ソフトウェア工学　52－10
（1987・2・13）

知識に基づくプログラミングにおける自己適用を目指して

間野　暢興

電子技術総合研究所

プログラム合成システムの提供する機能を用いて、そのシステムプログラム自体を生成することを目標とする．

このプログラム合成システムは、（ジャクソン法が対象とする）ファイル処理、集合や関係などの抽象データ型処理プログラムを合成の対象とするもので、知識ベースを有し、コンテキストデータベースとコルーチンによる問題解決機能を持つ．情報は主に対象物と関係からなる（木あるいはグラフの構造を持つ）モデルの形式で表現され、アルゴリズミックな手続きにより処理が行なわれる．

合成システムの各モジュールは、合成システムが用いる問題定義用フロー構造（の階層）上に、サブモジュール、抽象データ型、データフロー、コントロールフロー、仕様記述などを用いて表わされる．合成システムの用いる知識ベース、問題作業領域、合成システムプログラム内で使用される作業用データ構造、の各々（の構成要素）は、抽象データ型として記述され、システムの知識構造中に埋め込まれる．これらの抽象データ型の操作および各モジュールのプログラム（のモデル）を、合成システムの適用あるいはその部品の機能の利用により生成することを目標とする．

合成システム自体（のモデル）の記述を生成保持することは、システム設計、ドキュメンテーション、あるいはシステム保守に役立つ．また、合成システムの実際的規模の対象例題としても興味深い．

TOWARD THE SELF-APPLICATION IN KNOWLEDGE-BASED PROGRAMMING

Nobuoki MANO

Language Processing Section, Computer Science Division, Electro-Technical Laboratory,1-1-4, Umezono, Niihari-gun, Ibaraki-ken, 305 Japan

Program-synthesis system should have its self-description and be able to generate the system programs from it, using the function of the system itself.

The object program-synthesis system is the one for synthesizing file-processing programs which the Jackson strucured programming method covers and data-processing programs which use abstract data types such as sets and relations. The synthesis system has a knowledge base and problem solving facilities consisting of context data base and coroutine programs. The informations are mainly represented by a model-form consisting of objects and relationships, and processed by algorithmic procedures.

We give the definitions of hierarchical modules, using the flow structure representation of problem definition in the synthesis system. Some of the submodules can be synthesized from the input-output specifications by the sytem itself. Problem-oriented abstract data-type can be gotten by modifying the standard abstract data-type in the knowledge-base of the synthesis system.

1．　はじめに

筆者はこれまで、ジャクソン法のようなファイル処理プログラムおよび抽象データ型を用いるデータ処理プログラムの合成方式の研究を進めて来た[1][2][3]．そこにおいて対象としてきた問題は、比較的小規模の問題であった．本論文では、そのプログラム合成システム自体を対象としてシステムプログラムを生成する問題を考えてみたい．扱う問題はシステム規模の問題であって、これまでの小規模な閉じた問題とは手段を異にする．ただし、ここでとる方式は、上記の特定システムに限るものではなく、一般システムを対象とする場合にも共通するものを目指す．

一般に、システムの処理構造は複雑であり、入出力仕様により仕様を定義するのは、その規模の故に困難である．設計者がある程度まで問題の処理構造をトップダウンにブレークダウンしてシステムに与えることが必要となる．ここでは、ユーザは、モジュール、抽象データ型によるストリームあるいは蓄積型データ構造、モジュールを構成する主な算譜構成要素、および抽象データ型の操作、そしてこれらの間のコントロールフロー、データフローにより、問題処理構造を階層的フロー構造の形式で記述する．そしてこの上に、ストリームの構造、抽象データ型のデータ型不変量、モジュールの入出力仕様記述、などを重畳して表現する．なおこれは合成システムにおいてユーザが問題定義を行なう際に用いる情報表現形式をそのまま利用したものである．

ユーザからの情報を基に、システムは、内蔵されたプログラミングに関する知識と自己記述の情報を用いて、最初に各問題抽象データ型の形成、その後で各モジュールの完全化、を行なう．これらの全段階において、情報は全てモデルの（意味モデルと称する意味ネットワークあるいはフレームのような）形式で表現され、その特性を生かした処理が行なわれる．

実用を目指すプログラム合成システムにおいては、システムが変更容易性、漸増的拡張可能性を備えていることを必要とする．そのためには、システム自体についての記述（すなわち、自己記述）をシステムが所有し、それを基に自身のプログラムを生成あるいは修正できる機能が欲しい．このような機能を備えたプログラム合成システムの例としては、ＣＨＩシステムが挙げられる[4][5]．そこでは、Ｖという言語を設定し、それを用いて規則の形式で全ての自己記述を行なっている．しかし、その表現は、必ずしも分かりやすいものではない．この論文で対象としている合成システムは、対象物と関係からなる統一されたモデル表現形式を採用しているので、分かりやすい．合成システムのプログラムは、木あるいはグラフのような構造の処理を持つデータのアルゴリズミックな処理を沢山含んでいる．そこで、標準データ抽象データ型を知識として蓄えておき、それを変形させることで目的とするプログラム（合成システムプログラム自体）を得ることができるようにする．一般に、部品化プログラミングの機能は、ソストウエア工学における重要な研究課題であるが、その機能を実現するための具体的方法は未だ十分に明白にはされていないように思われる．プログラム合成システムを対象とする自己適用の研究は、そのための興味深い場を提供してくれる．

第２章では、知識に基づくプログラミング支援システムとしての本方式における全体の手順を示す．第３章では、階層的フロー構造による対象システムの記述の与え方について述べる．第４章では、上記階層的フロー構造からの抽象データ型の記述の形成を扱う．第５章では、それらの抽象データ型を用いたシステムモジュールプログラムの生成について述べる．

2．　全体の手順

本方式によるシステム設計とそのプログラム生成の流れは以下に述べるようになる．１）－２）を第３章、３）－４）を第４章、５）－６）を第５章で、各々詳細に述べる．なお、ここでモジュール（およびサブモジュール）とは、抽象データ型の操作を用いたシステム本体のプログラムあるいはサブプログラムを指す．

１）　トップレベルのモジュールおよびその中あるいはその下位階層で使用され

るサブモジュールを、以下のように、各サブモジュールごとにモデル表現によるフロー構造で定義を与える．すなわち、モジュールおよびサブモジュール（プロセスやコルーチンも含む）、ストリーム、蓄積型データ構造である抽象データ型、その操作、状態などの対象物の集まりと、それらの間のコントロールフローとデータフローにより記述する．それにより各抽象データ型（に属するべき）操作が漸増的に集積される．この段階では、操作名は暫定的に、知識ベースにある一般的な抽象データ型のもの（例えば集合の操作「登録」のような）でよい．

２）　モジュールの入出力仕様、状態の記述、各ストリームおよび抽象データ型のデータ構造（あるいは、その内部拘束条件を含むデータ型不変量）、複合データ構造の場合にはそれらの間で成り立つ不変関係、を与える．

３）　対象システムで基本となる自然型の抽象データ型を定義する．合成システムの例では、データの構造を表わす部分空間、およびプログラムの構造を表わす部分空間がそれに当たる．対象システムの用いるデータ構造がその型のデータ構造をその一部として持つ場合には、それを役割型として知識構造に登録する．

４）　各問題抽象データ型を構造の簡単なものから複雑なものへとボトムアップに定める．操作の引数と値を含む入出力仕様を定義する．知識構造の中にある標準抽象データ型を利用できる場合には、一般名、データ型不変量、操作の集まりからそれを選び出し問題向きに書き直す．このとき、システムの用いる関係についての自己記述の情報が利用される．その操作はその抽象データ型に関連した名前となり、一般抽象データ型の操作と上位下位関係で結ばれる．そしてモジュール内で実際に用いられていた操作呼び出しをそのインスタンスにより置き換える．

５）　各モジュールの仕様とプログラムを定める．上記のようにして定義された抽象データ型を用いた各モジュールの内容を決定していく．ＣＡＬＬ関係の階層構造を辿りボトムアップに各モジュールの入出力インターフェースと内容を定めて行く．このとき、プログラム合成システムあるいはその一部の機能を利用して、入出力仕様からそのモジュールのプログラムの生成を行なう．

６）　フロー構造を木構造に編集する．また、データの構造を考慮に入れて、周りのモジュールとの間に値の受渡しによるデータフローを付ける．

３．　階層的フロー構造によるシステムプログラムの定義

モジュールおよび（抽象データ型の）操作は、入力仕様、出力仕様、入力部分、プログラム部分、出力部分からなる．データの構造に基づくプログラムの場合、入力部分および出力部分には入力仕様、出力仕様中のデータの構造分割に基づく表現、プログラム部分にはその入力部分の表現に対応した構造のプログラム表現がおかれる．もっと複雑なプログラムの場合、入力部分および出力部分は空であり、プログラム部分はフロー表現により記述される．図１（ａ）にそのプリミティブを示すように、サブモジュール、（抽象）データ型の操作、ストリーム、状態、入出力仕様の記述、蓄積型データ構造をもつ抽象データ型、を対象物とし、これらのインスタンス間の関係をコントロールフローおよびデータフローにより結ぶ．ストリームにはそのデータ構造の記述を与える．抽象データ型にはデータ型不変量を与える．合成システムの基本部分のプログラム部分は図１（ｂ）、またその内の前処理サブモジュールのプログラム部分は同図（ｃ）のようになる．

問題解決機プログラムの動作を簡単に述べると次のようになる．問題出力仕様（多重ゴールであるとする）に整合する出力部分を持つ操作を知識ベースから取り出し、候補の組み合わせを作る．複数の組み合わせができたときはその一つを取り出し、以下に述べるような手順を基本とする試行を行なう（その他は格納しジェネレータコルーチンにより後で一つずつ取り出して試行する）．

①　候補操作の入力部分を並列に展開する．

②　そしてそれらの間の相互干渉をチェックする．干渉がある場合には、候補の間で順序付けを行なう．

③　またサブゴールどうしのレベルをチェックする．ここでレベルとは、データ構造の木表現において根からその節までの間に繰り返し表現が現われる回数をい

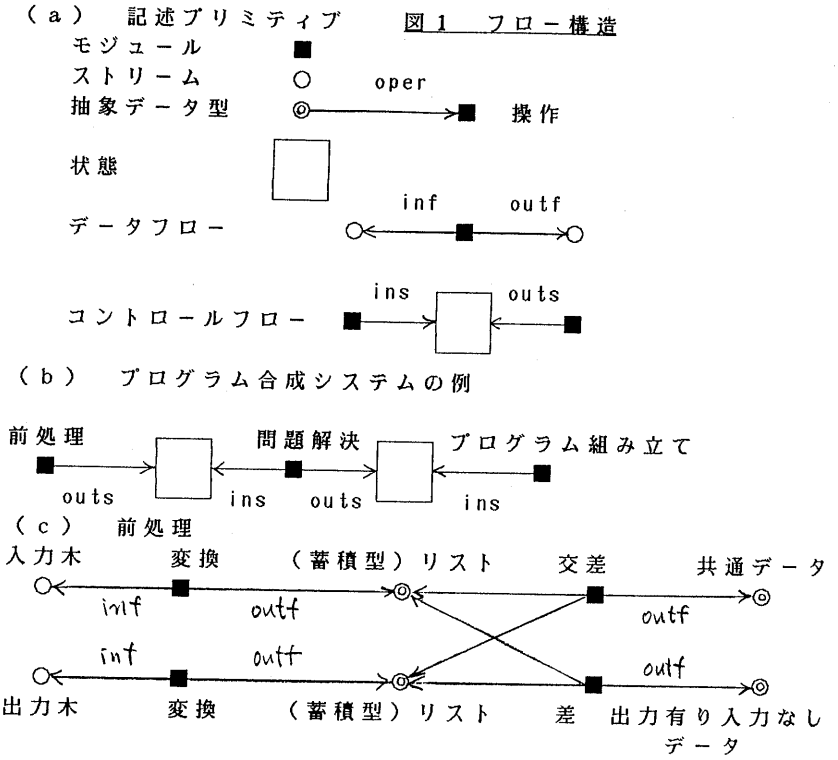
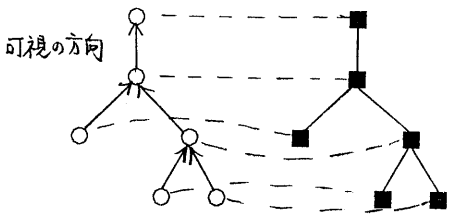


図２　コンテキストデータベース
○：ローカルデータベース　■：コルーチン



う．データ木と対応するプログラム木の節でいえば、これはその節が何重のループ構造のネストの中であるかを表わす．調和がとれるときは、展開された情報の融合を行なう．

④　その結果と問題入力との一致を調べる．

この問題解決機は、制御としてコルーチン、作業データ領域として図２に示すような木構造のコンテキストデータベース（節のデータベースをローカルデータベースと呼ぶ）を使用する．この機構は、古くはCONNIVER[6]、最近ではURANUS 7]からヒントを得ている．木の一つ一つの節には各々コルーチンが対応する．親ルーチンは候補の択一や仮定の場合分けが起こる度ごとに、子コルーチンを新たに生成し、それに対応したローカルデータベースの名前をパラメータとして渡す．あるコルーチンからは、その対応する節と木の根との間の路上に存在するデータベースの情報は見えるが、その他のローカルデータベース中の情報は見えない．また、データ（Ｒ　ａ　ｂ）と－（Ｒ　ａ　ｂ）（－は否定を表わす）がその路上に存在するときは、両者は打ち消し合ってどちらのデータも存在しないものと

する．この機構は、プログラムの合成において様々な使い方が可能であるが、二三の例を下に挙げる．

① 後戻りとしての使い方（上述）．例えば、ジャクソン法の構造一致の問題で、出力木の葉の定義を入力に還元する途中、複数の候補からの択一が必要になる場合がこれに当たる．また配列やポインタなど副作用のあるプログラムの合成を考える場合には、特にコンテキストデータベースの機構は有効である．

② 途中で場合分けの分岐を含む使い方． 例えば、Insert-sort プログラムのような再帰プログラムを入出力仕様から合成する場合を考える．そのサブモジュールであるInsertプログラムの合成を行なうとき、二つの入力の要素間の大小関係に関して二つの場合分けが生ずる．それらを各々別のコンテキストに仮定として持つ枝分かれの環境を作り、各々の条件下で入出力仕様を満たすプログラム合成の問題解決を行なう．

ここでは、全体のモジュールの分解はユーザが行なうことにしているので、一つのモジュールごとに（必要があれば）一つのコンテキストデータベースシステムが使用されるが、これらは本来は相互に関連を持つ筈のものである．その意味で、将来一つのコンテキストデータベースに統合されて扱われるべきものであろう．

4． 抽象データ型の定義

3．で述べたフロー構造により与えられた蓄積型データ構造を抽象データ型としてそれに属する操作を集め、問題抽象データ型としてシステムの知識構造へ埋め込む．フロー構造においてはそのインスタンスを使用するように書き替える．問題抽象データ型は、標準抽象データ型に手を加えることにより作り出す．例題の合成システムで使用するデータ構造としては、以下の三種類がある．

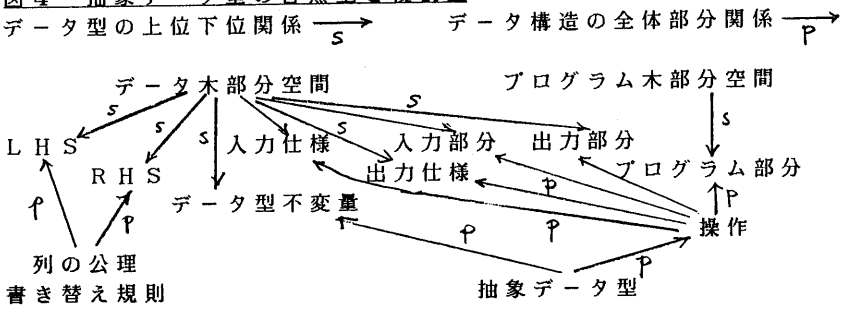
（1） 知識ベース

知識ベースには、対象物および関係についてそれらの上位下位関係からなる知識構造がある．対象物には、現在のところ、データ集合、算譜構成要素、場所集合、識別子、状態の五つの型がある．特にデータ集合に関連するものとして抽象

図3 抽象データ型（自然型）データ木部分空間の操作

整合1： データ木部分空間Xデータ木部分空間－＞束縛リスト
整合2： データ木部分空間X束縛リストXデータ木部分空間
－＞bool
具現化： データ木部分空間X束縛リスト－＞データ木部分空間

図4 抽象データ型の自然型と役割型



データ型（特定の問題によらない自然型と、問題の中での役割を表わす名前で呼ばれる役割型とがあり、役割型は自然型の下位データ型となる）、算譜構成要素に関連するものとして操作、の上位下位関係は知識構造の中心をなす．特に指定のない限り、下位のデータ型は上位のデータ型の、データ型不変量および操作を継承する．合成システムにおいては、自然型としてデータの構造記述（*1）およびプログラムの構造記述（*2）を中心としたそれぞれの部分空間が用いられる．図３に前者の操作群を示す．以下の文中で *1 および *2 印を後につけたものはそれぞれの上位の自然型を示す（図４参照）．各抽象データ型は、データ型不変量(*1)と操作群（列のような一般的なデータ型の場合は更に、ＬＨＳ(*1)＝＞ＲＨＳ(*1)の形式で表わされる公理および関連変換規則群）により記述される．操作は、入力仕様(*1)、出力仕様(*1)、入力部分(*1)、プログラム部分(*2)、出力部分(*1)からなる．問題解決や標準抽象データ型を利用した問題抽象データ型の生成において、部分グラフの整合による選択抽出の能率を上げるために、知識ベースにはその部分空間をキードメインとする索引を用意する．

関係の上位下位関係は、反射律、対称律、推移律などの性質が成り立つか否かによる．これとは異なった分類として、データの構造を表わす関係、プログラムの構造を表わす関係、－－－というような関係の類別を用意する．これは、合成システムプログラムが動作する際遭遇する関係を解釈するときに用いる分類である．

（２）　問題定義作業領域

問題定義作業領域は、コンテキストデータベースの中に取られる．問題の出力仕様および入力仕様はその根である基本データベースに作られる．そして択一の場合分けが起こるたびに子プロセスとそれに対応するローカルデータベースが作られ、コンテキストデータベースの木の親のデータベースの下に新たな葉として追加される．コンテキストデータベースは、一つの抽象データ型として扱われる．その構造を表わすデータ型不変量は、ローカルデータベースをノードとする木構造で表わされる．

（３）　システム動作用データ構造

図１（ｃ）のリストはこの一例である．

５．　モジュールおよび操作プログラムの生成

１）　モジュールおよび操作の内容の生成には、次のような幾つかの方法がある．

①　プログラム合成システム自体の利用．

合成システムに入出力仕様を与え、システム内蔵の知識を用いて合成を行なわせる．入力構造に合わせたアクション部分の融合もこの一部として行なわれる．

②　標準抽象データ型の操作プログラムの入力構造（それにともないプログラム構造も）の修飾による問題抽象データ型の操作プログラムの生成．

標準抽象データ型の操作プログラムの入力構造に、合成システムで用いる関係の分類情報に基づき場合分けを導入し、木の走査プログラムから合成システムのグラフモデルの走査プログラムを得る（図５）．

③　ループや場合分けの融合．

④　木のアクションとしてのサブモジュール．

木構造をendorderに辿りながら、後始末の形でアクションを行なうプログラムは、プログラム合成システムに数多く存在する．木の各節における列の公理の適用による部分木の書き替え、のようなプログラムは、モジュール内で使用するサブモジュール（プロダクションルールの適用を行なう）の指定により生成される．

２）　階層構造をなすモジュール群におけるモジュール間インターフェースは、値の受渡しのデータフローを中心としている．これは、図６に示すように、属性文法の継承属性入出力、合成属性入出力の見方が参考になる．データ構造とモジュールの関連から、ボトムアップに探索が行なわれる．

３）　フロー表現から、トポロジカルソートを利用して、プログラム木の編集

を行なう．

なお、ＣＡＬＬ関係を辿ることにより全モジュールの枚挙は行なえる．実現変換については、ここでは触れない．

図５　木を辿るプログラムとその修飾

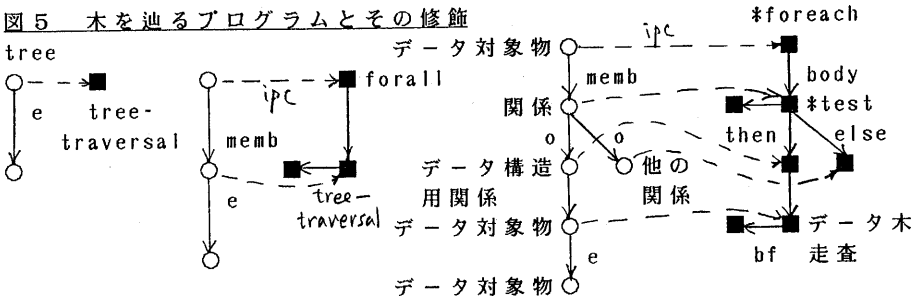
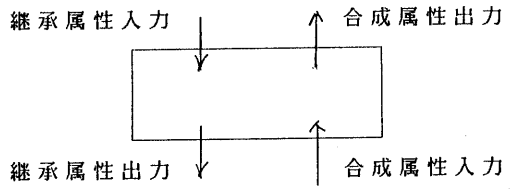
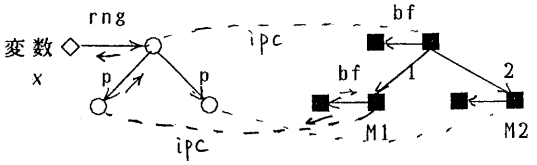


図６　モジュールとその入出力



データ構造とサブモジュールＭ１、Ｍ２との関係



関係のラベルの説明

p	データ集合の全体部分関係
o	データ集合の直和
e	列データの繰り返し
memb	集合データの要素集合の関係
rng	変数の値域
ipc	入力木とプログラム木の構成要素間の対応
then	判定文の条件が成立する場合に行なうセグメント
else	判定文の条件が不成立の場合に行なうセグメント
body	繰り返し文の本体を指す
bf	前始末の算譜構成要素を指す

６．　おわりに

合成システムプログラムを例題として、一般のシステムプログラムをプログラムに関する知識を内蔵したシステムを用いて生成する方式について述べた．同時にそれは、合成システム自体の記述にもなっている．合成システムプログラムを生成するのに、合成システム自身を直接用いることはまだ不可能に近い．しかし、本論文で述べたように、合成システムを構成する幾つかの基本的な機能を一般のシステムプログラム作成時に利用することは十分可能である．またプログラムを入出力仕様と同時にモデル化して表現しておく利点も、本文中の説明から明らかになったことと思われる．第３章に述べたコルーチンを用いる問題解決機は、データの構造に基づくプログラムの生成というプログラミングの方法論の範囲外のものである．この種の問題は、分散システムプログラムとともに、今後の研究課題である．

最後に、日頃お世話頂く棟上ソフトウエア部長と、言語処理研究室、情報研究室の方々に感謝します．

参考文献

[1] 間野暢興：”抽象データ型を用いたプログラムの合成における知識表現と問題解決”、情報処理学会ソフトウエア工学研究会資料46-9 (1986).
[2] 間野暢興：”仕様とプログラムのモデルに基づく抽象プログラムの実現変換”、電子通信学会オートマトンと言語研究会資料AL85-69 (1986).
[3] 間野暢興：”処理構造を持つ抽象プログラムの合成について”、電子通信学会オートマトンと言語研究会資料AL85-88 (1986).
[4] Phillips,J.: "Self-described Programming Environments: An application of a Theory of Design to Programming Systems" (1983).
[5] Smith,D.R, Kotik,G.B. & Westfold,S.J.: "Research on Knowledge-Based Software Environments at Kestrel Institute", IEEE Trans. on Software Engineering, Vol.SE-11, No.8 (1985).
[6] McDermott,D. & Sussman, G.J.: "The CONNIVER Reference Manual",MIT AI Memo No.259a (1974).
[7] Nakashima, H.:"Uranus Reference Manual", Bulletin of the ElectroTechnical Laboratory, Vol.50, No.8, pp. 829-912 (1986).