モデルを用いた抽象プログラムの導出と精製

正員 間 野 暢 興 (電子技術総合研究所)

Model-based Derivation and Refinement of Abstract Programs
Nobuoki Mano, Member (ElectroTechnical Laboratory)

This paper proposes a method which makes full use of the functions of information representation and procedural and declarative knowledge of S-model to derive programs using abstract data-types such as sets and relations and to refine them to programs manipulating arrays and pointers. In this method, program derivation requires 2-phases; first, a syntax-type (a program control structure and related subproblems with it) is generated by analyzing the structure of problem input-output specifications and constraints, and by using rules and axioms in S-model's knowledge-base, and then solution program model is derived by the multi-level search based on the syntax-type. Functions of S-model mentioned above are also indispensable for data refinement, as the refinement is essentially redesign of data structures in the concrete level.

キーワード：意味モデル、抽象データ型、複合データ構造、探索、データ精製

1. はじめに

抽象データ型¹⁾を用いるプログラミングは、モジュール化、高水準表現によるプログラム作成、その実現プログラムへの精製¹⁾による能率化最適化、多様な部品の再利用可能性など、様々な意義がある。本論文では、集合、関係データ(relation)などの抽象データ型を用いるプログラムの導出のための、そして、その具体データ型を用いるプログラム(最終的には、配列、レコード、ポインタなどのデータ構造を用いる実現レベルの命令型プログラム²⁾)への精製のための新しい方式を提案する。

以下に、a)情報の表現、b)プログラムの導出、c)精製の各観点から従来の方式について検討を加える。

a) 情報の表現形式： 論理による演繹あるいは規則を用いた変換には、主に一次元テキスト表現が用いられてきた。これは、仕様やプログラムの表現として簡潔、語彙・文法・意味・運用に関して共通の認識が定まっており馴染み深い、入出力が容易、などの理由による。しかし、テキスト表現は、概念の木やグラフ構造を表わすにはあまり適しておらず、意味的な操作を施すには構造として柔軟性に欠けている。

b) プログラムの導出方式： 従来提案されてきた方式としては、正当性維持変換(correctness-preserving-transformation)用規則を、仕様およびプログラムに半自動的に繰返し適用し、導出(と精製)を行なう方式が多かった³⁾。たとえば、CHI⁴⁾では、V言語で記述された仕

様記述を規則の適用により変換して行く。また、KIDS^{5),6)}は、段階的精製に近い方式で、アルゴリズム理論あるいは(メニューで選択された)規則の適用により、抽象表現を具体表現へと変換して行く。しかし、これらのプログラム変換規則による方式は、人間プログラマのとりプログラミング方式(データの構造に基づくプログラミング)とは異なっており、次に述べるような問題点がある：①段階的詳細化の過程において、具体レベルにおける記述を予め想定して導いて行くことが必要であり、②変換の途中、型変換操作など細かい操作の挿入や、公理の適用による簡単化など規則を取捨選択するのに、プログラマの適切な判断が必要となる。

c) 精製の方式： 具体/実現データ構造の選択においては、集合記述言語SETLにおけるデータ構造選択⁷⁾など、昔から幾つかの論文がある。しかし、ユーザの置かれた様々な状況に適応するためには、システムが自動的に具体データ構造を決定する方式よりも、ユーザがシステムの表示したものの中から選択する方式の方が実際的と思われる。

これに対して本論文で提案する方式は次の特徴を持つ。

a) 情報の表現形式： 本方式では、意味モデル⁸⁾により、仕様、プログラム、データ、知識を表わす。意味モデルは、対象物(object)-関係(relationship)の表現形式をとるオブジェクト指向(但し、メッセージによる手続き駆動は原則として用いない)のモデルである。意味モデルで表現された部品は次の特徴を持つ：①柔軟でカスタマイズが容易な構造を持つ、②問題入出力仕様、操作の事前事後

条件、状態などは（構造表現と一体化した）厳密な形式化された集合／論理表現で表わされる、③入力とプログラムの構造の要素どうしの対応を付けることにより、データの構造に基づくプログラミングが行なえる、④計算機科学の分野でのアルゴリズム研究の成果⁹⁾を、精製の過程において部品の再利用の形で利用し、プログラムの能率化を図る。

b) プログラムの導出方式： 本方式は、次の二つのフェーズからなる、意味モデルの特性を生かした変換／探索による導出方式である：①問題入出力仕様と拘束条件の構造の解析、および規則と公理の適用によるプログラムの構文型（制御構造とそれに関連する部分問題の集合）の導出、②構文型に基づく（コンテキストデータベース¹⁰⁾の内容を反映した）探索によるプログラム抽象構文木の導出。本方式は、導出の明確な手順を与える、探索の組合せ的性質を基に多様な範囲の問題を扱える、などの利点を持つ。

c) 精製の方式： 精製は本質的にはデータ構造の再設計であり、それゆえ対応する精製プログラムの設計には原則として探索の機能を必要とする。また、抽象プログラムの構文型と具体プログラムの構文型とが準同型の場合には、表現関数を用いた仕様／状態の変換により抽象レベルの構文型から具体レベルの構文型を導出し、その構文型に基づく探索により具体レベルのプログラムを導出する。精製において意味モデルを情報の表現に用いる利点を次に挙げる：①抽象データ型のデータ型不変量の表現は、そのデータに関する特性および具体データ型選択の判断に用いる情報を表わしている。②抽象レベルのプログラムで用いられる操作群は、具体データ型を決定するもう一つの判断材料であるが、容易に見出せる。③場所集合を中心とした実現レベルの状態記述は、データ集合を中心とした抽象レベルの状態記述から容易に導くことができる。

本論文の内容は次の通りである。上記のa), b), c)に、第2章、第3章、第4章がそれぞれ対応する。第2章では、意味モデルの概要について紹介する。第3章では、抽象レベルにおける仕様からの構文型導出方式、および、構文型に基づく探索により抽象プログラムを導出する方式について述べる。第4章では、抽象プログラムの、配列を用いる実現プログラムへの精製方式について述べる。本論文を通した例題として、集合分割の視点からクラフのスパニング木を求めるアルゴリズム¹¹⁾（GST問題と呼ぶ）とその精製を取上げる。同値類の処理には、Fischer-Gallerアルゴリズム¹⁾が採用される。

2. 意味モデルによる問題と知識の表現

2.1 対象物と関係

意味モデルは、対象物と関係からなる単一継承のオブジェクト指向モデルであるが、これらのクラスは複数のローカル知識ベースに、インスタンスは複数のローカルデータベースに属することが許される。クラス名は英数小文字で、インスタンス名は英数大文字で表わすものとする。

対象物は、原子対象物、構造体対象物、および、部分空

間に分類される。原子対象物としては、データ集合（data-set、記号○で図示。データ構造、入出力仕様、状態、拘束条件の記述において、データの構造の要素記述に用いる。

2.2 参照）、場所集合（location-set、記号△で図示。配列、レコード、ポインタなどのデータ構造と参照呼びを用いる命令型言語の意味記述に用いる）、識別子（identifier、記号◇で図示。プログラム記述に用いる変数名や欄名）、算譜構成要素（program component、記号■で図示。プログラム構造の記述に用いる対象物で、if文、while文などの算譜構造物、操作（関数と手続き）の呼び出し文、代入文、および部分問題からなる）、および、論理演算子（論理式の記述に用いる、否定、論理和、含意、および同値、の各演算子）からなる。構造体対象物は、データ型、操作、公理、および規則に分類される。部分空間（長方形で図示）は内部に意味モデルを含む対象物であり、状態はその一つの型である。

関係のクラスは、そのドメイン対象物クラスとレンジ対象物クラスにより定義される。再帰的に定義される関係も許す。関係は述語も含む。

2.2 データの構造の表現

抽象レベルでのデータの構造は、データ集合をノードとする木で表わされる（実現レベルに関しては、4.1 参照）。各データ集合クラスは一定の定義域を持つ。そのインスタンスは、クラスと同一の定義域を持つが、個々の基数（cardinality）は異なる（ここでは有限集合とする）。木の根のデータ集合の基数は1である。

データ集合（および場所集合）どうしを結ぶ関係（のインスタンス）は、それらの集合の要素間の対応を示す対応情報をその属性として持つ。これは、一方のデータ集合の一つの要素に対応する他方のデータ集合の要素数の最小値 min および最大値 max、($0 \leq \min \leq \max$)を $[\min, \max]$ の形式で、関係の向きと同方向および反対方向について、それぞれ方向を示す矢印付きで記したものである。この対応情報を用いて、問題や操作の仕様の持つ数学的性質を表わすことができる。 ∞ は、有限集合の場合、不定個を表わす。

基数cの集合a-setを図1左のように表わす。関係membは、ドメイン対象物として集合を表わすデータ集合を、レンジ対象物として集合の要素を表わすデータ集合をとる関

図1 集合と分割の表現

Fig.1 S-model representation of a set and a partition.

$$\begin{aligned} & \text{union } p = a\text{-set } \Delta \#(a) = c \quad (c > 0) \quad \text{--- ①} \\ & \frac{(\forall si, sj \in p)(si = si \vee si \cap sj = \emptyset) \wedge \sim (\{ \} \in p)}{\text{②}} \quad \text{③} \end{aligned}$$

- ① 静的実データ――immutable な表データ。
- ② 動的実データ――mutable なデータ構造。
- ③ 動的状態表明――拘束条件や状態の表明文。

図2において、関係diffで結ばれた部分空間は、木構造のコンテキストデータベースを形成する。矢印は可視の方向（木のある節点においては、その節点から根への枝上にある全てのローカルベースの内容を見ることができ、それらはその節点におけるコンテキストとして働く）を示す。構文型に基づく探索においては、制御構造に従って部分問題の探索を行なう際に、それまで探索が成功に終わった部分問題の履歴がコンテキストデータベースの枝に蓄積されており、そのコンテキスト下で次の部分問題の探索が行なわれる。

2.5 知識構造

一般のプログラミング知識を表わす構造は、システムの知識構造の一部に含まれており、データ型と操作の上位下位構造をその基本的骨組とする。知識ベースの設計者は、一般プログラミング知識構造の根となる data-type クラスや operator-type クラスのメソッドを用いて、上位（一般的）のデータ型と操作から順に定義を格納することによって、予め一般プログラミング知識構造を作成しておく。

データ型は、操作集合、データ型不変量、公理集合、繰返し文に関する構文型の推測規則集合（図4）や精製の具体データ構造選択規則集合（図7）など、を成分として持つ（下位データ型は上位データ型から継承する）。操作（手続きおよび関数）は、入出力の引数とその値となるデータ集合、事前条件、事後条件、およびプログラム本体、を成分として持つ。上記の規則集合を構成する規則はプロダクションシステムを、（構文型を構成する）部分問題は還元型と前進型の探索機を、メソッドとして持つ。

2.3 で述べた集合の分割データの源となるデータ型は分割データ型であり、以下の仕様を持つ操作init、equate、およびequalsにより定義される。操作initは分割の初期状態を作るconstructor、操作equateは二つの頂点が含まれる頂点部分集合どうしをその和の頂点部分集合で置き換えるconstructor、そして、関数equalsは二つの与えられた頂点が同じ頂点部分集合に属するか否かを判定する述語、である。なお、操作の入出力引数の；の前は入力変数、後は出力変数を表わす。precond およびpostcondは、それぞれ事前条件および事後条件を表わす。

```
init( ; p: partition)
  precond:  { x | x ∈ s }
  postcond: p = { {x} | x ∈ s }.
equate(x1:a, x2:a)
  precond: p ∧ x1 ∈ s1 ∈ p ∧ x2 ∈ s2 ∈ p ∧ s1 ≠ s2
  postcond: p' = (p - s1 - s2) ∪ {s1 ∪ s2}.
equals(x1:a, x2:a; res:bool)
  precond:  p ∧ x1 ∈ s1 ∈ p ∧ x2 ∈ s2 ∈ p
  postcond: p' = p ∧ (((s1 = s2) ⇒ (res = true)) ∨
    ((s1 ≠ s2) ⇒ (res = false))).
```

木データ型の持つ操作tree-constructは、二つの disjoint な木とその各々の上に端点を持つエッジから、それらを含む一つの木を作り出す。

3. 変換プログラムの導出方式

3.1 問題の範囲

集合、分割、写像、同値関係データ (equivalence relations)、関係データなどの抽象データ型の集まりの直積結合型複合データ構造を操作する問題を対象とする（階層型結合は除く）。直積結合とは、複数の表の欄どうしが意味の上で結合しているものである。

ここでは、入力構造を出力構造へ変換する問題を考える。導出の対象とするプログラムは、一個の繰返し文および条件文を含むことを許すが、これらの繰返しやネストは許さないものとする（それらの場合は、人間が介入して、繰返し文を別の操作として定義しておき、それを呼ぶようにする）。このタイプの問題は、入力構造成分の取り出しと出力構造の組み立ての両方からなる。入力構造成分の取り出しの部分は、出力構造にたどり着くまでの通りのコードである。出力構造の組み立ての部分は、集合あるいは関係データに登録するコードである。出力が複合データ構造の場合、これらに同時登録操作を行なっても、データ構造間の不変関係は維持される必要がある。

3.2 問題の仕様

ユーザの与える問題の仕様を次に記す。

- ① 問題の入出力仕様。問題は、入出力の引数とその値となるデータ集合、入力仕様、および、出力仕様により定義される。
- ② 関係データの記述（主鍵、および欄間の関数従属性の指定）¹²⁾、関係データどうしの欄のデータ集合を通しての（等号あるいは部分集合関係による）結合。
- ③ 拘束条件として、複合データ構造間の不変関係を含むループ不変量の表現（データ集合間の対応情報による）。
- ④ 一般プログラミング知識構造中の標準抽象データ型を、問題抽象データ型の上位データ型として指定。

例題：集合の分割の観点に基づきグラフのスパニング木を求める問題（GST問題）

このグラフのスパニング木を求めるアルゴリズム¹³⁾は、集合と分割の二つの視点から同時にデータを解釈することができる。集合の視点では、グラフGを構成するエッジ集合G.Eと頂点集合G.Vとを入力とし、そのエッジ部分集合（＝スパニング木）を出力（エッジ集合t）とする、入力エッジ集合のフィルタリング問題である。分割の視点では、最初に初期設定部分（操作init）において、頂点集合からその単元の集合の集まりである頂点（集合の）分割を作成する。次に本体部分では、出力のエッジ集合と頂点集合を同時に各々エッジ分割（スパニング部分木の集合）qおよび頂点分割（頂点部分集合の集合）pのデータ型として解釈する（頂点部分集合 vs3 とスパニング木 t3 とは関係 tvs により結ばれ、後者が空集合でなければ、1対1に対応する）。そして、入力エッジeごとに、その頂点v1の属する頂点部分集合vs1と頂点v2の属する頂点部分集合vs2が異なる場合にはそれらを合併し、同時にそれらの頂点部分集

図 3 GST問題の仕様と拘束条件 (ループ不変量)
Fig.3 The specification and constraint
(loop-invariant) for GST problem.

合のスパニング木 t_1 と t_2 およびエッジ e との合併を繰り返すことにより、最終的に単一に統一されたスパニング木 t_3 を出力として得る。

関係 tvS の定義は次のようになる。ここで、 head と tail は、エッジとその両端点の各々を結ぶ関係である。

$$\begin{aligned} (\text{tvS } t_3 \text{ vs}_3) \equiv & ((t_3 = t_1 \cup t_2 \cup \{e\}) \wedge \\ & (\text{vs}_3 = \text{vs}_1 \cup \text{vs}_2) \wedge (\text{vs}_1 \neq \text{vs}_2) \wedge (\text{tvS } t_1 \text{ vs}_1) \wedge \\ & (\text{tvS } t_2 \text{ vs}_2) \wedge (\text{head } e \text{ vs}_1) \wedge (\text{tail } e \text{ vs}_2) \wedge \\ & (\text{vs}_1 \in \text{vs}_1) \wedge (\text{vs}_2 \in \text{vs}_2)) \\ & \vee ((\text{vs}_3 = \{v\}) \wedge (v \in \text{vs}_3) \wedge (t_3 = \emptyset)) \end{aligned}$$

GST問題の仕様を以下に記す。

(a) 初期化部分問題

①入力仕様: $\{v \mid v \in G.V\} \wedge p = \emptyset$.

出力仕様: $p = \{\{v\} \mid v \in G.V\}$.

②頂点分割は、頂点を主鍵とし頂点部分集合はそれに関数従属する動的な実関係データとして表現。

(b) 主要部分問題

①入力仕様: $p = \{\{v\} \mid v \in G.V\} \wedge q = \emptyset$.

出力仕様: $p = \{G.V\} \wedge q = \{t\} \wedge (\text{tvS } t \text{ G.V}) \wedge (t \subset G.E)$.

②グラフのエッジ-頂点表は、(入力) エッジを主鍵とし頂点はそれに関数従属する静的実関係データである。

③ループ不変量の一部

$$\begin{aligned} \text{union}(p) = G.V \wedge & (\forall \text{vs}_i, \text{vs}_j \in p) (\text{vs}_i = \text{vs}_j \vee \\ & (\text{vs}_i \cap \text{vs}_j = \emptyset) \wedge \sim (\{ \} \in p) \wedge (\forall t_i, t_j \in q) \\ & (t_i = t_j \vee t_i \cap t_j = \emptyset) \wedge \text{union}(q) = t \subset G.E \wedge \\ & \forall \text{vs}_i \in p \exists! t_j \in q (\text{tvS } t_j \text{ vs}_i) \wedge \\ & \forall t_i \in q \exists! \text{vs}_j \in p (\text{tvS } t_i \text{ vs}_j). \end{aligned}$$

(ただし、 $\exists!$ = there exists a unique)

この拘束条件 (データ型 o-forest とデータ型 vertex-partition のデータ型不変量およびそれらを構成する要素間に成立する不変関係) の意味モデル表現は図3のようになる。関係 ev はエッジとその頂点間の関係を表わす。

④一般プログラミング知識構造中の標準抽象データ型 (下左) の下位データ型として、問題抽象データ型 (下右) を登録する。

集合 $\leftarrow \text{-----}$ 入力エッジ集合 $G.E$ 、

集合の分割 $\leftarrow \text{-----}$ 頂点集合の分割、
木 $\leftarrow \text{-----}$ 出力エッジ木。

3.3 プログラムの構文型の導出方式

(1) 前処理 (問題の入出力構造の解析)

問題入出力仕様と拘束条件の構造の解析により、下記の3種類の基本データ構造と、それらの間の関係を見出す。

・入力データ構造 (inds)

・出力データ構造 (outds) $\leftarrow$ 出力木の葉は同一ラベルでも根の対象物が異なる。

・入出力両用データ構造 (iods) $\leftarrow$ 入力木と出力木の、根どうし、および葉どうし、が同一のインスタンスデータ集合。根と葉を結ぶ関係 (とその対応情報) は、入力木と出力木とは異なっている。

基本データ構造間の関係の種類として次を見出す。

・入力データ構造と入出力両用データ構造の間の関係。

等号関係あるいは部分集合関係により結ばれる。

・出力データ構造と入出力両用データ構造の間の関係。

非葉の節どうしの関係は、拘束条件を表わす。

(2) プログラムの部分問題入出力仕様の導出と構文型の作成

(2-1) [繰返し文に関する構文型推測規則 (図4) の適用]

繰返し文前後状態から繰返し文本体の事前事後状態の表明を得るために、経験から得られた繰返し文に関する構文型推測規則群 (図4参照) を利用する。これらの規則は、データを集合と基数の観点から捉えたものである。入力データ indata を $\leftarrow$ 既処理部分、当該要素、未処理部分 $\rightarrow$ と、集合中の一要素に注目した一般的な分解形で表わし、出力データ outdata もそれに合せた分解形で表わす。図の (a) と (b) は入力構造 (inds) の出力構造 (outds) への変換に用いられる規則、(c) は入出力両用構造 (iods) の書き換えに用いられる規則である。規則の $\Rightarrow$ の左辺は、繰返し文の前 (厳密に言えば、その出力変数の初期化文の前の状態) と後の状態を表わしている。右辺は、初期化状態 (ない場合もある)、繰返し文本体中の操作の事前状態、およびその事後状態、を表わしている。(c) の規則のように、右辺が条件ごとに記述が分けられることもある。得られた二つの状態の内容を比較し共通する部分は取り去ると、(繰返し文本体中の操作に相当する) 部分問題の入出力仕様が得られる。

問題入出力仕様 (中の基本データ構造) と整合の取れる左辺を持つ規則の右辺を展開し、繰返し文本体中の操作の事前事後状態におけるデータ表現の記述を作り出す。整合のとれる規則が複数個存在する場合には、展開されたものを共通の状態ごとにまとめる。その際、条件付きのものがあれば、条件ごとに別々にまとめる。

(2-2) [関係の継承]

問題がさらに次のいずれかの性質を持つときは、その関係を (2-1) の結果に追加する。

・この分解された表現における部分集合の一項関係は、

(a) 集合の変換に対する規則

```

[st1: indata={a | a ∈ xs} ≡ xs1 ∪ {a} ∪ xs2 ]
[st2: outdata={f(a) | a ∈ ys} ≡ ys1 ∪ {f(a)} ∪ ys2 ]
⇒ [st02: outdata=∅]
[st11: indata={a} ∪ xs2 ∧ outdata=ys1 ∧ a ∈ xs]
[st12: indata=xs2 ∧ outdata'={f(a)} ∪ ys1 =
      {f(a)} ∪ outdata ].

```

(b) 集合から分割への変換に関する規則

($\forall ti, tj \in q$) ($ti=tj \vee (ti \cap tj)=\emptyset$) の下で、

```

[st1: indata={e | e ∈ initd} ≡ es1 ∪ {e} ∪ es2 ]
[st2: outdata=q ≡ {t} ∧ t ⊂ initd ] ⇒
[st02: union(q)=∅]
[st11: indata={e} ∪ es2 ∧
      outdata=q={t1} ∪ {t2} ∪ qrest ∧ e ∈ initd]
[st12: indata=es2 ∧
      outdata=q'={{e} ∪ t1 ∪ t2} ∪ qrest =
      q - {t1} - {t2} ∪ {{e} ∪ t1 ∪ t2} ].

```

(c) 分割の変換に関する規則

($\forall si, sj \in p$) ($si=sj \vee (si \cap sj)=\emptyset$) $\wedge \sim(\{ \} \in p)$ の下で、

```

[st1: iodata ={{x} | x ∈ initd}=p ]
[st2: iodata ={initd}=p ] ⇒
[ (s1 ≠ s2) ⇒
[st11: iodata ={s1} ∪ {s2} ∪ prest]
[st12: iodata'={s1 ∪ s2} ∪ prest
      =iodata - {s1} - {s2} ∪ {s1 ∪ s2} ]
[ (s1 = s2) ⇒
[st11: iodata ={s1} ∪ {s2} ∪ prest]
[st12: iodata'={s1} ∪ {s2} ∪ prest=iodata ] ].

```

ただし、st1, st2: 繰返し文の前の状態、後の状態、
st02, st11, st12: 初期化状態、繰返し文本体中の操作の
事前状態、事後状態。

図4 繰返し文に関する構文型推測規則
(初期化部分と場合分けを含む)

Fig.4 Rules for inferring syntax-types
on iteration-statements.

入出力仕様において成立する集合の一項関係を継承する。

・この分解された表現における部分集合間関係は、入出力仕様において成立する集合間の関係（入力と出力データ構造間の関係）を継承する。

・この分解された表現の構造内部で成立する部分構造間関係は、問題の拘束条件として宣言されているデータ構造間関係（複合データ構造をデータ型としてみた場合のデータ型不変性）を継承する。

(2-3) [繰返し文の融合] 入力データ構造が共通で出力データ構造が複数個の場合は、繰返し文を融合する。また、入力データ構造と入出力共用データ構造があり、入力データ構造の走査により入出力共用データ構造がexhaus-

tiveに枚挙できる場合には、入力データ構造の繰返し文を用いて統一的にプログラムを表現する。

(2-4) [不変関係維持のためのプログラム変換] 出力データ構造が複数の場合、それらへの登録操作後も不変関係が維持されるように、必要ならば繰返し文本体中の操作の等価性を保ったままの移動（判定文の各場合分けへの分配など）を試みる。文の入替えや逐次文の判定文内への分配は、相互に干渉のない文どうし（事前条件から判明する）ならば許される。

例題：GST問題の抽象レベルのプログラム導出

(1) 入出力仕様の解析

- ・入力データ構造(inds)ー入力エッジ集合とその頂点表、
- ・入出力両用データ構造(iods)ー頂点集合の分割、
- ・出力データ構造(outds)ー出力エッジ木の集合。
- ・入力データ構造と入出力両用データ構造の関係として、頂点集合から入力エッジの頂点head-vertex およびtail-vertex のそれぞれへ部分集合を表わす関係subset。
- ・出力データ構造と入出力両用データ構造の関係として、関係tvs。

(2) 構文型の導出

説明の便宜上、問題の解として得られる、分割データ型を用いたGSTアルゴリズムのテキスト表現を図5に示す。図中の[ASXi: aaaa] (aaaaは論理式) は、表明を表わしたものである。ASX2 (X=B, E, H, I, --) は、その時点の（直前の操作の呼び出し文実行後の）状態を表わす。ASX1はその操作の満たされるべき事前条件を展開したもので、部分問題の出力仕様をゴールとすれば、サブゴールに相当する。ASX は事実を表わす。判定文の場合は、then-partにASXYを、else-part にASXNを用いる。下線は、プログラムの構文型の骨組を示すために付けた。

以下に、構文型の導出過程を説明する。

①入力エッジ集合には、図4の規則(a)と規則(b)が適用され、その結果にさらに、二つの観点から見た同一データ構造の（不変）関係 $t = \text{union}(q)$ を重畳すると、図5のASC2とASI2の表明文が得られる。

②頂点集合の分割には、規則(c)が適用され、エッジの両端点異なる頂点部分集合に属する場合(ASGY)、ASH2の表明文が得られる。エッジの両端点同一頂点部分集合に属する場合(ASGN)には、変化は起こらない。

③プログラムは、入力構造(inds)の入力エッジを走査する繰返し構造を基とする。そして、入出力共用構造(iods)である頂点集合は、入力エッジ集合との関係evの対応情報が[1, ∞]であることから、それに従ってexhaustiveに走査されることになる。①で得られた表明文ASI2と、②場合分けASGY（その中にASH2の表明文を含む）およびASGNとが、並置される。

④ここで、問題の拘束条件（図3）として与えられたデータ構造間不変関係（新たに組み立てられた出力エッジ木と頂点部分集合の関係[ASJ2: (tvs t3 vs3)]）に起動されて、ASI2の表明文は場合分けASGYとASGNの中へ分配

```

proc gst( )
e:input-edge of G.E, t:o(output)-edge-set;
v1,v2: vertex of G.V;
[ASA: {v | v ∈ G.V} ∧ {e | e ∈ G.E} ∧
(∀ e ∈ G.E ∃ y1,y2 ∈ G.V)((head e y1) ∧ (tail e y2))]
begin
  [ASB1: {v | v ∈ G.V}]
  init();
  [ASB2: p={ {v} | v ∈ G.V} ∧ (∀ v ∈ G.V ∃ vs)(v ∈ vs ∈ p)]
  t:=∅; [ASC2: t=∅] ー 部分問題-01
  forall e ∈ G.E do
    begin [ASD: e ∈ G.E]
      [ASINV: union(p)=G.V ∧
        (∀ vsi,v sj ∈ p)(vsi=vsj V (vsi ∩ vsj)=∅)) ∧
        ({ } ~ ∈ p) ∧ (∀ ti,tj ∈ q)(ti=tj V (ti ∩ tj)=∅)) ∧
        t=union(q) ⊂ G.E ]
      [ASE1: e ∈ G.E ∧ (∀ e ∈ G.E ∃ y1,y2 ∈ G.V)
        ((head e y1) ∧ (tail e y2)) ]
      v1:=head(e); v2:=tail(e);
      [ASE2: (head e v1) ∧ (tail e v2) ∧ v1 ∈ G.V ∧ v2 ∈ G.V
        ∧ v1 ∈ vs1 ∈ p ∧ v2 ∈ vs2 ∈ p]
      if [ASF1: v1 ∈ vs1 ∈ p ∧ v2 ∈ vs2 ∈ p ]
        ~equals(v1,v2) [ASF2: ]
      then [ASGY: vs1 ≠ vs2] ー 部分問題-11
        begin [ASJ: (tvs t1 vs1) ∧ (tvs t2 vs2) ∧
          t1 ∈ q ∧ t2 ∈ q ]
          [ASH1: v1 ∈ vs1 ∈ p ∧ v2 ∈ vs2 ∈ p ∧ vs1 ≠ vs2 ]
          equate(v1,v2);
          [ASH2: (vs3=vs1 ∪ vs2) ∧ (p'=p-{vs1}-{vs2} ∪ {vs3})]
          [ASl1: (head e v1) ∧ (tail e v2) ∧ v1 ∈ vs1 ∈ p ∧
            v2 ∈ vs2 ∈ p ∧ (vs1 ∩ vs2)=∅) ∧ (tvs t1 vs1) ∧
            (tvs t2 vs2) ∧ t1 ∈ q ∧ t2 ∈ q ∧ t=union(q)]
          t:= t ∪ {e};
          [ASl2: (t3={e} ∪ t1 ∪ t2) ∧ (q'=q-{t1}-{t2} ∪ {t3}) ∧
            t'=union(q') ∧ t'=t ∪ {e}]
          [ASJ2: (tvs t3 vs3) ] ー 部分問題-21
        end
      else [ASGN: vs1=vs2] noop
      end
    end
  end
end

```

図5 分割データ型を用いたGST問題のプログラム
(下線部は構文型を表わす)

Fig.5 Program for GST problem using 'partition'.
(underlined parts represent its program syntax-type.)

されて入替えられる。これは、判定文の事前事後条件とエッジの出力エッジ集合への追加が、相互に不干渉なため、入替えてもよいからである。ただし、同一の出力エッジスパニング部分木にエッジ e を加えたものは、ループが発生しその事後状態が拘束条件チェックにおけるデータ型不変

量を満たすことができないので、算譜構成要素noop(no operation)を対応させる。

3.4 構文型に基づく多レベル探索

逐次文を導出するための基本的な探索として、還元型探索と前進型探索とがある。還元型探索は、問題出力仕様を、ゴールのサブゴールへの分割と、候補操作の事後条件事前条件の適用に基づくサブゴールの書き換えにより、問題入力仕様へと試行的に還元を行なう探索で、and/orグラフの形式をとる。これに対して、前進型探索は、問題入力仕様から出力仕様へと候補操作の事前条件事後条件の適用により状態空間探索を行なう。

繰返し文や判定文を含むプログラムを導出するためには、構造化された探索として、構文型に基づく多レベル探索が用いられる。この探索では、管理プロセスと（内部に上記基本探索機を持つ）作業プロセスの集合とが協調して、因果関係の順序に従った各部分問題の解の導出と、データ構造間の不変関係充足の検証を行なう。

管理プロセスは、与えられた構文型に基づく因果関係の方向に従って、上記部分問題の集りから一つずつ取り出し、その探索を担当する作業プロセスを生成して、その時点での状態に関するコンテキストデータベース（図2参照）の葉を与えて駆動する。

このコンテキストデータベースには、①大局的宣言、②それまでに入った（そしてそれから未だ出ていない）判定文の条件のスコープ、③それまでに入った（そしてそれから未だ出ていない）繰返し文の（ループ不変量と継続条件）のスコープ、および、④それまでのレベルを通して経過してきた部分問題の出力仕様の列による状態変化（とくに変数のスコープ）、が含まれる。

作業プロセス中の基本探索においては、この情報も考慮に入れて候補操作の抽出がなされる。個々の部分問題の出力仕様には、探索によりそれを満たす、①入力仕様、②場合分け条件、および、③候補操作の事後条件、が見出される。③の事前条件はそのコンテキストで満たされなければならない。プログラムの構文木から取り出された部分問題の探索が成功終了したとき、得られた解のプログラムコードはその位置に埋め込まれる。解かれた部分問題の出力仕様は、コンテキストデータベースに追加され、次に解かれるべき部分問題の探索に用いられる。判定文および繰返し文については、プログラム論理の検証がなされるとともに、スコープに関するコンテキストデータベースの処理がなされる。

なお、ある部分問題における探索が失敗に終わったときには、その前に行なった部分問題へ後戻りしてやり直す。

例題：GST問題の抽象レベルのプログラム導出

図5に示す部分問題は、01,11,21の順に解かれる。

■部分問題-01〔出力仕様はASC2〕：ASC2を作り出す代入文は容易に求まる。

■部分問題-11〔出力仕様は、ASGYとASGN〕：出力仕様は、分割データ型の述語equalsの事後条件と整合する。そ

の事前条件の展開から得られるASE2における v_1 、 v_2 の指すデータ集合は、関係subsetを順方向に辿り、head-vertex およびtail-vertex に達し、（それぞれ関数headおよびtailにより）入力エッジ集合に関する繰返し文による状態の要素[ASD: $e \in G.E$]にまで還元される。

■部分問題-21（[ASGY: $vs_1 \neq vs_2$]）の下で、出力仕様は、 $ASH2 \wedge ASI2 \wedge ASJ2$ ）：ASH2は、分割データ型の操作equate(v_1, v_2)の事後条件と整合し、その事前条件を展開したASH1は、ASE2とASGYにより満たされる。ASI2は、操作tree-construct（2.5 参照）の事後条件と整合する。その事前条件を展開したASI1は、ASE2とASJにより満たされる。ASJは、 $vs_1 \neq vs_2$ ならば、関係tvsが1対1対応であることから、異なる出力エッジスパニング部分木 t_1 と t_2 が存在することから導かれる。ASJ2で関係tvs(t_3 vs t_3)が成立することも、関係tvsの定義（3.2）を用いた還元によって、同様に示される。

4. 情報の意味モデル表現を用いた精製方式

本方式の対象問題は、抽象レベルおよび具体レベルのいずれにおいても直積結合型で表現可能な複合データ構造であり、これらのプログラムの構造は準同型であるとする。

4.1 実現レベルにおけるデータの構造と仕様の

意味モデル表現

場所集合は計算機の記憶をモデル化したもので、状態によらない不変的存在の対象物である。場所集合のラベルは、内部構造が一定の場所集合クラスを表わし、そのインスタンスも同一の内部構造を持つ。

実現レベルにおける仕様および状態の記述には、次の関係が用いられる。

- ・loc：変数からその（基数1の）場所集合への関係。
- ・store(A,B)：場所集合Aから場所集合Bへと指すポインタを表わす関係。
- ・data：場所集合からそれが格納しているデータ集合への関係。

図6（a）に、集合に対応する配列と、同値関係データに対応する配列上のポインタ木の集合の意味モデル表現を示す。配列全体の表現は、次の3つの成分から構成される。

- ①配列を表わす場所集合とそれを構成するコラムの場所集合を関係lmemb（対応情報中の c は配列のサイズを表わす）で結ぶ集合記憶表現、
- ②場所集合とインデックスデータ集合とを結ぶ関係ind（1対1対応）、—②と③は、図の点線左側の部分。
- ③ある単一データ集合とその要素のデータ集合を関係membで結ぶ集合データ表現（抽象レベルのデータの構造の意味モデル）と、配列を表わす場所集合からデータ集合への関係data—図の点線右側の部分（ここでは、配列中のデータはポインタと同じ働きをするので、場所集合とそれに関する関係lmembを用いて表わすことにした）。インデックス変数 i で指される配列内部のデータは、これらの集合から関係 $\in$ で繋がれた要素対象物間の関係の集り

（a）配列とその上の木集合の表現

（b）セルフループ付きの木の再帰的定義

図6 セルフループ付き木の集合による同値類の実現
(Fischer-Gallerアルゴリズム)

Fig.6 Representation of equivalence classes
by pointer-trees with self-loops.

によって表わされる。

配列データ型の持つ操作としては、読み出しと書き込みがある。配列名とインデックスを入力引数とすることで、配列の参照呼びが行なわれる。これらの操作の事前事後条件は、事前条件から図6（a）に示した部分を、さらに事後の変化分を表わす部分空間から事前条件を、それぞれ関係diffで結ぶ差分表現をとる。

また、同図（b）は、（a）の木の根からその葉の場所集合への再帰的關係lt_nを、根にセルフループを持つ木として関係lt_{n1}を用いて再帰的に定義したものである。関係oは択一を表わす関係である。変数 x の指す場所と変数 y の指すそれとが異なる木（異なる同値類）に属する場合、 y の属する木の根の關係store（セルフループ）を x の属する木の根を指すように付替えて木を融合することにより、二つの同値類の融合を表わす。

4.2 一段の精製の手順

具体データ構造選択規則：標準抽象データ型を精製する具体データ構造を定める際には、次に述べる情報が判断の手懸かりとなる。

- ①抽象データ構造に働く操作の集まり（これには、問題入力から問題出力へのアクセスの方向性に関する情報も陰に含まれている）、
- ②データが静的か、動的に変化するか、
- ③データのサイズは一定か不定か、上限があるかないか、
- ④分類されたデータか、否か。

②、③、④は、データ型不変量を見れば判明する。それゆえ、精製用の具体データ構造選択規則は、操作群+データ型不変量⇒具体データ型のデータ型不変量の形式で表わされる。

精製の手順： 精製過程の基本構成単位である一段の精製は、次の手順で行なわれる。

(1) 標準抽象データ型の取り出し： 問題入出力仕様をもとに一般化した問題データ型不変量表現を作り、それと部分整合するデータ型不変量を持つ標準抽象データ型を一般知識構造から取り出す。

(2) 具体データ構造選択規則の選定： 関連標準抽象データ型に格納されている具体データ構造選択規則の中から、問題に適合する規則を指定する。選ばれた具体データ構造に問題記述に必要な属性記述の欄を付加して、問題向き具体データ構造を作り出す。

(3) 表現関数の決定¹⁾： 抽象レベルにおけるデータ集合と関係を、その具体レベルにおけるデータ集合（あるいは場所集合）と関係により表わす。これを基に、抽象データ型のデータ型不変量と具体データ型のそれから、問題具体状態から問題抽象状態への写像を定める表現関数retrが得られる。なお、抽象レベルのデータ表現と具体レベルのデータ表現は、異なった構造を採る場合もある。図1における等価関係を表現するのに、セルフループ付き木を用いる図6がその例である。

(4) 仕様および状態の変換： 上記の表現関数retrを逆向きに適用する（そして可能ならばさらに公理を適用して単純化する）ことにより、抽象レベルの各状態の内容から具体レベルの対応する状態の内容を作り、それらを抽象レベルの対応する状態に付加する。

(5) 抽象データ型の操作プログラムの導出： 抽象データ型の操作が事前事後条件だけ与えられてプログラム本体がない場合、以下のいずれかにより、その具体レベルにおけるプログラム本体を導出する。

①探索による導出： 抽象データ型の操作の事前事後条件を具体レベルに変換したものを入出力仕様として、それらを具体データ型の操作などを用いて繋ぐプログラムを探索により見出す。

②操作のカスタマイジング： 精製の標準具体データ型の選択の際、部品として用いるその操作プログラムの指定があるときは、問題具体操作入出力仕様と標準具体操作入出力仕様を比較し、その差（追加出力仕様）を見いだす。そして標準具体操作のプログラムに属性処理部分のコードを埋め込むことにより問題具体操作のプログラムを得て、探索に用いる操作として使用する。

(6) 抽象レベルのプログラムの具体レベルにおける精製： 抽象レベルにおいて、用いる抽象データ型の操作のプログラムおよび問題プログラムが存在する場合、これらの具体レベルのプログラムへの精製は、以下の手順で行なわれる。

(6-1) 具体レベルのプログラムの構文型の作成： 抽象

レベルにおける状態付きプログラム木と、その精製である具体レベルにおける状態付きプログラム木とが構造上準同型であるときは、(4)において具体レベルの状態を付加した抽象レベルの状態付きプログラム木から、具体レベルにおける構文型を作成する。

(6-2) 構文型に基づく多レベル探索による具体プログラムの導出： 3.4 で述べた探索方式による。

(6-3) 部分問題に一つでも完全な失敗のものがある場合には、(2)における具体データ構造の選択のところまで後戻りして再試行を行なう。

(7) 一段の精製は成功終了する。 [終り]

4.3 実現レベルへの精製過程

集合／関係データなどの抽象データ型を用いるプログラムを、配列などの具体／実現データ型を用いるプログラムへ精製するには、次に記すように、一般に数段階の精製を経る。

①集合・分割データ型から関係データレベルへのデータ型の精製： 抽象データ型を精製して、関係データ型のレベルで複合データ構造のデータ表現を描える。これは、実体－関連モデル¹³⁾を用いる場合と同様に、同一表現により問題全体の構造を把握するためである。また、関係データは、集合、写像、同値関係データなどを広く含んだ一般性のある表現であり、多くの現実世界のデータは関係データの集合により表現できること、偏った方向性のない中立的表現であること、などもその理由である。

②中間的なレベルのデータ型への精製： 中間レベルで、アルゴリズム研究の成果を利用して最適化を図るために、異なった構造のデータ表現への精製を経由する場合がある（下の例題はその例である）。

③実現レベルへのデータ型の精製： 最終的には、関係データの全てを、アクセスの方向性を考慮に入れて、配列、レコード、ポインタのデータ構造で表わす。データ集合を中心とした仕様／状態表現は、場所集合を中心とした仕様／状態表現に変換される。精製の最終結果として、実現プログラムの（状態表明文付き）抽象構文木が得られる。これを基に命令型プログラムが生成される。

例題（GST問題）

実現レベルまでの精製の過程は次に記す段階を経る：

- ① 集合⇒関係データ、頂点分割⇒写像⇒同値関係データ、
- ② 同値関係データ⇒（セルフループ付き）木集合、
- ③ 関係データと木集合⇒配列。

図7に、これらの精製に用いられる具体データ構造選択規則R1、R2、および、R3を示す。R2は入力エッジ両頂点表(inds)に、R3は出力エッジ集合(outds)に適用される。

頂点分割(iods)は、R1の適用により根にmappingのセルフループを持つ木の集合として表わされ、さらにR2の適用により配列上のセルフループ付き木の集合として表わされる（図6参照）。この木は、入力変数の指すノードから根の方向にながめると、その方向の対応情報は[1,1]であるから、線形（リンクト）リストとみなせる。そこで、木の

■同値関係データ型が内蔵している精製用規則 R 1 :

「操作 `initm`、`equatem`、および `testm` が施されるときには、同値関係データの最適化具体データ構造として、同一集合に属するデータ間の写像関係(mapping) を枝、key をその根とし、根に mapping のセルフープを持つ木の集合 (fforest) を用いる (図 2 (c))。」

■関係データ型が内蔵している精製用規則 R 2 :

「関係データにおいて、それに働く操作が主鍵 (基数 m) の欄のデータを入力とし、それに関数従属な欄のデータを出力とする操作だけである場合、方向付けした最適化具体データ構造として主鍵をインデックスに割りあてたサイズ m の配列を用いる。」

■集合データ型が内蔵している精製用規則 R 3 :

「空集合を作る操作および単一要素を加える操作からなる操作群を施す集合 (基数の最大値 n) の場合、集合の最適化具体データ構造としてサイズ n の配列を用いる。」

図 7 具体データ構造選択規則の例

Fig.7 Some example rules for selecting concrete data-structures.

任意のノード名を入力とし、その木の根ノード名を出力とする関数 `root` が内部関数として導入される。分割データ型の操作 `equate` および `equals` に対する配列を用いるデータ型の操作 `equate-at` および `equals-at` は、関数 `root` の呼び出し文を含むプログラムとして定義される。

同じく操作 `init` に対する配列を用いるデータ型の操作 `init-at` では、配列の各コラムにそのインデックス値を設定する。これは、同値類の処理に Fischer-Galler アルゴリズムを用いることを表わしている。

5. おわりに

本論文では、集合や関係データなどの抽象データ型を用いるプログラムの導出と、その配列やポインタを用いるプログラムへの精製に、意味モデルの持つ表現機能と探索機能、およびプログラミングに関する知識、を活用する方式を提案した。本方式では、中間形態として構文型を利用し、探索と変換により導出を行なう。それは次の理由による：①精製は本質的にはデータ構造の再設計であり、そのプログラム導出には探索の機能を必要とする、②抽象レベルのプログラム構造と具体レベルのプログラム構造とが準同型の場合には、データの精製に基づく状態の変換 (抽象レベルのデータ集合を中心にした仕様/状態表現から、実現レベルの場所集合を中心にしたそれへの変換) により、抽象レベルの構文型から具体レベルの構文型を導くことができる、③組合せ的な探索を用いることにより、より広範な問題のプログラムを導出できる可能性がある。

現在のところ、本方式にはまだ計算の複雑さを評価する機能は導入されておらず、最適化の評価に基く精製の制御は行なっていない。その機能の導入は今後の課題である。

(平成 6 年 7 月 25 日受付、平成 7 年 3 月 9 日再受付)

文 献

- (1) C.B.Jones: "Software Development: A Rigorous Approach", Prentice-Hall (1980).
- (2) 木村 泉・米沢明憲: 「算法表現論」、岩波情報科学講座 (1984).
- (3) H.Partsch and R.Steinbruggen: "Program Transformation Systems", Comput.Surv.,15,3, 199~236 (1983) (玉木久夫訳: 「プログラム変換システム」、コンピュータサイエンス、b i t 臨時増刊 (1984)).
- (4) A.Goldberg and G.B.Kotik: "Knowledge-Based Programming: an Overview of Data and Control Structure Refinement", H.L.Hausen (ed.): "Software Validation", North-Holland, 287~310 (1984).
- (5) D.R.Smith and M.R.Lowry: "Algorithm Theories and Design Tactics", Sci.Comput.Prog.,14, 305~321 (1990).
- (6) D.R.Smith: "KIDS: A Semiautomatic Program Development System", IEEE Trans.Softw.Eng.,16,9, 1024~1043 (1990).
- (7) E.Schonberg, J.Schwartz and M.Sharir: "An Automatic Technique for the Selection of Data Representation in SETL Programs", ACM Trans.Prog.Lang. Syst.,3,2,126~143 (1981).
- (8) N.Mano: "Modeling of Data-processing Software for Generating and Reusing their Programs", Proc. 10th Int.Conf.Softw.Eng.,231~240 (1988).
- (9) A.V.Aho, J.E.Hopcroft, and J.D.Ullman: "The Design and Analysis of Computer Algorithms", Addison-Wesley (1974).
- (10) D.McDermott and G.J.Sussman: "The CONNIVER Reference Manual", MIT AI-Memo 259a (1974).
- (11) T.E.Cheatham, J.A.Townley, and G.H.Holloway: "A System for Program Refinement", 4th Int.Conf. Softw.Eng., 53~62 (1979).
- (12) E.F.Codd: "Further Normalization of the Data Base Relational Model", R.Rustin(ed.): "Data Base Systems", Prentice-Hall,33~64 (1972).
- (13) P.P.Chen: "The Entity-Relationship Model", ACM Trans. Database Syst.,1,1 (1976).

間野 暢興 (正員) 昭和14年4月19日生れ。38年東大工学部応用物理学科卒業。40年同修士課程修了。同年通産省電子技術総合研究所入所。以来、コンピュータグラフィックス、ソフトウェア工学、人工知能、並列処理の研究に従事。現在情報アーキテクチャ部言語システム研究室主任研究官。工博。情報処理学会、I E E E など6つの学会の会員。