ソフトウェア工学 46－9
（1986. 2. 6）

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

電子技術総合研究所　　間野　暢興

1．はじめに

集合や列などの抽象データ型を用いるプログラム（抽象プログラム）をその入出力仕様から合成する問題としては、ジャクソン法[1],[2]の構造一致問題、構造不一致問題、関係データの処理などがある．紙面の都合上、ここでは構造一致の問題しか扱わない．

ここでは、プログラム、仕様、知識の全てが、対象物と二項関係の集まりによる統一的なモデル（意味モデルと呼ぶ）を用いて表現される．モデルを用いてプログラムの自動生成を行なうシステムとしては、MODEL システム[3]、SLシステム[4]などがあるが、これらがデータフローの解析を中心としているのに対し、本方式では、①仕様とプログラムが別々に表現された（同時にそれらの間の対応が示された）モデル（属性変換文法（Attribute translation grammar[5]）の式と関連性がある）、②構造表現を利用したゴール指向的な問題解決方式、を採用している点が大きな違いである．

ジャクソン法における構造一致のプログラムは、単一記号先読み文法(single symbol lookahead grammer)に従う入力を走査する受理機(accepter)[6]に出力を作り出すアクション部分を追加した変換機（transducer)である．本方式では、問題の出力仕様をゴールとみて、それを構成する複数のサブゴールを、それらの定義形（ユーザ定義あるいは知識）を用いて入力仕様に還元を計り、入出力木の構造一致性を検査する．構造一致の場合は、その過程において見いだされた操作をアクション部分に用いる．これを入力の構造だけから定まる受理機と融合することにより、変換機のプログラムを得る．

本論文の構成は次のようである．第２章では、システムにおける情報を統一的に表現する意味モデルの表現形式を簡単に紹介し、同時に本論文で使用する知識とその表現について述べる．第３章では、構造一致の問題について（多段先読みや後戻りの必要なものは除く）、意味モデルの特性を効果的に利用したプログラム合成アルゴリズムを詳細に述べる．例題としては、文献[5]に例題として取り上げられているトランザクション問題を用いた．

2．仕様とプログラムのモデル及び合成に関連する知識

2．1　知識の種類

ここで扱う抽象プログラム合成に関連する知識の種類を以下に記す．

① 対象物の型とそれに関する知識構造

本論文で用いる対象物の型としては、データ（図上では、ファイルあるいはストリームとして外部にあるデータ集合は記号○で、記憶内部にあるデータ集合は記号◎で表わす）、算譜構成要素（記号▩で表わす）、識別名（記号◇で表わす）、及び状態（内にグラフを含む長方形で表わす）の四つがある（２．２、２．３参照）．実際に用いられる対象物(object)は、generic-objectのインスタンスである．

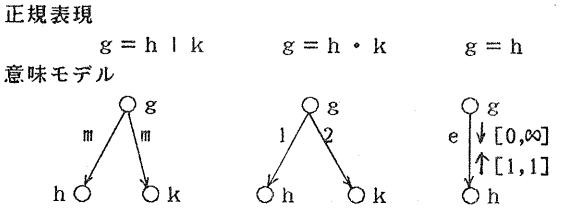
② 関係の上位下位関係（推移律などに関する）．　③ データ型（自然型と役割型がある）及びその上位下位関係（２．４参照）．　④ 機能の点からみた操作の上位下位関係の分類．

⑤ スキーマ（データ型に関連した知識として定義形の形で保持される．ユーザの与える定義形とは異なる）（２．４参照）．　⑥ 列に関する公理（３．２参照）．　⑦ 葉の情報の伝播に関する規則（３．３参照）．

2．2　データの構造記述[7],[8]

データの構造は、要素としてデータ集合を用い、それらの関係による繋がりで表わす．意味モデルでは、データ集合はユニークな固有名（英大文字で表わす）を持つラベル（英小文字で表わす）付きノードにより表わされる．ラベルはデータがとる値域を表わしており、（状態によりその定義が異なってもよい）データ型である．このデータ型は、２．４に示すように、データ型間の上位下位関係による知識の基本構造の下に連結される．データ構造の木の根となる単一データも、便宜上単元からなるデータ集合と考える．データどうしの関係はラベル付きのエッジで表わされる．データを持つ

図１　列データ構造の正規表現とその意味モデル表現



エッジには、ノードと同じように、個々のエッジにユニークな固有名をつけ（但し図示はしない）それに関する記述を作る．図１に列データ構造の正規表現に対応する意味モデル表現を示す．図の（ｃ）の関係e のエッジに付けられた角括弧内の数値のペアは、矢印の方向に見た一方の集合の一つの要素に対応する他方の集合の要素の最小数と最大数のペアを表わす．この表わし方によれば、

$A::=B^{*}$ は A->B: [0,∞], B->A: [1,1]、
$A::=B^{+}$ は A->B: [1,∞], B->A: [1,1]、
で表わされる．

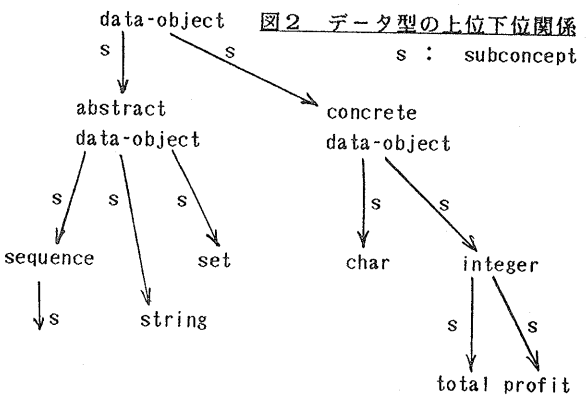


図２　データ型の上位下位関係
s ： subconcept

意味モデル表現では、以後入出力仕様に現われるデータ構造表現のことを入力木、出力木と呼ぶことがある．これは、入出力仕様は一般にグラフとなるが、データの基本構造は木で表わされるからである．

入出力データ構造は、実体(x)、実体本体(x-body)、（実体の）属性、ヘッダ、デリミッタ、ダミーノードなどの要素により構成される．実体本体は、さらに繰り返しの要素からなる部分を指す．実体の属性は、２．４に述べる定義形によりその意味が定義される．これらの、単一要素、連接、離接における組み合わせのパターンの例は図５にその一部が示されている．入力木に存在する実体に対応し、属性やヘッダ、デリミッタが付け加わった出力木の実体は入力木のそれと同じラベルを持つ．一つの木で表わされるデータ構造の複数の葉を、ある見方から、共通な上位概念を持つ下位概念の具現化したものとして考えることも可能である．これは、入力データを指す大域変数のデータ型を抽出する場合に行なわれるが、それ以外にも、中間ファイルにおいてデータを別の見方により一括して処理する場合、構造一致の検査の過程で簡単化を計るときなどにおいて用いられる．

２．３　プログラムの構造

意味モデルでは、プログラムの構造はノードとして算譜構成要素から構成される．これに属する型としては、繰り返し文型（その下位概念として、while 型、repeat型）、判定文型、割りつけ文型、操作型、関数型、ダミー型、プロセス型などがある．定義（generic-object）を表わすラベルは英小文字、適用を表わすラベルはそのインスタンス(object)で英大文字で表わす．割りつけ文型及び操作型に属する算譜構成要素は、program codeの下にその文を記す．繰り返し文型及び判定文型の算譜構成要素の場合は、それぞれ反復条件（または終了条件）及び判定条件の式を持たせる．ダミー型とは、プログラムの構造を抽象構文木で表わすときに、複数の枝を分岐させるために導入されるノードをいう．

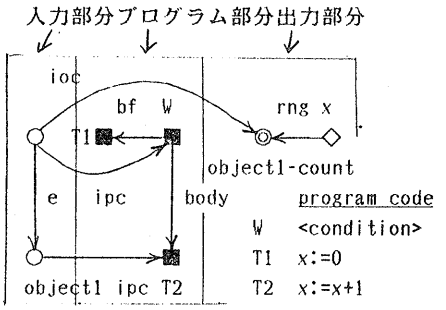
プログラムの構造を示す算譜構成要素間の関係としては、1,2, ,then,else,bodyなどは、抽象構文で普通用いられている通りである．このほか、繰り返し型の算譜構成要素とその前にくる初期設定のそれとを、関係before（図ではbfと略す）で結ぶ．

２．４　データ型とそれに関連する知識

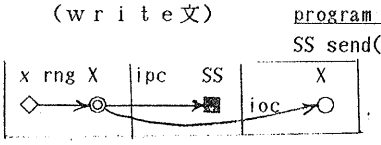
データ型は、図２に示すように、上位下位の関係で構造化されている．データ型としては、自然型（図２の set, integer など）と役割型（図２のtotal など）とがある．

図３　いくつかの定義形

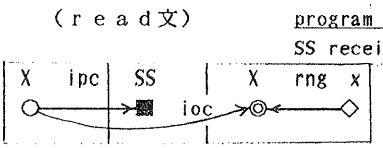
（ａ）　出現回数の計算



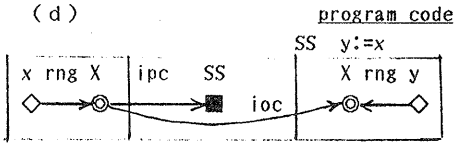
（ｂ）　ｓｅｎｄ文
（ｗｒｉｔｅ文）



（ｃ）　receive 文
（ｒｅａｄ文）



（ｄ）

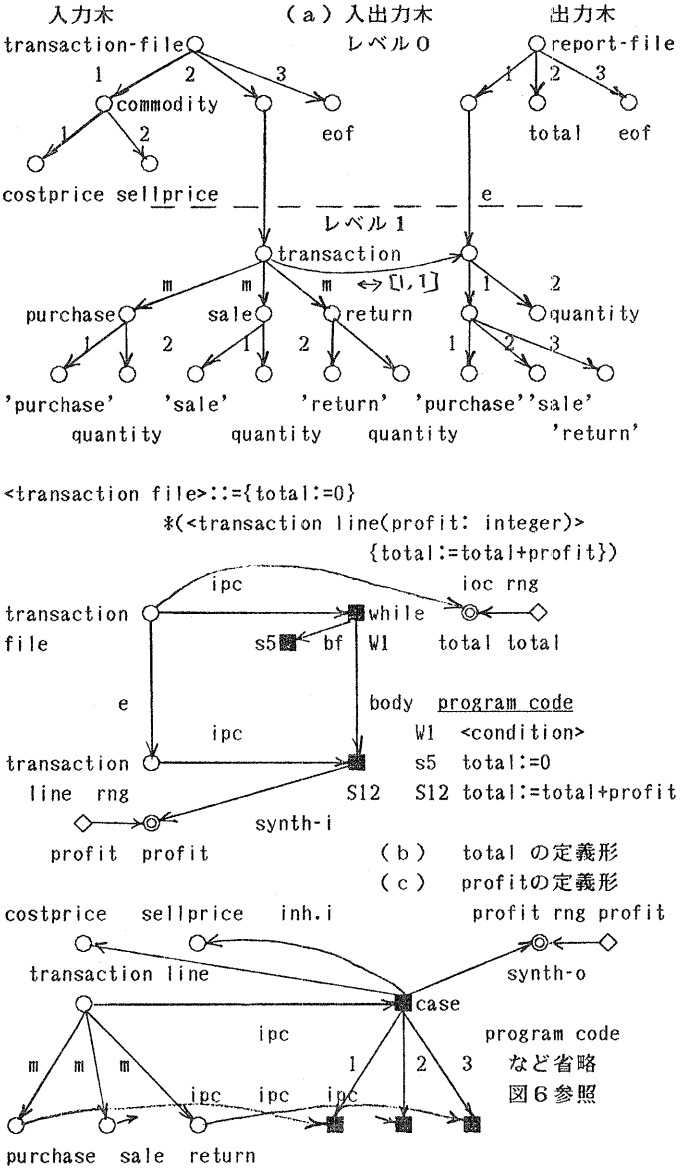


下位のデータ型はとくに指定のない限り、上位のデータ型の操作を継承する．たとえば、役割データ型total はその上位の自然データ型integer の演算 +1 (実際には、ADD1という名称の手続き型の操作)を継承する．また各データ型は、それに関連するスキーマを知識として持つことがある．

定義形

定義形は入出力仕様からプログラムを導出する際に用いられる．（入力木に現われない）出力木の葉のラベルの語の計算上の意味は定義形により与えられる．これらの定義形の形式は、＜入力部分、プログラム部分、出力部分＞の３つ組からなる（図３、４参照）．定義形のプログラム部分はアクションを中心にした記述になっている．定義形の形をとるものは、ユーザ定義形、データ型の操作、スキーマ、などがある．スキーマは一般化された内容のプログラムコードの集まりを含むものである．図３にいくつかの定義形を示した．図で、関係ipc（input-program-correspodenceの略）及びioc（input-output-correspodence)は、それぞれ入力部分に現われるデータ集合と対応するプログラム部分中の算譜構成要素及び出力部分のデータ集合を結ぶ関係、rng（rangeの略）は、識別名の一種である変数とその値域のデータ集合との関係、を表わす．プログラム部分の算譜構成要素に付けられたプログラムコードは、入力部分あるいは出力部分に現われる変数を用いている．入力部分のグラフが構造を持つ定義形の記述内容は、対象問題の構造をじかに反映したものではなく、列の公理の併用により対象として広い範囲の構造をカバーしていることに注意したい．

図４　トランザクション問題の入出力木と語の定義



２．５　問題の仕様記述

ユーザは、問題の仕様として次にあげるものを指定する．

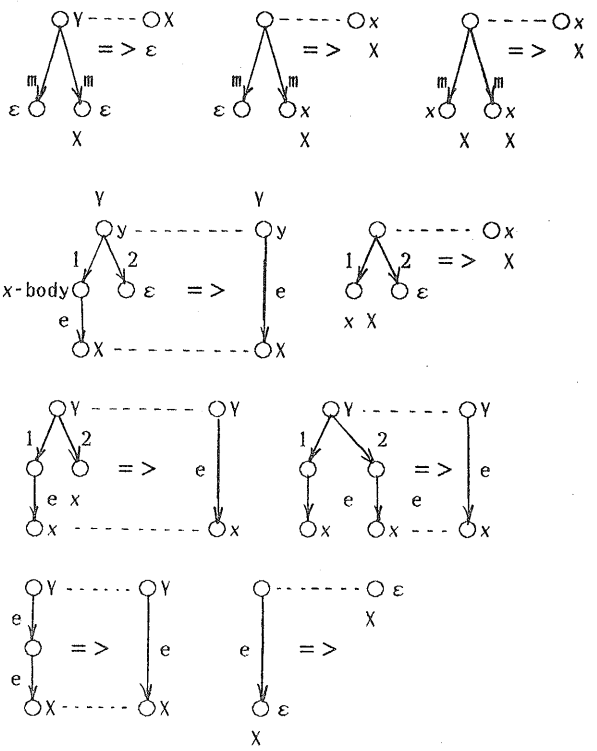
① 入力及び出力のデータの構造．　② 役割型の定義．　③ （システム知識構造にある）自然型と入出力データの役割型のとの上位下位関係（あるいは、必要ならば、上位の役割型の導入）．

④ 出力木の語の定義形．　⑤ ヘッダ、デリミッタ．　⑥ ｒｅａｄ操作の対象となる単位、その内部構造（識別名とデータ）．あるいは、入力木の葉に共通する上位のデータ型及びそのデータ集合を指す大域変数．⑦ 入力木の葉（節）と出力木の葉の対応（１対１、あるいは多対１）．

例題（トランザクション問題）

トランザクション問題は、入出力木間の節の対応ずけと実体の属性情報の取り扱い、及び後者の定義形においては属性変換文法における合成属性と継承属性の両方の具体例を含み、プログラム設計のアルゴリズムを簡潔に示すのに好都合な問題である．図４（ａ）にこの問題の入出力仕様の意味モデル表現を与える．（ｂ），（ｃ）は、出力木の葉に現われる(profit-)total、及びprofitのユーザ定義形である．（ｂ）の中の式は、図の意味モデルに対応する属性変換文法の式である．

図５　列の公理（の一部）



３．　問題解決法

３．１　木の前処理

ある木の節（あるいは葉）から根へと辿るとき、途中に出会う関係e の回数をその節（あるいは葉）のレベルという（木の根のレベルを０とする）．同じレベルにあり、根に至る関係e の枝が同一な節あるいは葉の集合を一つのノードにまとめたものを凝集ノードという．

入力木及び出力木を走査して、ヘッダ及びデリミッタ以外の、入出力木の葉として共通に存在するもの、及び出力木の葉としてのみ存在するもの（実体の属性）のリストを作る．

３．２　構造一致の判定と演算の枚挙

構造一致の判定は、木の根から葉の方向に再帰的に辿って行なう．これには、（１）入出力木の対応する共通ラベルの全ての葉から根に至る分岐上の節間の写像関係の成立の検査、及び、（２）出力の葉としてだけ存在する全ての葉の定義形による入力への還元可能性の検査、が必要である．この構造一致の検査は、人工知能の問題解決法における計画(planning)に相当する．計画には、図５に示す列の公理の他に、２．４で述べた定義形が用いられ、問題出力（あるいは入力）と定義形出力部分（あるいは入力部分）、あるいは定義形どうしの入力部分と出力部分に現われるデータ集合どうしあるいは変数名どうしの結びつきが束縛(binding)リストに登録、管理される．これら両方の検査に合格（即ち成功）した時、束縛リストに残されたその検査の軌跡から、使用される定義形が確定し、用いられる算譜構成要素の集まりが抽出される．

後に述べるように、これら二つの検査で行なわれる内容は異なるが、列の公理を用いた書き換えによる問題部分木の簡単化手続き（Ａ）は共通に使われる．

この手続きの内容は次の通りである．ｃ－ｌｉｓｔはその手続きで参照するリストである．
問題部分木のコピーを作り、その根から再帰的に辿り、戻りのフェーズで書き換えを行なう．葉の場合は、そのラベルを持つノードがｃ－ｌｉｓｔに載っていればそれに、載っていなければε（空）に書き換える．節の場合は、その節を根とみた部分木のパターンに整合する左辺を持つ列の公理を選んで、その右辺のように書き換える．データの構造の木表現は、この簡単化の変換を行なうのに都合がよい．

（１）の検査

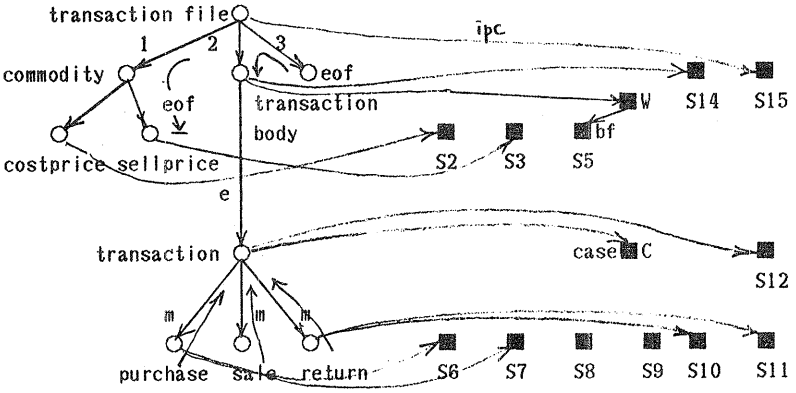
出力木の凝集ノードが一つの葉となる場合に枝ごとに行なう．最初に、手続きＡにより出力部分木の簡単化を行なう．そのとき、ｃ－ｌｉｓｔの値としては、入力部分木の葉と共通のラベルを持つ出力部分木の葉のリストを与える．次に、得られた出力木の葉をｃ－ｌｉｓｔの値として、手続きＡにより入力の部分木の簡単化を行なう．簡単化された入力部分木と出力部分木との整合性を調べる．整合が取れる場合には、ｃ－ｌｉｓｔに載っている入力木の葉と、ｗｒｉｔｅ文定義形（図３（ｂ）参照）の入力部分のノードとの束縛

を束縛リストに登録する．整合の取れないときは構造不一致として戻る．

出力木の葉として現われるヘッダ、デリミッタの処理

出力木の葉として現われるヘッダ、デリミッタは、（1）の検査においては無視されるが、それを出力する算譜構成要素は問題の解を与えるプログラム木の構成要素として必要である．それら用の定義形が用意され、その入力部分にある節は対応する問題入力木の節とのペアとして束縛リストに登録される．

図６　トランザクション問題における入力木への出力仕様の還元



program code

```
S2 costp:=costprice; S3 sellp:=sellprice; S5 total:=0; W input = eof;
C  (1) input='purchase' (2) input='sale' (3) input='return';
S6 profit:=0; S7 send(reportline('purchase',quantity));
S8 profit:=(sellprice-costprice)*quantity; S9 send(reportline('sale',quantity)
); S10 profit:=(costprice-sellprice)*quantity; S11 send(reportline('return',qu
antity)); S12 total:=profit+total; S14 send(totalsline(total)) ; S15 send(eof)
```

（２）の検査

各レベルごとにそのレベルのサブゴールがある限り以下の手続きを行なう．　現在のレベルのサブゴールリスト(p-s-list ）にあるノードを一つ取り出し、出力部分がそれに関連するノードを持つユーザ定義形あるいは知識ベースにある定義形の適用を行なう．　定義形入力部分の木に出て来る葉のノードのリストをｃ－ｌｉｓｔに作る．それをもとに問題入力部分木に簡単化手続きＡを適用して得られた書き換え結果と、指定された定義形の入力部分との整合性をみる．その結果、

①　整合のとれない場合，　定義形の入力部分が、同一レベルの節だけからなるとき、その節をp-s-listに登録する（ただし、すでに展開生成済みのノードは再登録しない）．異レベルの節を含むとき、構造不一致で戻る．

②　整合のとれた場合、　問題入力木の節名と対応する定義形の入力部分の木の節名との束縛を作り、束縛リストに登録する．さらに、定義形の入力部分に指定された継承属性があれば、それを求めて問題入力木を捜し、ｒｅａｄ文の単位となる葉まで辿る．これは、用いる規則は異なるが、３．４に述べる入力木の葉の情報の伝播と同じ形式で行なわれる．見いだされれば、対応する入力木の葉との間に図３（ｄ）に示した定義形を介在させて、それらの束縛を束縛リストに登録する．定義形の入力部分の示された合成属性があれば、次のレベルで還元されるべきサブゴールリストに登録する（次のレベルの出力木の葉で満たされる場合もある）．

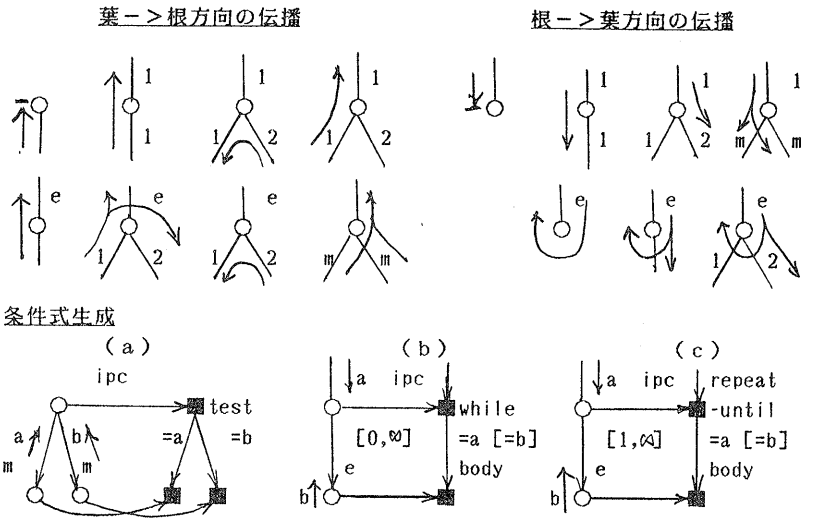
全体の制御、構造一致の場合のアクション部分の入力木への附加

上の二つの検査は、理解しやすいように別々に説明したが、実際には融合したものが再帰的に行なわれる．問題の入力と出力が構造一致と判定されたときは、問題入力木を辿り、その各節から束縛リストに登録された情報をもとに、定義形の入力部分を見いだし、定義形の入力部分、プログラム部分、出力部分、の各グラフ及びそれらの繋がりを作業領域に具現化(instantiate）していく（図６参照）．定義形の出力部分のノードが束縛リスト上の値でさらに別の定義形のノードに繋がっている場合には、続いてその定義形も具現化していく．このようにして作られた構造を利用して、３．４のプログラム木組み立てがなされる．

例題（図６参照）　　レベル０では、ユーザにより与えられたtotal の定義式の意味モデル（図４（ｂ））を用いて、その入力部分が入力木と整合が取れるかどうかを確かめる．これは入力木の他の葉をεとして列

の公理を適用することにより、確かめられる．（構造一致後は、プログラム部分の算譜構成要素s5,S12,W1 を作業領域に設け、入力木のtransaction ノードから関係ipc で繋ぐ．） そしてそのS12 の関係synth-i で示される合成属性入力profit P1 を次に還元さるべきサブゴールとして登録する

図７　葉の情報の伝播と判定文及び反復文の条件式生成



レベル１では profitの定義形（同図（ｃ））の還元、及び出力木のレベル１の部分の還元がなされる．profitの定義形の還元においては、入力部分のグラフが入力木と整合が取れる．（構造一致後は、プログラム部分に現われる算譜構成要素が作業領域に作られ、対応する入力木のデータに関係ipc で結ばれる）．それから、定義形の入力部分にある継承属性入力（関係inh-i で示される）sellprice,及び costpriceを求めて、入力木を根の方向へ探索がなされ、commodity の葉にあるのを見いだす．（構造一致後、そこへ定義形の変数への割りつけの文が挿入される）．出力木のレベル１の部分の還元に関しては、列の公理の適用により、出力木のreportlineの部分木の部分が入力木のtransaction lineの部分木と同型に変換できる（構造一致後は、入力木のpurchase、sell、及びreturnの葉に関係ipc でそれぞれを印刷する算譜構成要素S6,S8,S10 が付けられる．

３．３　葉の情報の伝播及び反復文・判定文の条件式の生成

入力の構造を表わす文法が単一記号先読み文法であるための条件は次の二つである．

条件１：　＜Ａ＞：：＝$\alpha_1 | \alpha_2 | \cdots | \alpha_n$ のとき、各分岐の区別ができるためには、α_i から導出されるstringの最初に表われる全終端記号の集合が他のα_j のそれと異なっていなければならない．

条件２：　＜Ａ＞から導出されるstringの最初に表われる全終端記号の集合が＜Ａ＞に続く部分のそれと異なっていなければならない．（これは＜Ａ＞：：＝a^* の場合を考えれば分かる．）

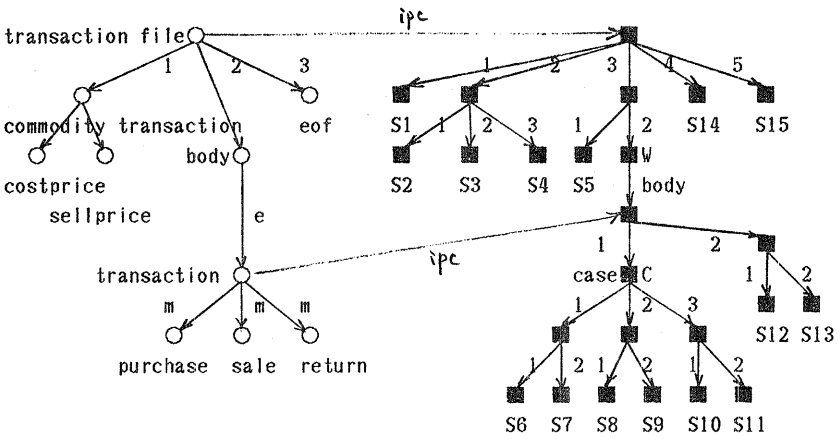
これらの条件からも分かるように、単一記号先読み文法による入力記述から、プログラムを組み立てるには、分岐の判定の条件式や繰り返しの終了（あるいは継続）条件式を定めるために、データの文法記述における終端記号の情報が必要である．データの構造は意味モデルで記述すると木で表わされ、終端記号は木の葉の位置を占める．木の葉の情報は、図７に示す規則に従って葉から上（根方向）へ伝播させることができ、入力木の節と対応するプログラム木の判定文や繰り返し文の節を見いだしてそれらの条件式を組み立てることができる．

木の節のパターンに応じた葉の情報の伝播方向は、図７から分かるように、上（根の方向）から下（葉の方向）へ伝播する場合と、下から上へと伝播する場合の両方の場合がある．上から下へと伝播して来た場合には、終端記号の葉に達した時は、そこで伝播が止まる．下から上へと伝播して来た場合には、a^* は一回も実行されない場合も含むので、前（下）から後ろ（下）へと伝播する．これに対し、a^+ は最低一番回は実行されるので、それはあり得ない．図７の条件式の生成は、伝播の途中出会う節のパターンに対応するプログラム木の判定文あるいは繰り返し文の算譜構成要素に付ける条件式を表わしている．ここで、=a（あるいは≠a）とは、入力を読み込むことで得られた大域変数の値が、矢印の方向から伝播してくる葉のラベルと一致する（しない）条件式を表わす．repeat-untilは反復の終了条件式が複数個あるときはｏｒで結ぶ．while は反復の継続条件式が複数個あるときは＆で結ぶ．

例題（図６）

場合分けの算譜構成要素においては、それぞれの場合の判定の際に比較の定数として用いられる'purchase'、'sell'、'return'がそれぞれの葉からもたらされる．レベル０における入力木のノードtransaction file bodyに関係ipcで結ばれる繰り返しの算譜構成要素W1は、その反復条件を入力木における後ろのeof のデータから得る．

図８　トランザクション問題のプログラム木



program code

```
S1 receive1; S2 costp:=costprice; S3 sellp:=sellprice; S4 receive2(input);
 S5 total:=0; W input = eof; C (1) input='purchase'  (2) input='sale'
(3) input='return'; S6 profit:=0; S7 send(reportline('purchase',quantity));
S8 profit:=(sellprice-costprice)*quantity; S9 send(reportline('sale',quantity)
); S10 profit:=(costprice-sellprice)*quantity; S11 send(reportline('return',qu
antity)); S12 total:=profit+total; S13 receive(input);
S14 send(totalsline(total)) ; S15 send(eof)
```

３．４　プログラム木の組み立て

プログラム木の組み立ては、入力木を再帰的に辿り、葉から根の方向へと戻るときになされる．

その際、入力木の節から関係ioc により半順序に付けられたデータの繋がりを辿り、それらのデータに関係ipc で繋がれた算譜構成要素をトポロジカルソートにより全順序化して、新たに導入したダミーノードの下に統合する．関係beforeについても、新たにダミーノードを導入して処理する．

ｒｅａｄ文の導入もこの過程でなされる．ｒｅａｄ文は、入力木の葉毎（但し、eof に相当するところは除く）に、また先読みのためレベル０の始めのところにひとつ置く．

例題（図８）

図６に示した入力木に附加された全ての算譜構成要素、及び、追加されたｒｅａｄ文をまとめて、プログラム木が完成する．

３．５　プログラム木のコード化

プログラム木をpreorderに辿ってコード化を行なう．変数のデータ型を宣言するためには、データのgeneric-objectを求め、その役割型のすぐ上位の自然型を取り出す．

４．　おわりに

意味モデルとよばれる、仕様、プログラム、及び知識の統一的なモデル表現を用いて、ジャクソン法の構造一致問題のプログラムを合成する方法について述べた．この方法は、仕様の解析とプログラムの組み立てにモデルの構造表現機能を活用した問題解決を行なう方法である．また、本方法と属性変換文法の考え方との関連性も指摘した．実際問題を解く際には、入力走査部分とアクション部分の分離と融合、同一入力に対するアクション部分の重畳、などの操作が必要となるが、仕様とプログラムを分離し両者の要素間の関連を付けた定義形の表現形式は、そのような操作を行なうのに好都合である．

構造不一致の場合のプログラム合成法、及び抽象プログラムの実現変換法については、別の機会に譲る．

おわりに、日ごろお世話頂く棟上ソフトウエア部長、有益な議論をして下さる言語処理研究室及び情報システム研究室の方々に感謝致します．

参 考 文 献

[1] 峰尾欽二：プログラミング方法論（ジャクソン法），情報処理，Vol.23,No.11,pp.1063-1074 (1982).
[2] Jackson,M.A.: Principles of Program Design, Academic Press(1975). （鳥居宏次訳，構造的プログラム設計の原理，日本コンピュータ協会（1980））.
[3] Prywes,N. & Pneuli,A.: Compilation of Nonprocedural Specifications into Computer Programs, IEEE Trans. on Software Engineering, Vol.SE-9, No.3, pp.267-279 (1983).
[4] 渡辺　敏、岡本　務、福田　満、中村雄三："設計用言語（ＳＬ）の処理方式"、情報処理学会ソフトウエア工学研究会資料４２－４　(1985).
[5] Coleman,D., Hughes,J.W. and Powell,M.S.: A Method for the Syntax Directed Design of Multiprograms , IEEE Trans. on Software Engineering, Vol.SE-7,No.2,pp.189-196 (1981).
[6] Coleman,D.: A Structured Programming Approach to Data , Macmillan Press (1978).
[7] Abrial,J.R. : Data Semantics,in Data Base Management, Klimble,J.W. and Koffeman,K.L.(eds.), North-Holland (1974).
[8] Chen,P.P.: The Entity-Relationship Model, ACM Trans. on Database Systems, Vol.1,No.1 (1976).