Vol. 28 No. 10　　情　報　処　理　　Oct. 1987

解　説

2. 方　式

2.6 実行可能な仕様記述†

佐伯　元司††

1. はじめに

ソフトウェアの要求定義を開発段階で正確に定義することは難しく，そのためユーザの要求を再確認するためのプロトタイピング手法が注目されている[1),2)]．プロトタイピングの実現方法としては既存のプログラム言語を用いたり，過去に作られたモジュールを加工，結合（再利用）したりしてプロトタイプを作る手法があるが，その中の手法として，記述された要求仕様自身をプロトタイプとして利用する，すなわち仕様自身を直接実行させ，仕様の分析や検査を行う手法がある．

本解説では，実行が可能であるように設計された代表的な仕様記述言語とそのシステムについて「実行」という立場から特徴を整理し説明する．「実行可能な仕様記述システム」という言葉は，仕様からプログラムを合成するシステム，たとえば本特集号でも述べられているプログラム変換や部品合成を行う広範囲のシステムも指すが，本稿では計算の手順が対象領域固有の知識によらず，記述言語のセマンティックスのみに従うシステムに限定する．このため，従来の検証を目的とした仕様記述言語が宣言的な意味のみをもっていたのと異なり，実行可能な仕様記述言語は機械で実行可能ななんらかの計算モデル，つまり操作的な意味をもっていなければならない．したがって本稿で述べる仕様は，記述された手順と本質的に異なったより効率的な手順で実行されることはない．本稿で取り上げるシステムで使用されている技術には，従来のプログラム言語のコンパイラ，インタプリタの技術によるところが大きいものもある．

2. 実行可能な仕様記述

2.1 Operational Approach と Functional Approach

要求の仕様化は，現実世界の対象物から必要な情報のみを抽出し，モデル化する過程である．どのようなモデルを用いるかによって，仕様記述言語も種々のものが設計されている．たとえば，GIST 言語[7)]は対象物を relational model によって捉えるべく，その構文要素が設計されている．実行可能な仕様記述言語は，仕様化のモデル以外に実際に機械上で実行するための計算モデル，つまり操作的意味をもっている．本稿では，現在までに開発が進められている実行可能な仕様記述言語を operational approach[3)] に基づくものと，そうでないものとに分類し，解説する．この捉え方は，対象システムのどの部分に注目するかで分類したもので，前者はシステム内部に，後者は外界とシステムとの境界に注目する手法である．後者は，システムの入力と出力の関係のみに注目する手法で，本稿では functional approach と呼ぶことにする．入出力関係は，関数，述語，等式，結合子[34)]などの数学的概念を用いて記述される．これに対し，operational approach は，システムにある入力がなされたとき，出力をするまでのシステムのふるまい (behaviour) を仕様として記述する．具体的には，システムの内部状態をモデル化し，その内部状態の遷移を規定する．operational approach は，仕様化のためのモデルと計算モデルとが非常に接近しており，一体化されている場合もあるが，functional approach では両者はかけ離れている．

これら2つの手法は排他的であるというのではなく，たとえば operational approach における状態遷移を，入力を遷移前の状態，出力を後の状態とみなしてその入出力関係を定義すれば，functional approach 用の仕様記述言語でも operational approach に応用できる．

† Executable Specifications by Motoshi SAEKI (Department of Computer Science, Faculty of Engineering, Tokyo Institute of Technology).
†† 東京工業大学工学部情報工学科

2.2 実行システム

仕様を直接実行するとは，仕様として書かれた内容をシミュレートすることである．実際に機械上で実行され，出力するという点では本質的にプログラムと同等であるが，実際のシステムが置かれる環境下での実行ではないし，現実の時間軸に沿った実行でもない．これは，シミュレートすることにより発注者の意図を再確認し，正確な，虫のいない要求仕様を作成することがねらいである．それゆえ，実行効率は人間と機械の許容範囲内であればよい．このシミュレーションという立場での実行システムは以下のような特徴をもっている．

1) 未定義部分を含んだ仕様が実行可能

実際のプログラムと異なり，仕様チェックの対象外となっている部分が，現実のシステムどおりに厳密に機械で実行されなければならないということはない．未定義部分が含まれていても，実行システムがなんらかのダミーコードを埋め込んで実行したり，人が対話的にその未定義部分の出力データを与えたり，実行を制御したりすることができるようになっている．

2) 時間的な制約条件が実行可能

「5秒以内に応答せよ」といったシステムの性能を表す時間的な制約条件も重要な要求である．仕様の実行は実際の時刻ではなく，シミュレーション上の時刻に沿って行われるため，仕様実行システムがシミュレーション上の時刻を生成し，その時刻に従って実行をスケジュールしていく．

3) 対話的に実行可能

仕様の実行時に人が介在し，その実行を制御できるという機能をもつ．これは，プログラミング支援システムでのデバッガの機能に該当するが，実行システムと一体化されているのが普通である．たとえば，実行を中断させて，実行状況をチェックしたり，非決定的なプロセスのスケジュールを意図的に行ったりすることが可能である．さらには，シミュレーション上の時刻を変えたりすることも可能なシステムもある．

3. Operational Approach

Operational approach の代表的な手法として，システムを1つの有限状態遷移機械として捉える手法，複数のオブジェクトとその間の関係として捉える手法 (relational model)，複数のプロセスとして捉える方法，論理的手法などがある．

本章では，これらの手法の中から代表的な仕様記述言語及びシステムとしておのおの RPS[5]，GIST[7),8)]，PAISLey[12),13),37)] 及び時間論理を用いた言語[14)~16)] を選び，解説する．

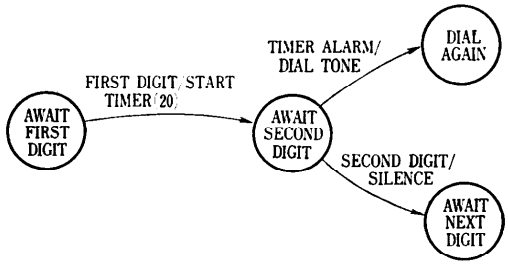
3.1 RPS

RPS は，GTE Laboratories で開発されている，実時間システムのための要求定義環境を与えるシステムで，その仕様記述言語は，RTRL と呼ばれる有限状態遷移機械をベースとした言語である．この言語による電話の交換器システムの仕様記述例の一部を図-1(a)に，図-1(b)にその状態遷移図を表す．このような実時間システムに対して課せられる時間的な制約は，1) システムの反応に関するものと 2) 外界からの刺激 (stimulas) に対するものと2種類ある．図-1 の %TIMERS セクションに記述されたタイマ定義は，「ユーザ（外界）は1桁目をダイヤルして（外界から

```
%FEATURE local_to_local_call ;
%TIMERS
   collect_digit_timer (20) --
   the user has 20 seconds to dial each
   digit of a phone number ;
%REQUIREMENTS digit_collection ;
   INSTATE await_first_digit ;
   TRANSITION ;
      (first_digit) :
         START collect_digit_timer ;
         NEWSTATE await_second_digit ;
      END TRANSITION ;

      INSTATE await_second_digit ;
      TRANSITION ;
         (second_digit) :
            SEND silence TO calling party ;
            NEWSTATE await_next_digit ;
         (collect_digit_timer.alarm) :
            SEND dial_tone TO calling party ;
            NEWSTATE dial_again ;
      END TRANSITION ;
```

(a) RTRL による記述



(b) 状態遷移図

図-1 RTRL による仕様記述例[6)]

の刺激）から 20 秒以内に次の桁をダイヤルしなければいけない」ことを記述するためのもので，この制約は後者の例である．

RTRL によって記述された仕様は，拡張状態遷移行列に変換され，Feature Simulator に入力され，シミュレートが行われる．拡張状態遷移行列は，通常の有限状態遷移機械の遷移行列に状態遷移に関する時間の制約を付加したものである．Feature Simulator は，この遷移行列より可能な状態遷移を選択し，状態遷移を引き起こすことによって仕様を実行する．実行中に，外界からの刺激待ち状態に陥いると，Feature Simulator はプロンプトやメニュを表示し，ユーザに刺激の入力を促す．このようにして，対話的に実行が進められていく．RTRL を状態遷移行列ではなく，テストプログラム記述言語に翻訳し，プログラムテストを行う環境で実行させ，虫を発見しようとする研究[7] もある．

RPS のような状態遷移機械を基礎とする手法を大規模システムに適用する場合，どうしても状態数が多くなるため，ユーザがまず仕様を記述する段階での方法論（抽象化やモジュール分割法）やツールの整備が必要となるであろう．

3.2 GIST

3.2.1 GIST 言語

GIST は，USI の Balzer らによって開発が進められているシステムで，仕様記述の対象となるシステムを，複数のオブジェクトとオブジェクト間の関係として捉える．システムの内部状態は，システムに存在するオブジェクト及びその属性値と，それらのオブジェクト間に成立する関係によって表される．システムのふるまいは，これらがどのように変化していくかを，オブジェクト／関係の生成，消去という relational database アクセスの基本操作を用いて記述される．GIST による仕様は，以下の3つの部分から構成される．

1) 構造の宣言 (structural declaration)

オブジェクトの型（オブジェクトが持つ属性も含む）やそのインスタンス，オブジェクト間に成立する関係 (relationship) を宣言する．

2) 刺激/応答規則 (stimulas-response rule)

システム内で起こる動作を記述する．その動作がひき起こされる状態 (stimulas) と，動作の内容 (response) を記述する．

3) 制約 (constraints)

関係に加わることのできるオブジェクトの制限や，動作によってひき起こされる状態変化の制限などを記述する．

図-2 に GIST 言語で記述された「港の管理システム」の例を示す．予約語 type から始まる部分が，構造の宣言を行っている部分である．たとえば，port 型のオブジェクトは属性として，Pier(桟橋)，harbor をもち，その値はおのおの pier 型，ship 型のオブジェクトである．値域を表す型名に続く予約語 multiple, unique, optional, any はオブジェクト間の対応状況を表す．たとえば，port の1つのインスタンスは複数個の pier 型のインスタンスを属性としてもつことができるが (multiple)，1つの pier 型のインスタンスは1つの port 型のインスタンスしか応答づけられない (unique).

刺激/応答規則は，agent セクションの action 部に

```
begin
  type port(Pier | pier :multiple ::unique,
            harbor | ship :any ::optional);
  type pier(Handle | cargo :multiple,
            Slip | slip :multiple ::unique);
  type slip();
  type ship(Carry | cargo :any, Destination | port,
            berth | slip :optional ::optional);
  always required Berth_Are_In_Ports
     for all s | ship |T
        s :berth=$=>s ::harbor :Pier :Slip=s :berth;
  type cargo()optional
              supertype of<grain() ; fuel()> ;
  always prohibited Fuel_And_Grain
     there exists s | ship, g | gain, f | fuel ||
                    s :Carry=g and s :Carry=f ;
  agent manager(Port | port :unique ::unique)
     where action MoveShip[s | ship, p | pier]
             precondition  s ::harbor=manager :Port
             precondition  manager :Port :Pier=p
             definition  update :berth of s to p :Slip ;
           action Loadship[s | ship, c | cargo]
             precondition  s :berth ::Slip :Handle=c
             precondition  s ::harbor=manager :Port
             definition  insert s :Carry=c ;
           action AssignCargo[c | cargo, p | port]
             definition  Loadship[p ::Destination, c]
     end
end
```

図-2 GIST による仕様記述例[9]

記述される．図-2 の例では，agent manager の1つのインスタンスは port 型のインスタンスを唯一もっており，その port に対して MoveShip, LoadShip, AssignCargo で指定された動作を行うことができる．たとえば，MoveShip は ship s を pier p に移動する動作で，precondition で指定された条件を真にする状態でのみ起動することができる．MoveShip を行うことのできる条件は，動作主である manager が所有している port にpier p が属していなければならないことと，その manager の port に ship s が停泊している (harbor) ことである．ある状態において適用可能な動作が唯一になる，つまり precondition が真である動作とそのオペランドの対が唯一になるとは限らない．このように可能な状態遷移は，集合として規定される．

動作の内容の定義は，オブジェクト/関係に対する4つの基本操作，つまり 1) 新しいオブジェクトの生成，2) 既存のオブジェクトの消去，3) 新しい関係の挿入，4) 既存の関係の削除，を通常の手続き型のプログラム言語でよく知られている逐次実行，条件分岐，繰り返し，手続き呼び出しといった形式を用いて行われる．

制約は，上記2つの記述要素で規定された状態遷移集合から不要な状態遷移を排除する役割をもっている．図-2 の例では，予約語 always から始まる2つの statement が制約を表している．required はすべての状態で要求される「肯定の制約」，prohibited は逆に禁示されている「否定の制約」を表す．前者の例は，すべての ship s について，s が berth (停泊所) をもっているならば，その berth は s が停泊している port に属している pier の slip (船架) となっていなければならないことを表している．

3.2.2 GIST 仕様の実行

GIST インタプリタは，ある状態で適用可能な動作が複数個あった場合，それらのうちの1つを選択し，実行する．動作本体の実行は，通常の手続き型言語の場合と同様である．実行した後の状態が制約を満たしていない場合は，backtrack し，適用する操作の選択をやり直す．このような実行を仕様の検査目的で使用するには，複数の入力データを与え，複数回の実行を行って出力を調べる．そこで，GIST システムでは，複数回の実行という手間を省くために，1度の実行で出力の性質を得ることのできる記号実行 (symbolic execution) による手法[9]が取り入れられている．

GIST で記述されたシステムのふるまいが公理の集合で規定されると考え，その公理から前向き推論によってある帰結 (consequence) を導出する過程が記号実行であると考える．KOKO と呼ばれる記号実行システムは，GIST 仕様の各 statement を，遷移前，後の状態に関する公理に翻訳し，これらの公理を用いて帰結を推論していく．GIST 仕様の構成要素のうち，制約はそのまま公理の形をしている．動作の定義に使用する基本操作や逐次実行，条件分岐，繰り返しといった構成法に対して，動作前の状態で成立していた性質が動作後ではどのように変化するかを規定したスキーマ (predicate transformer) が付加されており，それによって，刺激/応答規則に関する公理が生成される．たとえば，オブジェクト ball2 を関係 Red に挿入する操作 insert Red (ball2) については，実行後の新しい状態は insert した事実を含んでいることを表す

insert Red (ball2)=〉 afterwards Red (ball2)

と，ほかの ball については，Red の関係は保存されることを表す

Red (ball1) before insert Red (ball2)
=〉 afterwards Red (ball1)
～Red (ball1) after insert Red (ball2)
=〉 beforehand ～Red (ball1)

というような規則が生成され，これらを用いて動作前，動作後に成立する性質が推論されていく．

METAFOR[10] や SXL[11] も GIST と同様に relational model を基礎とした実行可能な言語であるが，前者はタイプ (METAFOR では entity class と呼ぶ) 間に is a 階層を導入し，属性の上位タイプからの相続 (inheritance) を可能にしている．

3.3 PAISLey

PAISLey は，JSD[14] と同様にシステムを互いに通信し合う複数のプロセスでモデル化する仕様記述言語である．両者とも詳細は文献 4) に解説されているので，本稿では PAISLey の実行可能性という面からみた特徴について述べる．PAISLey では，各プロセスの状態遷移を successor mapping と呼ばれる状態遷移関数，すなわちプロセスの状態を入力とし，次状態を計算する関数を定義することによってシステムのふるまいを規定する．各プロセスの状態遷移，つまり successor mapping の適用は原則としてほかのプロセスに対して非同期的であるが，ほかのプロセスと通信したり，同期をとったりするために交換関数 (exch-

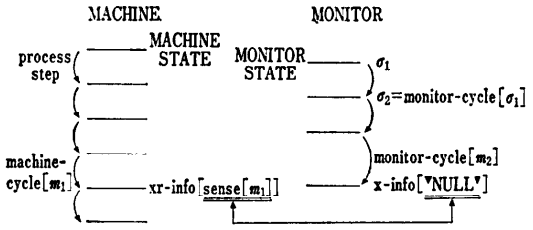


図-3 PAISLey におけるプロセス間の同期

ange function) と呼ばれる擬関数が用意されている．交換関数は，通信し合うプロセスの successor mapping の定義中に対になって出現する．交換関数の評価値は，相手の交換関数の引数の評価値である．図-3 は，machine プロセスと monitor プロセスが交換関数を用いて通信する様子を示したものである．各プロセスは，machine-cycle, monitor-cycle という successor mapping によって状態を遷移させていき，ある状態遷移において次状態を計算するために交換関数を評価する．交換関数の対は x-info と xr-info であり，このとき x-info の評価は xr-info の評価まで待たされる．x-info ['NULL'] の評価値は xr-info の引数 sence [ml] の評価値であり，xr-info [sense [ml]] は NULL となる．

時間的な制約は各関数の評価時間，つまり状態遷移（遷移中の部分過程も含む）に要する時間の最大値，最小値，平均値のいずれかを記述する．インタプリタは実行を始める前に，指定された時間的制約の無矛盾性のチェックや，ある関数の時間的制約がほかの関数に時間的制約を与えているかどうか (timing property inheritance) を調べる．

PAISLey の各プロセスは独立して並列に動作する (asynchronous parallelism) だけでなく，1 つのプロセス内でも可能なかぎり計算が並列に行われる (synchronous parallelism)．PAISLey では，通常の関数型プログラム言語と同様に引数を評価してから関数本体の評価を行うが，たとえば，式

sq-rt [r_sum [(square [a], square [b])]]

においては，r_sum の引数の対 square [a], square [b] は並列に評価され，両者の評価が共に終了した後に関数本体 r_sum の評価が行われる．PAISLey インタプリタは，各関数の評価開始と終了を異なるイベントとみなし，これらのイベントをインタリーブさせて扱うことによって，上記の並列性をシミュレートする．どのようにこれらのイベントをスケジュールするか，つまりどのようにインタリーブするかは，可能なかぎりランダムになるように擬似乱数を用いて，スケジューリングキューにためられたイベントを 1 つずつ順次選択していく手法をとる．スケジューリングは，関数評価に課せられている時間的制約を満たすようにトップダウンに行われる．たとえば，

pipeline-cycle
=put-output [pipeline-stage [get-input]]

の評価については，まず pipeline-cycle の評価開始イベントが選択されたときに，pipeline-cycle に課せられた時間制約を満たすように評価終了イベントを，put-output, pipeline-stage, get-input の時間制約を満たすように pipeline-cycle の評価に必要なイベントをスケジュールしていく．

ユーザは，インタプリタと対話的に仕様の実行を制御，たとえば中断，継続させたり，特定のスケジューリングを行ったりすることができる．関数が未定義だった場合，乱数による値や default 値をインタプリタが自動的に設定したり，ユーザとの対話によりユーザ指定の値を設定したりすることによって実行を続けることも可能である．このほかに，シミュレート時刻や関数評価に要する時刻を設定し直すコマンドも用意されている．

PAISLey は，海底光ケーブル通信システムの開発に適用され，Technology Transfer の問題も含めて大規模システムへの適用実験が始められた[37]．このようにプロセス間通信をベースとするような計算モデルの大規模システムへの適用手法（抽象化手法など）に関するノウハウが蓄積されていくにつれて，本格的な実用システムとなっていくであろう．

3.4 時間論理による手法

システムの許される状態遷移を，様相論理の一種である時間論理式を用いて規定する手法である．時間論理式で記述された仕様を実行するには，1)時間論理の証明手法を用いて仕様を充足するモデルを構築する．2)時間論理式にその解釈と矛盾しない操作的意味を与え，それに従って実行する手法が考えられる．

前者の例は，Wolper らの手法[15]や MENDEL[16]であり，証明法の 1 つであるタブロー法によって時間論理式を充足するモデルを構築し，制御構造の骨組みを得る．これより前者は CSP プログラム，後者は Prolog プログラムを得る．しかし，この手法は論理式を満たすモデルすべてを生成するため，その中から適切なものを選ぶ必要がある．2)の手法は，いわばこ

のような選択を論理式の構文から自動的に行うと考えてよい．たとえば，A→◇Bの論理式をプロダクションルールと同様に解釈し，→ の前件Aは必ず到達される可能性がある状態を表し，その状態に達すれば後件の状態Bに遷移するという意味に限定する．このような手法として，TELL/NSL の動的記述[17]と，時間論理より記述力の高い時区間論理を用いた手法[18]がある．

4. Functional Approach

Functional approach は，operational approach と異なり，システムの入力と出力の関係にのみ注目する手法である．Operational approach は，システムの動的なふるまいを記述しているという点で比較的実行に有利であったが，functional approach ではシステムのふるまいは全く隠されているため，記述言語にその宣言的意味以外になんらかの計算メカニズムを示す操作的な意味が追加されているのが普通である．代表的な手法として代数的仕様化技法，方程式（等式）*に基づく手法，データ構成法に基づく手法，論理式に基づく手法などがある．本章では，これらの手法の中から，OBJ[19),21)]，MODEL[26),27)]，DECARTES[28)]，TELL[29)]の例をあげて述べる．

4.1 OBJ

OBJ は，SRI International の Goguen らによって開発されている抽象データ型の仕様記述言語で，その意味（宣言的）的基礎を多ソート代数と等式論理においている．同様な代数的仕様記述言語は種々開発されてきたが，実行系までを支援しているシステムとしてASL[22)]，HOPE (NPL)[23)]などがある．

OBJ 言語による抽象データ型 stack の仕様記述例を図-4 に示す．OBJ 仕様は，「オブジェクト」（図では OBJ...JBO で囲まれた部分）と呼ばれるモジュールからなる．「オブジェクト」は，モジュール名を宣言するヘッダ，モジュールで定義するソートとオペレーションを宣言するシグニチャ，オペレーション間の関係を表す等式の集合であるボディからなる．OBJ では，例外事象に対するオペレーションをほかのオペレーションと分けて記述する．このようにして記述された仕様は，始代数モデルによってその宣言的な意味が与えられる．

OBJ インタプリタでは，ボディ部に記述されている等式を項書き換え規則として解釈し，実行する．すなわち，等式の左辺のパターンとマッチングする項の出現を右辺で置き換え，次々と項を変形し (reduction)，どの書き換え規則も適用不可能になったときに得られた項 (normal form) を実行結果とする．たとえば，stack-of-int の例では，項

TOP POP PUSH (2, PUSH (1, BOTTOM))

は，OK-EQNS の第一式，第二式を適用することによって

=〉 TOP PUSH (1, BOTTOM)=〉 1

となり，1が得られる．最初の第一式の適用時には，変数 I は2，S は PUSH (1, BOTTOM) が代入される．このような計算メカニズムで重要なことは，1) 停止性と 2) 一意性である．前者については，たとえば交換法則を表す等式 $X+Y=Y+X$ が仕様中に出現していた場合，この規則が無限回適用され，書き換えが停止しないことがある．一般に，与えられた仕様がこのような項書き換えシステムのうえで停止するかどうかを判定するアルゴリズムは存在しないが，有礎集合法 (well-founded set method)[36)] のように，停止することを証明するための手法についてはいくつかの研究が成されている．一意性は，どんな書き換えの戦略をとっても最終結果が一致するという性質である．たとえば，

```
OBJ STACK / BOOL
SORTS STACK ELM
OK-OPS
    PUSH : ELM STACK -> STACK
    POP_ : STACK -> STACK
    TOP_ : STACK -> ELM
    BOTTOM : -> STACK
    EMPTY?_ : STACK -> BOOL
ERR-OPS
    UNDERFLOW : -> STACK
    TOPL : -> ELM
VARS
    I : ELM
    S : STACK
OK-EQNS
    (POP PUSH(I,S) = S)
    (TOP PUSH(I,S) = I)
    (EMPTY? BOTTOM = T)
    (EMPTY? PUSH(I,S) = F)
ERR-EQNS
    (TOP BOTTOM = TOPL)
    (POP BOTTOM = UNDERFLOW)
JBO

IM (STACK => STACK-OF-INT)/ INT
SORTS (STACK => STACK-OF-INT)
       (ELM => INT)
MI
```

図-4 OBJ による仕様記述例[19)]

* 一代数的仕様化技法の等式と区別するために方程式と呼ぶことにしたー

POP PUSH (2, POP PUSH (1, BOTTOM))

は，第一式を適用できる箇所が2箇所あるが，どちらから先に書き換えを行ってもともに最終結果 BOTTOM を得る．項書き換えシステムが Church-Rosser の性質と呼ばれる合流性の条件を満足すれば，そのシステムは停止する書き換えにおいて一意性が保証される．合流性を満たさないシステムにおいても，等式の集合にその等式から導かれる新しい等式を追加したり，ほかの等式から導かれる等式を取り除いたりすることによって，合流性を満たすようにすることができる場合がある．本稿ではこれ以上ふれないが，興味のある読者は文献 24) を参照していただきたい．この仕様の宣言的な意味は，等式を公理とする代数モデルによって与えられるが，これは項書き換え系が与える操作的意味と一致しない．つまり，代数モデルの上で証明される＝の関係は，もとの項と書き換え操作の結果得られる項との関係とは一致していない．たとえば，代数の公理としては，

POP PUSH (I, S)＝S (1)

と

S＝POP PUSH (I, S) (2)

も同じ意味である．左辺の出現を右辺で置き換えるという方向性をもつ項書き換え系では，右辺は関係なく，左辺とマッチングする項のみしか適用できない．たとえば，(2) 式のみでは，POP PUSH (1, BOTTOM) は，POP PUSH (I, POP PUSH (1, BOTTOM)) としか書き換えが行われず，BOTTOM という項とは無関係になる．このように両者の意味は完全には一致していないが，書き換えの結果得られた項ともとの項とが代数的に等しい，つまり代数モデルのうえで合同関係にある（項書き換え系の代数に対する健全性）ということは保証されている[19]．

阪大の ASL も同様な言語であるが，共通な項の重複評価を避けたり，項の評価に必須な引き数の評価を優先したりするなどのソースレベルでの最適化を行うコンパイラが開発されている[22]．また，限定自然言語による仕様を ASL 仕様に変換し，実行させる研究も進められている[25]．

OBJ をはじめとして，このような代数的抽象データ型言語では，数学的にかつ詳細部にいたるまで厳密に記述しなければならないため，仕様作成の段階で大きな労力を要すると思われる．OBJ 2 にもみられるように仕様の部品化と再利用の研究が実用化への大きな足掛りとなるであろう．

4.2 MODEL

4.2.1 MODEL 言語

MODEL 言語は，Pennsylvania 大学の Prywes らによって開発が進められている仕様記述言語である．MODEL によって記述された仕様は，データ記述 (data description) とアサーション (assertion) と呼ばれる方程式の集合からなる．MODEL による仕様記述例を図-5 に示す．この例は，売上個数が記録された図-6 のような売上ファイル (IN) から売上額のレポート (OUT) を作成するプログラムの例である．

MODEL 仕様で用いられるデータ構造は，COBOL や PL/I の構造体と同様に木構造をしている．各ノードには，対応する構造の名前以外にその繰り返し回数を指定することができる．繰り返しが指定されたときは，その構造体を要素とするベクトルとみなすことができ，結果として全体のデータ構造を配列とみなすことができる．たとえば，IN ファイルの ITEM フィールドは2次元の配列とみなすことができる．アサーションは，データ構造中の要素に対して成立する等値関係を記述する．アサーション a_1, a_2 は，各データ構造のパラメータを定義するための式である．同じ項目番号 (ITEM) 同士の売上レコード (INREC) をまとめて1つの INGREP としている（図-6 参照）ため，隣接する INREC 中の ITEM 値が異なった箇所が INREC の最後の要素，つまり1つの INGREP の区切りを表している (a_2)．項目ごとの価格を格納しているファイル ITEMS はインデックスシーケンシャルファイルで，POINTER.ITEMREC は各 ITEMREC を検索する際のキー値からなるベクトルを表す．これは，a_1 により各 INGREP がもっている ITEM 値がセットされる．これによって，IN ファイルの ITEM 値に従って ITEMS ファイル中の ITEMREC が検索され，PRICE を読み出すことができる．a_4 は，1つの INGREP に含まれる INREC 中の QUANT の合計を TOTAL とすることを表している．したがって，a_4 には引数が省略されており，これを補うと，

TOTAL (SUB2)
＝SUM (QUANT (SUB2, SUB1), SUB1)

となる．このような引数の補充は，MODEL 言語のコンパイラが行う．

4.2.2 MODEL 仕様の実行

MODEL 言語の最大の特徴は，前節の OBJ の項書き換えシステムとは異なり，アサーションに対して特に操作的な意味を課していないことである．MODEL

```
/*  HEADER  */
MODULE:  SALES;
SOURCE:  IN, ITEMS;
TARGET:  OUT;

/*  DATA DESCRIPTION OF SOURCE AND TARGET FILES  */
1 IN IS FILE,
   2 INGREP (*) IS GROUP,
      3 INREC (*) IS RECORD,
         4 ITEM IS FIELD (PIC '(6)9')
         4 QUANT IS FIELD (PIC '(6)9');
1 ITEMS IS FILE KEY IS ITEM ORIG IS ISAM,
   2 ITEMREC IS RECORD,
      3 ITEM IS FIELD (PIC '(6)'9),
      3 PRICE IS FIELD (DEC (6,2));
1 OUT IS FILE,
   2 OUTREC (*) IS RECORD,
      3 ITEM IS FIELD (PIC '(6)'9),
      3 TOTAL IS FIELD (PIC 'B(6)9.V99'),
      3 COST IS FIELD (PIC 'B(6)9.V99');

/*  EQUATIONS DEFINING DATA STRUCTURE PARAMETERS  */
/* a1 */  POINTER.ITEMREC = IF END.INREC(SUB1) THEN IN.ITEM(SUB1);
/* a2 */  END.INREC = (IN.ITEM ^= NEXT.IN.ITEM);

/*  EQUATIONS DEFINING TARGET VARIABLES  */
/* a3 */  OUT.ITEM = ITEMS.ITEM;
/* a4 */  TOTAL = SUM(QUANT(SUB1), SUB1);
/* a5 */  COST = PRICE*TOTAL;
```

図-5 MODEL による仕様記述例[26]

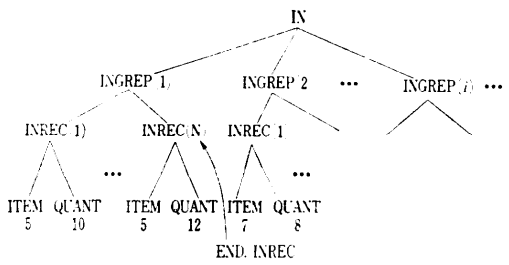


図-6 ファイル IN のデータ構造

のコンパイラは，データ記述とアサーションより，データ間の依存関係を仮定し，それを配列グラフ (array graph) と呼ばれる有向グラフで表現する．依存関係は，直観的には，データ値を求めるにはどのデータ値が必要かという関係である．コンパイラは，その配列グラフよりスケジュールと呼ばれるデータ値算出の手順の骨組みを求め，これより実際の COBOL もしくは PL/I プログラムを得る．もし，適切な依存関係が仮定できなければ，アサーションを連立代数方程式と解釈する．

図-5 の MODEL 仕様の配列グラフを**図-7** に示す．配列グラフの各ノードには，アサーション名か配列名が対応している．配列要素1つが1つのノードに対応しているのではなく，配列全体が対応している．枝に

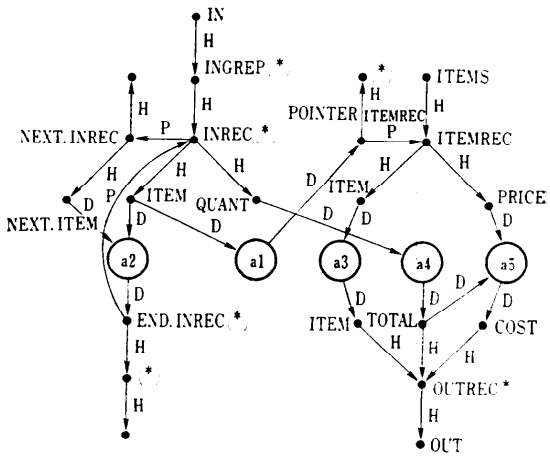


図-7 配列グラフの例[26]

付加されているラベルは，依存関係の種類を表す．たとえば，Dは方程式の左辺に出現する変数と右辺に出現する変数とを示している．この関係は，直観的には，左辺の変数の値を求めるには，右辺の変数の値を先に求めておくことが必要であるという仮定を表している．配列グラフは，曖昧な変数の解消や省略箇所の補充，次元の異なる配列の等値関係といった矛盾の発見にも使用される．

スケジュールを得るには，配列グラフのノードをトポロジカルソートすればよいように思われるが，グラフ中にループが含まれているときが問題となる．ループ上のノードに割り当てられた変数をVとすると，Vの値を計算するにはVの値自身も必要であることになり，ループ上のどの変数から計算を始めればよいかが判定できない．しかし，配列グラフに付けられている変数は1つの配列要素ではなく，配列全体を指している．したがって，$V(I)$を計算するのに$V(I-1)$の値しか必要ない場合でも，Vがループの中に入ってしまう．MODEL コンパイラは，まずグラフ中に含まれるすべての極大強連結部分グラフ (Maximally Strongly Connected Component：以下 MSCC と略す) を抽出する．MSCC とは，MSCC 中に含まれるどんな2つのノードをとってもそれらをつなぐ有向路が MSCC 中に存在し（強連結性），かつほかのノードを加えるとこの強連結性が失われるような部分グラフのことである．抽出した MSCC をおのおの1つのノードとみなしてトポロジカルソートを行う．その後，MSCC のノードについて添え字式までも含めた解析を行い，ループが存在しなくなるまでループを構成する枝を切断していく．アサーションが配列全体を表す変数を含むときは，その配列の次元と大きさを基に繰り返しブロックを生成する．この操作は，MSCC の分解と同時に行う．また，繰り返し回数が同じブロックを1つにまとめるなどの最適化も行う．

ループを解消できない場合は，そのループに含まれるアサーションの集合を連立代数方程式とみなし，Gauss-Seidel 法のような方程式を解く手法をスケジュールの中に埋め込む．たとえば，2つのアサーション $X=aY+b$, $Y=cX+d$ は，単純変数 X, Y のノードとともにループを構成し，解消できない．この場合には，この線形2元連立方程式を解く手法が組み込まれる．図-8 に図-5 のスケジュールを示す．

MODEL のアサーションは，代数方程式を基礎としているため，単純なデータ操作のみを扱うシステムにその適用範囲が限定されると思われる．分散型協調システムにおいて，分割された局所的なモジュールの記述に MODEL 言語を用いた例が報告されているが[27]，この例のようにほかのモデル化手法と組み合わせることにより，その利点が発揮できると思われる．

4.3 DESCARTES

Descartes 言語は，Hoare のデータ構成法に基づく仕様記述言語である．Descartes 仕様は，入力データと出力データ構造を直積，直和，列を用いて定義し，それらの各要素をパターンマッチによって対応付ける

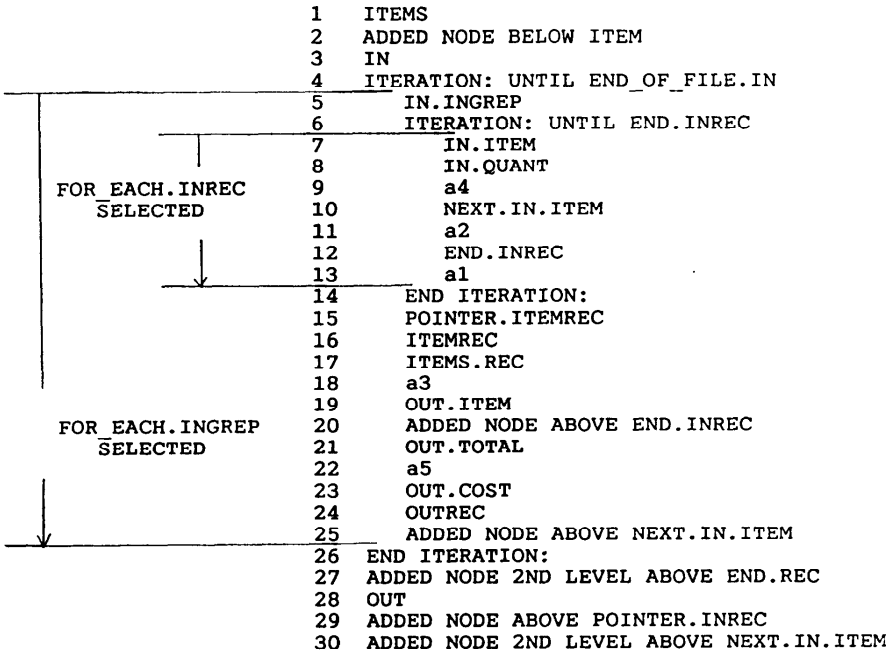


図-8 スケジュールの例[40]

```
(STACK)_PUSHED_BY_(ELEMENT)
    ELEMENT
        element_to_push
                ELEMENT_TYPE
    STACK
        elements*
                ELEMENT_TYPE
    return
        new_stack
                ELEMENT_TO_PUSH
                ELEMENTS
```

図-9 Descartes による仕様記述例[20]

ことによって入出力関係を規定する．実行は，データを記述に従って分解 (Analysis)，合成 (Synthesis) を行い，出力データを生成することにより行われる．図-9 に stack の push 演算の仕様記述例を示す．

モジュールは，合成木 (Synthesis tree) と分解木 (Analysis tree) の2種類の木の集合として定義される．各木は，Hoare のデータ構成法に基づいて記述される．木のノードには，マッチノード（英小文字で表される）と参照ノードの2種類があり，前者はマッチングにより値を得ることができるノード，後者は同じ名前をもったマッチノードより値をもらったり，マッチングを制御したりする役割をもつ．マッチング過程において，マッチノードは変数の役割を，参照ノードは定数と同様な役割を果たすと考えてよい．合成木はその根がマッチノード，分解木は根が参照ノードとなっているような木である．モジュールの値，つまり出力値は合成木の根である return ノードにマッチングされた値である．図-9 中で，分解木 STACK 中のノード elements* は ELEMENT_TYPE 型の要素の列からなることを表し，合成木 return 中のノード new-stack は ELEMENT_TO_PUSH と ELEMENTS の直積で構成されることを表す．列は，直積を任意回繰り返して構成されると考える．

図-9 の push 演算の実行過程を，整数型の stack を用いて説明する．したがって ELEMENT_TYPE は INTEGER と考える．入力引数 STACK には (294, 3112) が，ELEMENT には 84 が与えられたとする．84 はマッチノード element_to_push にマッチし（ただしその子ノードである参照ノード ELEMENT_TYPE が 84 のタイプと一致していたときに限る），合成木 return の参照ノード ELEMENT_TO_PUSH の値が 84 となる．同様にして ELEMENTS の値も (294, 3112) となり，それらの直積 (84, 294, 3112) が結果値となる（図-10 参照）．

Decartes は，データ構造化法にその基礎を置いているため，データベースシステムなどのデータ構造が明確で，かつそれが本質的であるようなシステムに効力を発揮すると思われる．

4.4 TELL

TELL システムの仕様記述言語 NSL は，限定された自然言語（英語）を基礎としている．NSL の詳細化手法は，記述中に出現する問題固有の単語がモジュールに対応するとみなし，これらの単語の意味を限定自然言語で次々に定義していくという手法を用いている．システムを静的つまり functional に，あるいは動的 operational に捉えているかは，使用した単語の構文カテゴリに反映する．入出力関係のみに注目しているときは，名詞と形容詞のみが使用される．NSL では，両者のアプローチに対して，単語の構文カテゴ

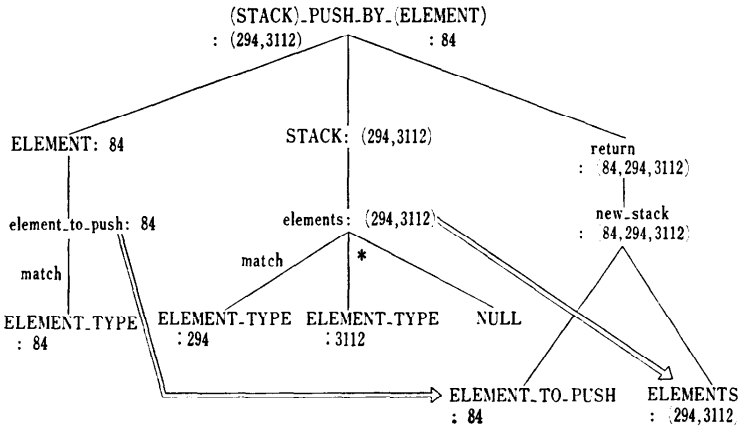


図-10 Descartes 仕様の実行メカニズム

TELL/NSLによる仕様

```
Pages T is pagenated from file F between minimum line number Min
  and maximum line number Max means that
   case 1) F is empty: T is empty.
   case 2) F is not empty:
        2-1) The result of separating a page from F is
               page P and file F1.
        2-2) Pages T1 is pagenated from F1 between Min and Max.
        2-3) T is the concatenation of P and T1.
end pagenated;
```

⇩

論理式

```
∀T ∀F ∀Min ∀Max[pagenated(T,F,Min,Max)↔
    pages(T)∧file(F)∧line_number(Min)∧line_number(Max)∧
    ((F=empty∧T=empty)∨
     (¬F=empty∧∃F1[file(F1)∧∃P[page(P)∧∃T1[pages(T1)∧
           separate_a_page(P,F1,F,Min,Max)∧
           pagenated(T1,F1,Min,Max)∧T=concatenation(P,T1)]]]))]
```

⇩

Prologプログラム

```
pagenated([],[],Min,Max) :-
        line_number(Min),line_number(Max).

pagenated([V02|V03],F,Min,Max) :-
        file(F),line_number(Min),line_number(Max),
        not(F=[]),
        separate_a_page(V02,V01,F,Min,Max),
        pagenated(V03,V01,Min,Max).
```

図-11 TELL/NSL による仕様記述例と翻訳[29]

リに対応させて異なる記述形態を与えている．本節では，そのうちの静的記述と呼ばれるシステムの入出力関係のみを記述する方式について簡単にふれる．NSLの静的記述は，1階述語論理式に変換され，厳密な意味（宣言的）が割り当てられる．この論理式を Horn Clause に変換し，Prolog プログラムへと変換する．その例を図-11 に示す．Prolog プログラムの input resolution と閉世界仮説が与える操作的な意味と，NSL の宣言的な意味とは完全に等価ではないが，宣言的意味に対する健全性は保証されている．このように限定自然言語で記述された静的な仕様をプログラム言語に変換し，実行させようとする研究に阪大のシステム[30]，電力中研の ARIES/I[31]，Weischedel らのシステム[32]がある．前者2つは，ともに対象領域固有の知識を変換時に使用するものである．自然言語の導入は，仕様作成におけるユーザインタフェースの向上に有効であるが，どのように自然言語を機械処理可能なように限定するかは，今後の課題であろう．また，Prolog に抽象データ型の概念を導入し，図的表現も使用できるように拡張された仕様記述言語として，Rueher らの言語[33]がある．

5. おわりに

本稿では，現在までに開発が進められている実行可能な仕様記述言語とそのシステムの代表例を述べた．仕様を直接実行させ，その実行結果に基づき，要求を確認（validate）するという手法は，従来のウォータフォール型のソフトウェア開発の欠点を補うために生まれてきた手法であり，現在も活発に研究が進められている．しかし，現実世界の対象物をいかにモデル化するかという点では，実行可能な仕様を構築する過程も従来の形式的仕様を構築する過程も，その難しさは変わらない．むしろ，実行可能性に注意が払われ，プログラミング段階でなされるべき配慮（たとえば，効率的なアルゴリズムの導入など）が入り込んでしまい，仕様の伝達性が低下してしまう危険性もある．仕

様をいかにして構築するかの手法の研究は，実行可能性を問わず重要な問題である．また，得られた仕様をいかに後段の過程に役立てるかも問題である．実行可能な仕様を確認のためだけでなく，Balzer らが述べているように変換によって実際のプログラムを得る[35]などの，仕様（の一部）をなんらかの形で後段の生成物に流用するための研究も重要な課題である．

なお，本文中で説明に用いた例は，**3.2.2** では文献9），**3.3** は12），13）より引用した．また，**4.1** と **4.3** の実行データ値の例はおのおの19）と28）より引用した．

謝辞 本稿の作成でご苦労をいただいた卯木淑子さんに感謝いたします．

参考文献

1) 有澤 誠：ソフトウェアプロトタイピング，近代科学社（1986）．
2) 松本吉弘：ソフトウェア工学演習，朝倉書店（1984）．
3) Zave, P.: The Operational versus the Conventional Approach to Software Development, Comm. ACM, Vol. 27, No. 2, pp. 104-118 (1984).
4) 野木兼六：要求定義技術の最近の動向，情報処理，Vol. 27, No. 1, pp. 21-30 (1986).
5) Davis, A. M.: Rapid Prototyping using Executable Requirements Specifications, ACM SIGSOFT, Vol. 7, No. 5, pp. 39-44 (1982).
6) Dasarathy, B.: Timing Constraints of Real-Time Systems: Constructs for Expressing Them, Methods of Validating Them, IEEE Trans. Soft. Eng., Vol. 11, No. 1, pp. 80-86 (1985).
7) Balzer, R., Goldman, N. M. and Wile, D. S.: Operational Specification as the Basis for Rapid Prototyping, ACM SIGSOFT, Vol. 7, No. 5, pp. 4-16 (1982).
8) Swartout, B.: GIST English Generator, Proc. of AAAI 82, pp. 404-409 (1982).
9) Cohen, D.: Symbolic Execution of the Gist Specification Language, Proc. of 8th IJCAI, pp. 17-20 (1983).
10) McGowan, C. L., Feblowitz, M. D. and Chandrasekharan: The METAFOR Approach to Executable Specifications, Proc. of 3rd International Workshop on Software Specification and Design, pp. 163-169 (1985).
11) Lee, S. and Sluizer, S.: SXL: An Executable Specification Language, Proc. of 4th International Workshop on Software Specification and Design, pp. 231-235 (1987).
12) Zave, P.: An Overview of the PAISLey Project-1984, ACM SIGSOFT, Vol. 9, No. 4, pp. 12-19 (1984).
13) Zave, P. and Schell, W.: Salient Features of an Executable Specification Language and Its Environment, IEEE Trans. Soft. Eng., Vol. 12, No. 2, pp. 312-325 (1986).
14) Cameron, J. R.: An Overview of JSD, IEEE Trans. Soft. Eng., Vol. 12, No. 2, pp. 222-240 (1986).
15) Manna, Z. and Wolper, P.: Synthesis of Communicating Processes from Temporal Logic Specifications, ACM Trans. Prog. Lang. Syst., Vol. 6, No. 1, pp. 68-93 (1984).
16) 本位田真一，内平直志，大須賀昭彦，粕谷利明：推論型システム記述言語 MENDEL，情報処理学会論文誌，Vol. 27, No. 2, pp. 219-227 (1986).
17) 米崎直樹，佐伯元司，市川 至，蓬萊尚幸，土井秀俊：TELL/NSL における動的記述の Prolog への変換，第30回情報処理学会全国大会講演集，pp. 495-496 (1985).
18) Diaz-Gonzalez, J. P. and Urban, J. E.: ENVISAGER: A Visual Object-Oriented Specification Environment for Real-Time Systems, Proc. of 4th International Workshop on Software Specification and Design, pp. 13-20 (1987).
19) Goguen, J. and Meseguer, J.: Rapid Prototyping in the OBJ Executable Specification Language, ACM SIGSOFT, Vol. 7, No. 5, pp. 75-84 (1982).
20) Goguen, J. A.: How to Prove Algebraic Inductive Hypotheses without Induction: with Applications to the Correctness of Data Type Representations, LNCS, Vol. 87, pp. 356-373 (1980).
21) Futatsugi, K., Goguen, J., Meseguer, J. and Okada, K.: Parameterized Programming in OBJ 2, Proc. of 9th ICSE, pp. 51-60 (1987).
22) 井上克郎，関 浩之，谷口健一，嵩 忠雄：関数型言語 ASL/F とそのコンパイラ，電子通信学会論文誌，Vol. J 67-D, No. 4, pp. 458-465 (1984).
23) Feather, M. S.: Program Specification Applied to a Text Formatter, IEEE Trans. Soft. Eng., Vol. 8, No. 5, pp. 490-498 (1982).
24) 二木厚吉，外山芳人：項書き換え型計算モデルとその応用，情報処理，Vol. 24, No. 2, pp. 133-146 (1983).
25) 関 浩之，並河英二，藤井 譲，嵩 忠雄：自然語によるプログラム仕様の形式的意味定義—自然語による仕様から代数的仕様への変換—，情報処理学会ソフトウェア工学研究会 52-7, pp. 49-56 (1987).
26) Prywes, N. S. and Pnueli, A.: Compilation of Nonprocedural Specifications into Computer

Programs, IEEE Trans. Soft. Eng., Vol. 9, No. 3, pp. 267-279 (1983).
27) Prywes, N. S. and Pnueli, A.: Automatic Program Generation in Distributed Cooperative Computation, IEEE Trans. Syst. Man, Cyber., Vol. 14, No. 2, pp. 275-286 (1984).
28) Urban, J. E.: Software Development with Executable Functional Specifications, Proc. of 6th ICSE, pp. 418-419 (1982).
29) 市川 至, 蓬萊尚幸, 佐伯元司, 米崎直樹, 榎本肇: 自然言語に基づく静的システムの仕様のプロトタイププログラムへの変換手法, 情報処理学会論文誌, Vol. 27, No. 11, pp. 1112-1128 (1986).
30) 上原邦昭, 藤井邦和, 豊田順一: 自然言語による仕様からのプログラム自動合成, コンピュータソフトウェア, Vol. 3, No. 4, pp. 55-64 (1986).
31) 原田 実, 篠原靖志: 部品合成によるプログラム自動合成システム ARIES/I, 情報処理学会論文誌, Vol. 27, No. 4, pp. 56-62 (1986).
32) Weischedel, R. M.: Mapping between Semantic Representations using Horn Clauses, Proc. of AAAI 83, pp. 424-428 (1983).
33) Rueher, M.: From Specification to Design: An Approach Based on Rapid Prototyping, Proc. of 4 th International Workshop on Software Specification and Design, pp. 126-133 (1987).
34) Banatre, J. P. and Le Metayer, D.: A New Approach to Systematic Program Derivation, Proc. of 4 th International Workshop on Software Specification and Design, pp. 208-215 (1987).
35) Balzer, R., Cheatham, T. E. and Green, C.: Software Technology in 1990's: Using a New Paradigm, Computer, Vol. 16, No. 11, pp. 39-45 (1983).
36) Dershowitz, N. and Manna, Z: Proving Termination with Multiset Orderings, Comm. ACM, Vol. 22, No. 6, pp. 465-475 (1979).
37) Berliner, E. F. and Zave, P.: An Experiment in Technology Transfer: PAISley Specification of Requirements for an Undersea Lightwave Cable System, Proc. of 9 th ICSE, pp. 42-50 (1987).

(昭和62年6月15日受付)