の解決システムも実現しよう．このような体系化あるいは標準化は早期に進められるのが望ましいが，そのためには知識処理によるプログラミング支援の技術をまず確立することが先決である．

一方，部品プログラムの開発はかなり人に依存し，専門家によって行われるのが望ましい．部品プログラムの効率が全体の効率に影響するので，ある程度固定的になった概念は小さな部品の合成とするより手続き的にまとめて表して最適化したほうが良いが，何を部品とするかの判断や，処理内容と計算機構との関連で最適なプログラムを作成するのは高度の経験と技量を要するからである．この目的には従来の各種自動プログラミングシステムないし変換システムを組み込んだ，プログラム開発者向きの知識処理システムが利用できるものと期待される．これらの変換システムなどで扱えるのが比較的小規模プログラムであること，効率化を重点に置いているものが多いこと，出発点から手続き的プログラムで扱われ，実世界の表現を受けてプログラムに自動変換するものは少ないこと，などの理由による．

今後は大半の応用プログラム開発は知識処理方式によってユーザ自身が行い，専門職プログラマは優れた部品プログラムの開発と，知識処理方式によるプログラム生成に有効な知識の開発を行うという形態に移行してゆくことが期待される．この形態は実現の可能性があり，かつソフトウェア危機を回避する有効な方向に思える．

3. 問題の記述とモデル構築

自動プログラミングにおいて最初の問題は目的を明示的に正しくかつ完全に述べることである．ここで完全な記述とは，必要なプログラムが生成できるだけの十分な情報を含み，かつそれを阻害するノイズを含まない形の記述とする．これは仕様記述としてしばしば論じられ，多くの場合，完全な仕様記述の存在ないしその可能性が自動プログラミングの基本条件とされてきた．しかしこれではプログラム開発問題を根本的に解決することにはならない．多くの人にとって問題を明示的に正しく表現するというそのことが困難だからである．相手が人間の場合，記述が完全であるか否かは被依頼者自身がそれに基づいて作業手順を作成できるか否かに応じて，質問や返答の形でフィードバックされるので容易に確かめられる．記述内容の正しさ，すなわち解くべき問題を正確に表しているかどうかは，中間での確認を含め，依頼者からの問い合わせやテストによって確かめるほかない．通常，複雑な問題の正しい完全な記述は，一度にすべての情報を与えるのではなく，対話を通して，途中で確認しながら逐次的に情報を与えることによって詳細化を進めるのが最も確実な方法である．逐次的な情報の付加，削除，更新によって正しい内容にしてゆくという方法は多くの人が日常的に行っていることである．相手がコンピュータであってもこの状況は変わらない．ただし適切な知識表現を用いることにより，系の論理的チェックをコンピュータ自身が行う可能性がある．これは人には不得手な作業であり，コンピュータ化の大きな利点の1つとして，今後一層大型のプログラム開発を可能にする重要な条件でもある．

上記の問題記述は用いる言語の問題にも関連する．人が用いる言語は詳細化に適している．コンピュータでこれを可能にするには，途中の検証や情報付加による詳細化が，記述の内容や複雑さに関係なく行えるようにすることが必要であり，それには，コンピュータ自身で，明示的に問題を表現して体系化し，それを管理することが必要である．このような記述の体系を問題のモデルと呼んでおこう．プロトタイピングもこの一環として位置付けられる．すると次の課題はこのモデルをいかに表現し，処理するかである．これについては知識表現として 7. でも触れるが，このようにして検証されながら逐次的に形成され，変換された最終モデルが仕様に相当する．

このように，プログラミングは問題のモデル形成と，モデルからのプログラム生成の2段階からなるという認識はすでに一般的なものであり，これまでのソフトウェア開発も例外なくこの認識の上に行われてきた．しかしこれはコンピュータプログラミング以前に，電気・電子，航空，機械，化学，金属など歴史の古い分野でずっと以前から新規開発に共通する普遍的な方法論として確立されていたものなのである．いかなる分野，いかなる対象であれ，新しいものを開発する際は，まず要求を満たすモデルを作り，ついでそのモデルを実現する手順を作るという過程を辿る．前半を設計と呼び，後半を製作とか合成などと呼ぶ．逐次的に情報を与えることによって詳細化を進めながら，随時，必要な観点からのテストを行うという方法によって，大勢の人が共同で同一目標に向かって協力しながら，遂には個人ではとても達成できない大規模な設計を実現することができる．このような開発の方法論は多くの分野がその歴史の中で，ほぼ独立に確立してきたものである．それがほとんどすべての分野において共通の方法論に至った事実が，この方法の普遍性を表している（図-2）．

前半の設計部分と後半の製作ないし合成部分の作業は異質であるが，これらの分野では全体としては両方にほぼ同程度の重み付けを行っており，完全な仕様の存在を前提として議論するということはない．むしろ最近は多くの分野で，前半の設計の中でも概念設計など上流部分の自動化に努力が向けられてきている．その理由は，これを進めないかぎり本質的な進展が期待されないという認識と，これまでの技術ではこの部分の自動化が困難であったが，情報処理技術，とりわけ人工知能が発達したことにより，この可能性が生じたからである．これらの分野では対象に依存して開発の方法論がある程度定式化しているので，人工知能の導入の目的も明確であり，効果が期待される．

プログラム開発も開発問題として他の分野と本質的な相違はない筈である．たとえば機械設計を例に取ろう．機械設計では要求を満たす対象を実現することが目標であるが，仮に設計・製造支援システムの範囲を，与えられた要求記述から，それを満たすモデルの構築，そしてそのモデルを加工するための NC 機械用データあるいは FMS の駆動プログラムの生成までとしてみよう．これは合成段階で用いられる機械が異なる点を除けば，目的とする機能を果たすプログラムを出力するシステムであって，プログラム開発と基本において変わらない．機械その他の分野での開発問題とこれまでのプログラム開発の相違は，前者においては個々の対象分野が限定できるので表現がきわめて具体的で，分野ごとの言語や表現法が定められているのに対し，後者の場合，多様な分野を対象とし，しかも対象分野と関わりなく定められた汎用的な言語で表現するという制約がある点である．このためプログラム開発においては，入力は，原問題が前処理によって領域依存性がなくなるまでに抽象化されたものとなっている．しかしプログラムでも最初から対象が抽象的である筈がない．プログラミングを必要とした動機は，それぞれの分野に固有の言語で具体的に表現された現実の問題にある筈である．数学者が数学の問題を解くためにプログラミングを必要とするような特殊な場合でも，数学世界では数学言語によって表される問題が具体的表現である．このように他の分野では，原問題表現から開発が始まっているので開発プロセスの目的と範囲が明確に定義されている．それに対し，プログラム開発では，最終結果であるべきコンピュータ言語，もしくはそれと1対1で対応するという意味で実質上コンピュータ言語と同レベルの言語，が初期の段階に現れてくる．これがプログラム開発プロセスの定義を不明確にし，プログラム開発だけが他のものとは異なる開発方法を要する特殊のプロセスであるかのような印象を与える．しかしプログラム開発も本来は他の開発問題と同様であって，要求記述の段階から逐次的にモデルを構成する方式を検討する必要がある．そして他の分野と同様に，プログラミングでもこのプロセスを確立した上で人工知能を初めとする新手法を導入し，できるだけ早く自動プログラミングを実現する努力が必要であろう．

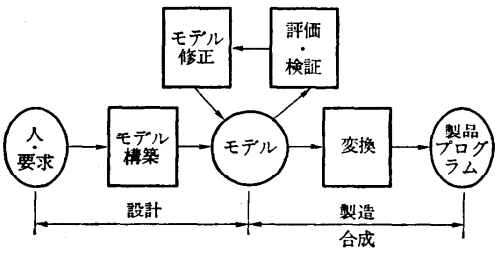


図-2 設計・開発プロセス

4. 言　　語

上記モデル化方式をコンピュータ上で実現するにはモデルそのものとその変換プロセスを明示的に記述せ

ねばならない．これは言語の問題である．

プログラミングを困難にしてきた１つの理由は，計算機用の言語が特殊のものであって，実世界で人が用いる言語とは異質のものである点にある．これら両言語は，一方は主として宣言的言語，他方は手続き的言語というように，シンタックスとセマンティクスの両方で全く異なり，意味を厳密に保つ言語間の変換が困難である．シンタックスのみを似せてもあまり効果がない．近年，自然言語によるプログラミングが試みられているが，語単位あるいはたかだか短い文単位で，手続き型言語との意味の対応が直接取れるような自然言語構文に限定される場合が多い．結果的に，全体の構造としては，プログラムは文に対応するマクロ機能を要素にした手続きとして表現されることになり，プログラミングの本質部分は従来とあまり変わらなくなる．このような擬自然言語は高級手続き型言語と見なすべきものであって，現実の問題の表現レベルとこの言語によるコンピュータへの表現レベル間に大きなギャップが残っている．多くの人にとって煩わしいのは現実の問題を手続きという異なった形式で表さねばならないこと，すなわち，問題の完全な記述をしながら，セマンティクスの異なる言語間の変換をせねばならないことである．これを一度で行うのでは困難なので，多くの分野でモデルを中間に置いて，問題の完全な記述を作る過程と，それをプログラムに変換する過程を分離すること，しかもその変換も何段階かに分けて行うという方法が確立されてきたのである．それには問題の記述言語とプログラム言語の中間に，モデルあるいは仕様を記述するための第３の表現形式が必要になる．この表現はコンピュータ処理が可能であると同時に，これで表現されたものは人にとって理解しやすいものであることが望ましい．するとプログラミングの手順は，逐次的に情報を与えて詳細化を進めながら問題をいったんこの中間の表現形式でモデル化し，そこで記述の正しさや完全性を確認した上で，これを一連の変換操作によってプログラム言語に変換する，というものになる（図-3）．したがって，この中間言語は柔軟であって，人の使う表現を受け取り，何段階かの変換をこの言語表現内で行って，最後にプログラム言語に変換ができるものであることが要求される．このようなプログラミング方式を想定すると，プログラミング問題は，（1）このような中間の言語はいかなるものか，（2）モデルをどのように表現するか，（3）人が与える問題記述をどのようにモデルに変換するか，（4）モデルをいかにプログラムに変換するか，などの，より具体的な問題に展開される．

制約的な自然言語でも，ユーザがあまり違和感をもたないで効果を発揮している例がある[6]．これは金融機関における日本語プログラム開発システムである．これが効果をあげているのは銀行ないしそれに準ずる金融機関の特殊性による．銀行業務では組織および作業手順が早期に確立され，かつ銀行法によってほぼ固定されてきた．その手順は規程書として詳細に定められた．すべての従業員はこれを熟知し，これに基づいて日常の作業を行っている．いわば規程書は人間を対象とした仕様書であり，この規程書に従って使われる日本語を計算機がプログラムに翻訳することができれば，日本語でプログラムすることができ，関係者はすべてプログラムを理解することができるようになる．木村[6]はこれを実現し，現場においてすでに大きな効果をあげていると報告している．

この方式が効果をあげているのは，（1）関係者がすべて同じ目的をもち，（2）個人差が結果に現れないように処理手順が定形化され，（3）しかもそのための教育が組織的に行われているからである．すなわち教育によって実問題がすでに機械向け仕様書のレベルで表現され，計算機理解のレベルとのギャップが少ないためである．分野を特定できない場合，あるいは特定できても仕事の内容が変化するため作業手順をここまで規格化できない場合，あるいはそれができてもすべての関係者をこれに習熟するまで教育できない場合には，これが効果をあげることは期待できない．

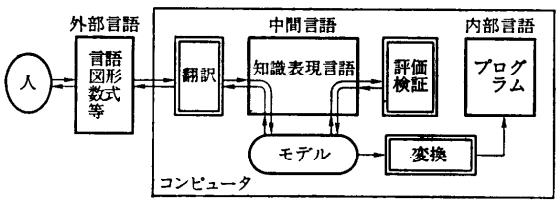


図-3　プログラム開発支援システム

変換方式でみればむしろ従来開発されてきたものよりずっと簡単なものであるにもかかわらず，あえてこの例を取り上げたのは，このシステムがすでに実用レベルで効果をあげていることの意味を明らかにするためであり，上記の議論との関連で示唆に富んでいるからである．規程書が作られていることは，この分野では業務内容が固定しているので，モデルの形式が厳密

に定められていること，しかもこのモデルは自然言語との対応が個々の短い文単位で付けられる程度にまで規格化されていること，教育によってすべての関係者に教えられるのはこのモデルの表現形式であること，などである．

これは分野が限定されていることによって，例外的にモデル表現の形式化が進んでいる場合であるが，開発作業においてモデルの形式を定めることが重要であることを示している．前記の各種技術分野では例外なしに固有のモデル表現形式を定めているが，モデル表現の形式を決めることが各分野の発展を可能にする必要条件と言うことができ，各分野でこれを決めるために多くの努力があった．プログラミングがこれら諸分野と異なる点は固有のモデル形式が定められていない点にあり，モデルをどのように定義し，位置付けるかが自動プログラミングの課題となる．

知識処理では上記中間言語を知識表現言語という．一方，人が自己の意図を表現する手段として用いるのは自然言語のみではない．意図を正しく表すことは人にとっても容易なことではなく，そのためにそれぞれの分野でさまざまな方法が開発されてきた．電気回路図，機械設計図，制御回路図，分子構造表現，PARTダイアグラムなどである．図形や画像，グラフは分野によってはとりわけ重要であり，数式，表なども用いられる．外部言語としてはこれらを含めておかねばならない．このことは中間の言語がこれら各種の表現媒体を等しく受け取り，統合的に表現し得るものであることを要求している．また，プログラミングにおいては最終結果はプログラムであるから，中間言語は手続きへの変換が可能なものでなければならない（図-3）．

5. 知　　識

自動プログラミング方式を実現する上でコンピュータの支援が不可欠であるが，今日のコンピュータの情報処理方式では開発型問題の支援能力に限界がある．設計や開発のように試行錯誤を基本として含む仕事と，手続きによって前もって行うべき仕事を定義しておく情報処理方式は基本の性質が異なる．自動プログラミングに人工知能が期待されるのは，設計や開発が要求する機能が人工知能，とりわけ知識処理の処理機構にマッチする点にある．

知識処理では短い（モジュール化された）宣言的な文で表された知識を，推論を通して問題解決に利用するという構造をもつ．具体的には知識とのマッチングによって入力を変換する．これが目標を含む多くの候補の中から試行錯誤を通して目標を見いだすという開発型問題の枠組みに適合する．またモジュール性の高い言語で記述されたモデルは情報付加によって詳細化するのに適していること，評価のためのモデルの変換やモデルの修正など小刻みな操作に向いていること，さらにこの評価やモデル修正に経験や知識を適用できること，知識表現を適切に設計することによって論理的なチェックが可能になることなど，多くの利点をもつ．

厳密に言えば，機構面で知識処理を現行のコンピュータと対比することは正しくない．これまでの知識処理のシステムは現行方式のコンピュータの上にソフトウェアシステムとして実現されているから，両方式とも利用する機構は同じものだからである．両者の本質的な相違は実問題の表現と処理の方式，いわば人間とコンピュータとの関係にある．従来の情報処理の方式は，問題を計算機構に適合する形式にまで，外部で，人が変換することを必要としたのに対し，知識処理は実問題とそれを機械的に処理する計算機構の間にあって，その変換機能を果たす．

知識処理の機能をソフトウェアによって実現できるから，知識処理の機能を含む自動プログラミングシステム（厳密にはプログラム開発支援システム）をソフトウェアシステムとして実現することも可能な筈である．事実，これまで開発されてきたほとんどの変換プログラムはカタログとか生成規則と呼ばれる変換規則を仕様記述やソースプログラムに適用して，目的とするプログラムを得ることを試みるが，この原理は知識処理と類似である．しかし知識処理のように，推論を独立した機能として定義しておくという意識はもたず，あくまで変換機構を埋め込んだ手続きとして実現されている．独立の推論機能をもつか否かによってシステムの動作は非常に異なる．たとえば CIP は優れた変換規則をもつシステムであるが，変換の適用はユーザに任され自動的な推論は行わない[16]．一方，知識処理では知識と推論を基本構成要素とし，システムの動作自体が知識に基づいて制御されるので，環境に適応した柔軟な動作を可能にする．すでに述べたように，知識処理の推論機構，情報の管理などの機能部分はサブシステムとして手続き的プログラムで実現されているが，実問題への応用にさいして，知識処理は推論に基づく問題解決手法を用いるという点で，情報処理の基本メカニズムが従来のコンピュータ方式と基本

的に異なっている．実際的には，まだ自動推論機能の能力が低いためシステム概念が確立されているとは言い難く，困難の多い変換は最終的には人の責任においてなされねばならないが，知識処理を前提として体系化しておけば，後にシステムの改善や推論能力を向上することによって現行方式の限界を越えることが期待できる．

知識の役割は単にモデルの変換に留まらない．知識処理方式の重要な特徴の１つは人とのコミュニケーションが容易になる点であるが，ここでの知識の重要性は **2.** で述べたとおりである．これを含めて，知識処理ではすべての動作が知識に基づいて進められる．また知識がそのように表現できねばならない．

6. ソフトウェアモデル

ソフトウェア開発の目的で，知識処理の中間表現すなわち知識表現言語によってシステム内に表された問題の表現をソフトウェアモデルと呼ぶことにする．実際には１つのモデルがあるわけではなく，分野に応じてそれぞれの表現があり，その総称である．また１つの分野の中でもプロセスの進行に応じてモデルが変換され，変形ができるので，それらも一括した呼び方である．ただし処理が後段に進むにつれ，分野による表現の相違は減り，代わってすべての分野に共通のコンピュータ言語に変換されてゆく．自動プログラミングをこのような過程として概念的に図-4 のように表す．上段は実世界での処理を表し，与えられたシステムへの入力が処理されて出力される．入力の型の種類は有限で，システムはそれぞれに応じて変換処理を施して出力に変える．

下段はそのコンピュータ内表現である．コンピュータへは実システムと同じ入（出）力が，コンピュータ向けに変換されたものが入（出）力となるが，それに対し実システムと等価な関係となるようにプログラムが作られる．プログラミングは上段のシステムから下段のシステムへの変換である．この過程はまず実システムを情報表現してモデルとし，次いでモデルを変換していってコンピュータプログラムに至る．初期の情報表現から最後のプログラムへの変換に至るまでに何段かの変換が行われる．これは宣言的表現から手続き的表現への大きなセマンティクギャップを埋めるための手順として取らざるをえないプロセスである．実システムの直接の情報表現（第一モデル）から手続きへの変換の直前までをソフトウェアモデルとする．自動プログラミングはこの過程の自動化である．これまでの変換プログラムは主としてプログラム化されたあとの，手続き表現内の変換であったが，上記のセマンテ

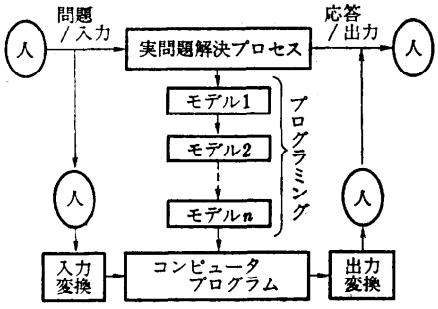
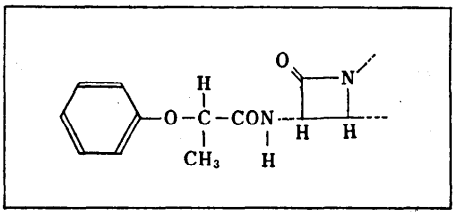
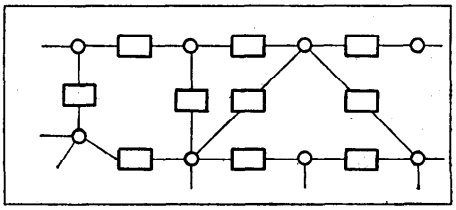


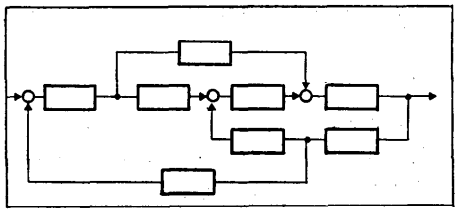
図-4　プログラミングの手順



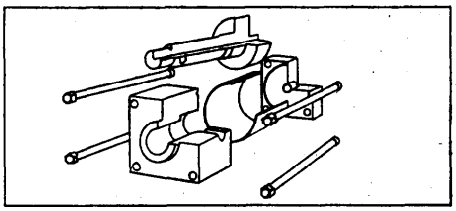
（a）分子構造モデル



（b）ビジネスモデル



（c）制御モデル



（d）機械モデル

図-5　各種モデル表現

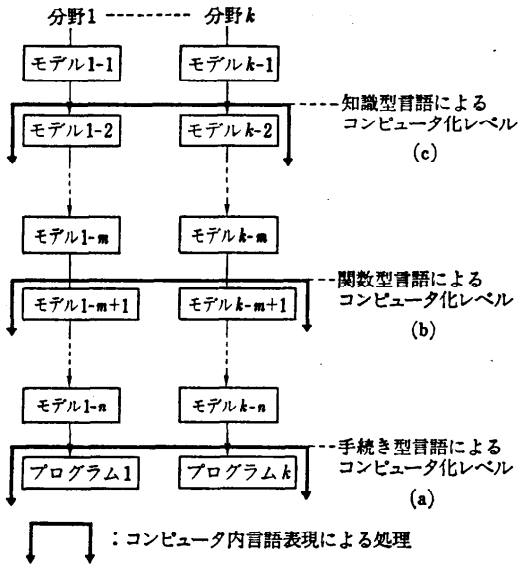


図-6　モデル記述言語とコンピュータ化のレベル

ィクギャップを越えるところに真の難しさがあるとしたら，このモデル変換の方式こそ確立すべきものである．

第一モデルに近いほど，表現が領域依存になる，図-5 はそれらの数例を示す．図-5 では繁雑さを避けてモデルの構造のみを示しているが，実際にはここにさまざまな属性などの記述が加わってモデルを表現することになる．モデルの表現が異なるため，当然，それからプログラムに至る変換過程も対象によって異なってくるが，ある段階までくると，領域に係わらない共通の表現が現れる．従来はこれが手続き型表現であったが（図-6（a）），これより高いレベル，すなわち原問題により近いとされるのが関数型表現である（図-6（b））．今日，これよりさらに現実の問題に近い表現として期待されるのが述語言語である．しかしこのための言語としては，図-5 のような多様な形態をとるモデル構造を表すために，データ構造が重要になり，述語論理の体系をデータ構造を含むように変更することが必要になる．分野による表現の相違はモデルの表現形式とその意味付けの相違であるが，このような言語を用いるなら，モデル構造はこの言語によるデータ構造の相違に過ぎず，その意味付けは述語によって表されて知識として登録しておき，推論によって意味の変換が行われるので，第一モデルの表現を直接この言語で受け入れることができる（図-6（c））．第一モデルは問題の定義として人が与えねばならない部分であるから，このことは形式一意味関係を知識として与えておくことにより，領域ごとの特殊性を残しながら，システムとしては領域に非依存のシステムの実現が可能になることを示している．

問題表現から最終の手続き的プログラムまでの変換過程はセマンティクスの全く異なる 2 言語間の変換であるが，手続き的表現は与えられた問題の解の表現であるため，この変換は実は問題解決過程に他ならない．したがって自動プログラミングは分野ごとにこれまでの研究によって求められてきた問題解決の各種手法を知識として蓄え，探索的に解を求めながら最後にそれをプログラムとして表すことが要求される．この知識はモデルとのマッチングによってこれを変換するものであるから，その言語はモデルを表現する言語と同じものであることが必要である．すなわち知識表現言語はモデルそのものと，問題解決の方法をすべて表現できるものでなくてはならない．上述のデータ構造を含む述語はほぼこの条件を満たしている．

図-7 はこの実例として制御分野での解析問題を示したもので，われわれが開発してきた KAUS システムを土台に，関傑波[5] が開発したシステムである．これは制御系設計支援システムとして，入出力間の応答を求めるプログラムを生成して実行し，結果によってモデルそのものを変更するという目的で開発されたもので，自動プログラミングとは少し目的が違うが，第一モデルからプログラム生成までの変換過程は同じである．図-7（1）は第一モデルであるブロック図の表現（a）がシグナルフローダイアグラム（b）に変換され，これに基づいて全局所ループおよび開経路の伝達関数の集合（c）が求められ，これが組み合わされて全体の伝達関数（d）が求められ，この結果を要求（入力）と組み合わせて解析し，出力を求めるプログラム（e）が生成されるまでの過程が知識に基づいて行われる様子を示す．図-7（2）は実行例である．この過程において問題解決手法として制御分野で開発されてきたさまざまな知識が用いられ，異なった知識源を構成する．知識処理の手法のもとでこれらを知識として用いることにより，プログラミングの専門家に頼らず，分野の専門家が使えるプログラム開発支援システムが得られる．これはまだ単純な例であるが，方式として 1 つの可能性を示すものである．

7. 知識表現と知識処理システムアーキテクチュア

知識処理を用いて自動プログラミングを実現するには知識が重要な役割を果たす．これは前述のように，数学の基本公式のような基本知識，各部品プログラムの内容を表す知識，問題解決のための知識，入力解析の知識などをもち，外部言語の理解とそれによるモデルの構築，問題解決手法の適用によるモデルの評価と修正，プログラムへの変換，最適化規則の適用など，最初から最後まで知識をベースにして処理が進められる．このような知識に基づく処理が理想的に行えるか否かは前記中間言語としての知識表現言語と知識処理システムアーキテクチュアの設計に依存する．

知識表現言語の重要性は，これによって知識処理の実質的な部分をすべて記述せねばならないことから明らかである．図-7 に知識として表現されねばならないさまざまの形式の一端が示されているが，これからも明らかなように，モデルの表現には多様な構造の表現が現れる．知識としてこのようなモデルそのものを表現できねばならないが，さらに重要なことはその変換をも知識によって表現することである．すなわち知識は異種構造の変換規則の表現ができねばならない．前記 KAUS はこれを考慮して開発してきたものであるが，紙数が尽きているので文献 2)，8)，15) を参照されたい．

最後にこれも重要なこととして，知識処理システムの論理構造に言及する．知識処理の本来の目的がヒューマンインタフェースを良くすることにあるが，実問題は宣言的な表現を中心になされるのに対し，実際の処理は手続き的に表現された言語で行われる機構になっているので，知識処理システムはそれが存在する根源的理由においてセマンティクスの異なる2種の言語を扱うことになる．そしてシステム内でのこの両言語の関係がシステムの性格をきめる．実際の処理は手続き的に行われるが，ヒューマンインタフェースを良くするためには，宣言的な表現のレベルで基本的な処理の構造がすべて定められ，手続きの部分はその下で宣言的表現の評価機構として隠されていること，言葉を換えると，宣言型表現レベルと手続き型表現レベルは前者がマスタ，後者はスレーブの関係になるようにシステムが構成されねばならない．これを実現するには知識表現言語が現行コンピュータの管理機構までも宣言的に記述することができるだけの記述力が必要である．今日の多くの知識処理システムは，見掛けはともかく，実質上，この関係が必ずしも望ましい姿になっていない．知識処理が大型の問題に対して実用化されていないのも，このような本質的な部分に議論が及んでいないことが理由の1つと言える．この問題に関しては 3)，4) を参照されたい．

これまで実問題は宣言的に表現されるものとして議論を進めてきた．このことは実問題がすべて静的な表現のものであることを意味するものではない．事務処理のようにトランザクションの処理手順が問題の定義になっているものもある．しかし設計に際して通常，この流れの全体がふかん的に見えるように，図-5 のような図的表現がなされる．この図において処理順序はノード間の関係に置き替えられるので，図の作成手順は任意であり，この意味で宣言的な表現として扱える．

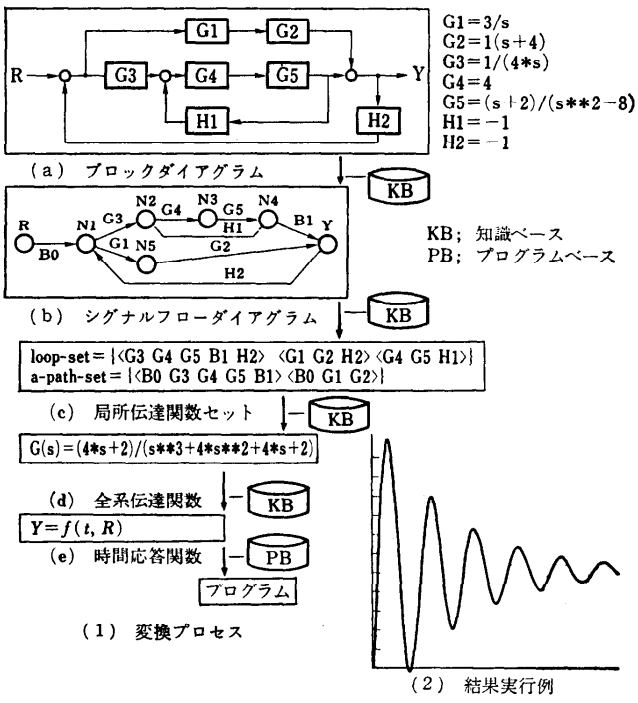


図-7　変換例（制御モデル）

8. むすび

自動プログラミングは単にプログラミングの一部をコンピュータで支援することで満足すべきではない．もしそのような発想で始めたら，結局，部分の自動化すら十分になしえない結果に終わるであろう．自動プログラミングを行うシステムには，自動化の対象となるプログラムより高度の表現能力とその表現に基づく処理能力が要求されるが，これまで行われてきた自動化の発想は対象のプログラムと同レベルである従来のプログラミング技法によって自動化を図るものであり，必然的に部分の自動化であった．本稿ではこれに代わるものを人工知能，とりわけ知識処理に求め，その可能性や条件とともに実現の方向を探った．これを実現に至らしめるには知識処理自体の強化が必要であり，まだ時間が必要であるが，筆者らは基礎的な実験を通して，実現の可能性を信じている．

参考文献

1) 鯵坂恒夫，阿草清滋，大野 豊：ソフトウエア設計自動化のための仕様記述言語，情報処理学会論文誌，Vol. 27, No. 7, pp. 173-181 (1986).
2) 大須賀節雄：多層論理—モデル記述の為の述語論理，Proc. of the Logic Programming Conference (1983).
3) 大須賀節雄：人工知能システム KAUS の論理構造，1987 年度人工知能学会全国大会.
4) 関 傑波，大須賀節雄：情報システムのあり方，ibid.
5) 関 傑波，大須賀節雄：図形によるマンマシンコミュニケーション，ibid.
6) 木村俊一：日本語プログラム，ibid.
7) 佐藤雅彦，玉木文夫：論理型プログラムの等価変換とプログラム合成問題への応用，情報処理，Vol. 25, No. 11, 創立 25 周年記念論文 (1985).
8) 山内平行，大須賀節雄：KAUS-4 の処理系について，ソフトウエア科学会，第一回大会論文集.
9) Balzer R.: A Gloval View of Automatic Programming, Proc. 3rd International Joint Conference on Artificial Intelligence, pp. 494-499 (1973).
10) Balzer, R.: Transformational Implementation: An Example, IEEE Trans. Software Engineering SE-7, 1, pp. 3-14 (1981).
11) Darlington, J.: An Experimental Program Transformation and Synthesis System, Artificial Intelligence, Vol. 16, pp. 1-46 (1981).
12) Green, C., Gabriel, R.P., Kant, E., Kedzierski, B.J., McCune, B.R., Phillips, J.V., Tapple, S.T. and WestFold, S.J.: Results in Knowledge Based Program Synthesis, Proceedings of 6th International Joint Conference on Artificial Intelligence, pp. 342-344 (1979).
13) IEEE: Special Collection on Requirements Analysis, Trans. on SE, Vol. SE-3, No. 1 (1977).
14) Manna, Z. and Waldinger, R.: Synthesis: Dreams => Programs, IEEE Trans. Software Engineering, SE-5, No. 4, pp. 294-328 (1979).
15) Ohsuga S. and Yamauchi H.: Multi-Layer Logic-A Predicate Logic Including Data Structure as Knowledge Representation Language, New Generation Computing, Vol. 3, pp. 403-439 (1983).
16) Partsch, H. and Steinbruggen, R.: Program Transformation Systems, ACM Computing Surveys, Vol. 15, No. 3, pp. 199-236 (1983).
17) Waters, R.C.: The Programmer's Apprentice: Knowledge Based Program Editing, IEEE Trans. Software Engineering, SE-8, No. 1, pp. 1-12 (1982).

(昭和 62 年 5 月 19 日受付)