依存情報を用いたモデル作成過程の表現とその可変ソリッドモデリングへの応用†

乾 正知†† 木村文彦††

機械の設計生産に関する問題解決は，計算機内に設計対象や作業環境のモデルを構築しつつ進めるものが多い．CADによるソリッドモデルの作成や，ロボットの作業計画の作成などはその典型例である．これらの問題解決では，まずモデルを作成しそのモデルを基に評価をおこなう．そして評価結果に応じてモデルを修正したり，モデルを作成し直したりを繰り返す．このような試行錯誤をともなうモデル作成を支援するためには，モデル作成の過程を計算機内に適切に表現し管理する必要がある．本研究ではこの過程を，モデル作成操作相互の依存情報と，操作によって変更されたモデルデータと操作との依存情報とで表現する手法を提案する．この依存情報を基にモデルデータを操作することで，モデルの過去の状態の復元や任意の操作の取消しなどの機能が統一的に実現できる．実際にソリッドモデリングシステムを本手法に基づくデータ管理システム上に実現した．このシステムでは，依存情報を基に試行錯誤の過程において，位相構造と幾何情報が変化するソリッドモデルの作成が可能であり，本手法の有効性が検証された．

1. はじめに

機械の設計生産に関する問題解決は，計算機内に設計対象や作業環境のモデルを構築しつつ進めるものが多い．CADによるソリッドモデルの作成や，ロボットの作業計画の作成などはその典型例である．これらの問題解決では，まずモデルを作成しそのモデルを基に評価をおこなう．そして評価結果に応じてモデルを修正したり，モデルを作成し直したりを繰り返す．このような試行錯誤をともなうモデル作成を支援するためには，モデルの作成過程を計算機内に適切に表現し管理する必要がある．

このようなモデル作成過程支援の重要性はよく理解されており，いくつかの商用CADシステムなどには，誤って実行された操作の取消し機能が用意されている．その実現手法は明らかにされていないが，操作実行前のモデルの状態を記録する，操作コマンドの実行系列を記録する，操作にともなうモデルの変化の差分を記録する，などが用いられていると考えられる．最近では，すべてのモデリング操作を可逆な基本操作だけで実現し，基本操作の実行履歴を管理することで，モデル作成過程を支援する手法も提案されている[1]．

一方人工知能の研究では，モデルの状態変化に関する推論やプラン生成問題の解決を目的に，操作にともなうモデルの変化を定式化し，計算機処理する研究がおこなわれている．たとえば文献2)や3)では，操作にともなうモデルの変化を表す「世界」という概念を導入し，モデル作成過程をこの世界の列として表現し管理する手法が提案されている．

これらの商用システムや研究では，モデル作成過程を，操作の実行順序にしたがって管理することを目的にしている．**図1**の積木の操作を例に考える．この例では，マニピュレータを用いて積木 x をつまみあげテーブル上へおろす操作 putdown (x) と，テーブルから積木 x をつまみあげ積木 y の上に載せる操作 stack (x, y) が定義されている．マニピュレータは，まずすべての積木を putdown 操作を用いてテーブル上に降ろし，次いで stack 操作を用いてそれらを上からA，B，C，Dの順になるように積み上げていく．図1(1)はモデルの初期状態を示しており，積木A，B，C，Dとテーブル (Table) の関係が，述語 on を用いて記述してある (on (x, y) は x が y の上にあることを意味する)．また積木 x の上面になにも載っていないことを，述語 clear (x) を用いて記述してある．

この積木の世界のモデルは，操作 putdown (A), stack (C, D), stack (B, C), stack (A, B) を実行することで，順に図1の(2)，(3)，(4)，(5)のように変化する．これまでの手法では，操作の実行順序に基づいてモデルを管理するので，たとえば(5)のモデルの状態で操作 putdown (A)を取り消すように命じると，モデルを putdown (A)実行前の状態，すなわち

† Representation of Modeling Processes by Data Dependency and Its Application for Variational Solid Modeling by MASATOMO INUI and FUMIHIKO KIMURA (Factory Automation Laboratory, Research Center for Advanced Science and Technology, The University of Tokyo).

†† 東京大学先端科学技術研究センター ファクトリーオートメーション分野

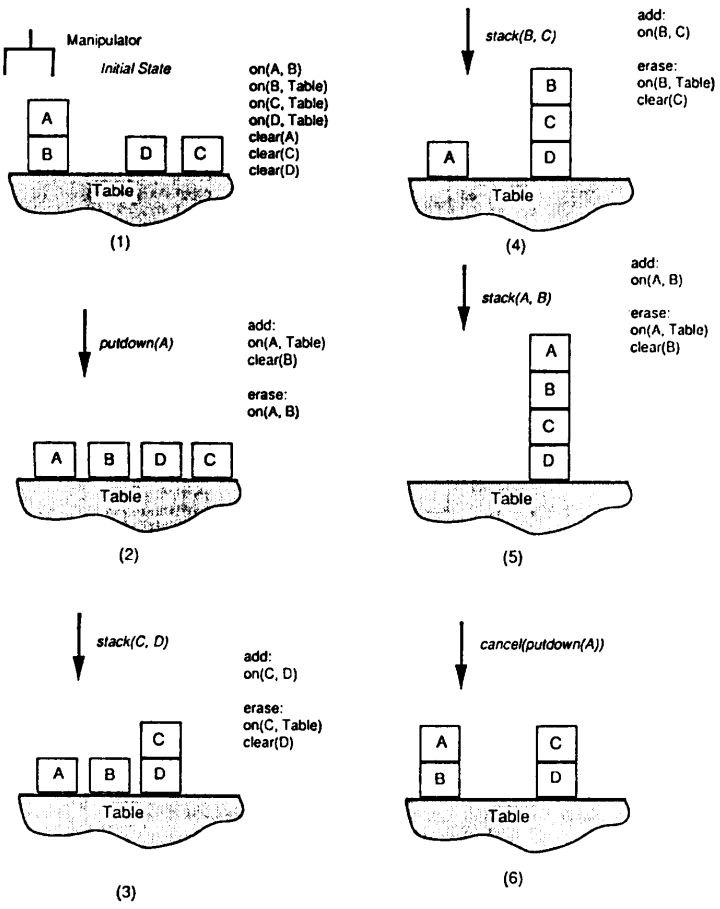


図 1 積木の世界のモデリングの例
Fig. 1 A simple operation process in block world.

初期状態(1)へ変化させる.

この操作の実行順序に基づくモデル管理のほかに,操作の影響に基づくモデル管理も考えられる.たとえば,モデルを図1の(2)から(3)の状態に変化させる操作 stack (C, D) は,putdown (A)によって付加されたモデルデータ on (A, Table) や clear (B)とは無関係に実行されている.したがって操作の影響に基づくモデル管理では,putdown (A) (の影響) を取り消しても stack (C, D) の実行は影響を受けない.その結果,モデルデータは(1)ではなく(6)のように変化する必要がある.たとえばソリッドモデリングでは,数百もの操作を用いてモデルを作成するが,操作の多くは互いに独立していることが多い.したがって試行錯誤をともなうモデル作成を支援するためには,従来の操作順序に基づくモデル作成過程管理だけでなく,操作の影響に基づく管理も必要である.

本研究ではこの問題を解決するために,操作を Fikes らの提案する枠組みを基に定式化する[4].そしてこの定式化に基づいて,モデル作成過程を,操作間の依存情報と操作によって変更されたモデルデータと操作との依存情報とで記述し管理する手法を提案する.依存情報を管理する道具として Assumption-based Truth Maintenance System (ATMS)[5] を利用する.この依存情報を基にモデルデータを操作することで,操作の実行順序に基づいてモデルの過去の状態を復元する機能と,操作の影響を取り消しモデルを再構成する機能が統一的に実現できる.われわれは以前この考えに基づき,操作に関するすべての情報を ATMS の依存情報で扱う手法を考察し文献 6) に報告した.しかし以前の手法では,操作の時間順序を表現する依存情報が操作の実行回数に比例して増加するため,システムの処理速度に問題があった.本手法では操作の時間順序は依存情報では扱わない.したがってかなり実際的な応用でも満足のいく成果が得られた.

本手法の有効性を検証するために,ソリッドモデリングシステムを実現した.このシステムでは,依存情報を基に試行錯誤の過程において,位相構造と幾何情報が変化するソリッドモデルの作成が可能であり,本手法の有効性が検証された.

2. 操作の定式化

2.1 推論規則による操作の定式化

Fikes らは,操作を前向き推論規則として扱うことを考え,操作を(1)その操作が適用可能となる前提規則を表す,存在限定子で量限定される変数を含むリテラルの連言 (前提規則) と,(2)その操作を適用した結果削除されるリテラルのリスト (消去リスト),(3)操作を適用した結果新たに加えられるリテラルのリスト (追加リスト),の3つからなるものとして定式化した[4].ここで(2)と(3)のリテラルにも変数の存在を許すが,それらは必ず(1)の規則適用時に実体化されていなければならない.たとえば前章で用いた操作 putdown (x) や stack (x, y) は,次のように記述できる.

putdown (?x)

前提規則：clear (?x) & on (?x, ?y)
消去リスト：on (?x, ?y)
追加リスト：on (?x, Table), clear (?y)

stack (?x, ?y)
前提規則：on (?x, Table) & clear (?x) & clear (?y)
消去リスト：clear (?y), on (?x, Table)
追加リスト：on (?x, ?y)

ここで & は論理積を表す．また接頭子 ? をともなうシンボルは変数を表す．putdown (?x) の前提規則は，?x の上面になにもなく，かつ ?x が ?y の上に載っていることである．また stack (?x, ?y) の前提規則は，?x がテーブルの上にあり，かつ ?x と ?y の上面になにも載っていないことである．それぞれの消去リスト，追加リストには，操作の実行にしたがってモデル（この場合には積木の配置）を変更するための情報が与えられている．

この推論規則による操作の定義は，一般的なモデリング操作を記述するには厳しすぎる．たとえば**図2**に示す図形どうしの結合操作では，操作の前提条件は互いに干渉するエッジの位相構造と幾何情報と考えられる．しかし図形の結合の仕方は図2(2)に示すように様々な場合が考えられ，それらすべてについて前提条件を規則として記述することは困難である．操作によって新たに付加される頂点の情報や消去されるエッジの情報も，図形の結合の仕方によって変化するので，それらをあらかじめ消去リストや追加リストとして定義することもできない．

2.2 操作の適応結果の記録

われわれの目的は，モデル作成過程を記録し管理することにある．したがって操作を推論規則として扱う必要はなく，操作の前提規則の適用結果と操作のモデルに対する影響だけが記録できればよい．

前提規則の適用結果とは，前提規則の変数がすべて実体化したものにあたるリテラルの連言である．たとえば図1の(1)のモデルの状態で，putdown 操作を積木Aに適用した場合には，

clear(A) & on(A, B)

が前提規則の適用結果となる．この結果の記録だけが重要なので，この結果と同等のリテラルの連言を検出するプログラムが用意できるならば，前提規則を「規則」として記述する必要はない．図2の図形処理の場合では，処理の途中で必ず干渉するエッジの組合せが

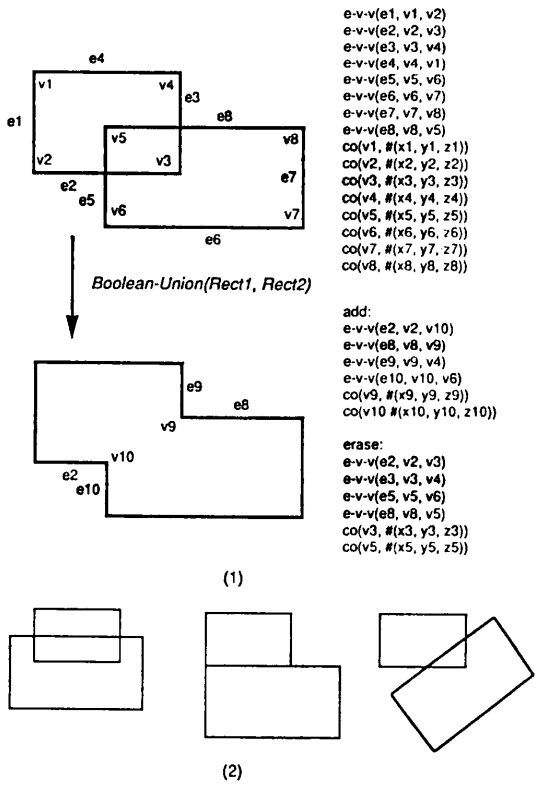


図2 2次元図形の操作の例
Fig. 2 A geometric modeling example which is difficult to formulate in Nilsson's definition.

検出されるので，それらの位相構造や幾何情報にあたるリテラルの連言を，「前提規則の適用結果」として記録するプログラムを書くことは容易である．

操作の影響の記録とは，操作によって付加もしくは削除されたモデルデータの記録であり，これは消去リストや追加リストのリテラルの変数が実体化したものである．前述の putdown(A) の場合には，モデルから削除された on(A, B) と，新たに付加された on (A, Table) および clear(B) が操作の影響の記録となる．これらの記録が重要なので，この結果と同等のリテラルを操作実行の際に検出できるならば，あらかじめ消去リストや追加リストの形で定義する必要はない．操作の実行にともない，変数を含まない幾つかのリテラルがモデルへ付加され，またモデルから削除されるので，それらを記録するプログラムを用意すればよい．

3. ATMS[5]

前章で述べた前提規則の適用結果と操作の影響は，依存情報を利用して記録することができる．本研究では依存情報の管理にATMSを用いた．本章ではATMS

について簡単に述べる．

ATMS では，仮定と呼ばれる何にも依存しないリテラルと，それ以外の推論によって導かれたリテラル(以後推論事実と呼ぶ) の2種類のリテラルを扱う．リテラルが推論によって導かれるとき，その導かれたリテラルと，推論で参照されたリテラルとの依存関係を定義することができる．この依存関係は，各リテラルをノードとし，依存関係をアークとする依存情報グラフの形で記述できる．

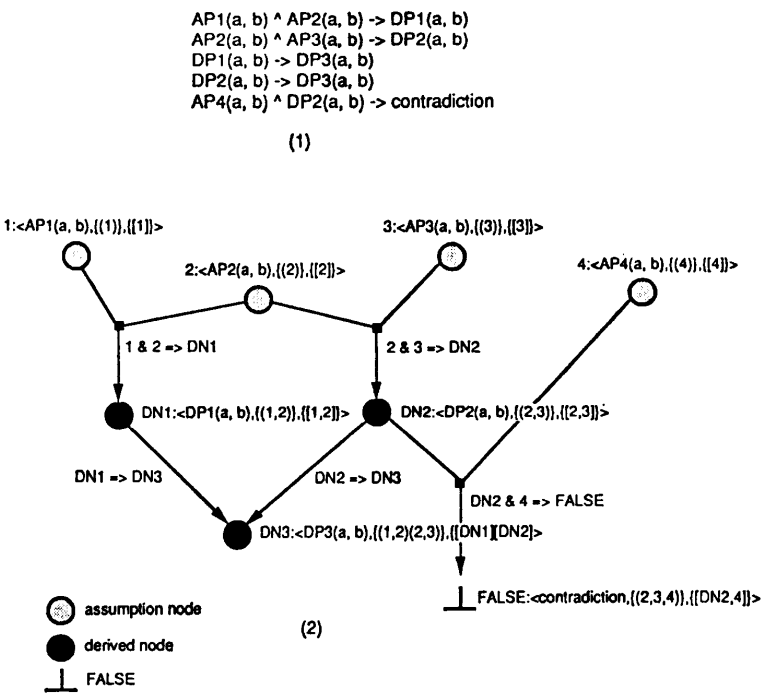


図 3 推論結果(1)とそれに対応する ATMS による依存情報の表現(2)
Fig. 3 Some implications (1) and their corresponding ATMS data dependency diagram (2).

図3には，(1)に示した推論に基づいて決められた依存情報グラフを(2)に示した．図3(1)中，

AP1 (a, b) ^ AP2 (a, b)
-〉 DP1 (a, b)

は，ふたつの仮定 AP1 (a, b) と AP2 (a, b) から推論事実 DP1 (a, b) が導かれることを意味している．このとき DP1 (a, b) は，AP1(a, b) と AP2(a, b) に依存するとみなす．図3(2)には，この依存情報がグラフの一部として示されている．すなわち仮定 AP1 (a, b) と AP2 (a, b) に対応してそれぞれノード1と2が，また推論事実 DP1(a, b) に対応してノード DN1 が定義され，その間の依存関係がグラフのアークの方向と結合で示されている．なおグラフでは，仮定に対応するノード(以後仮定ノードと呼ぶ) は灰色の円で，推論事実に対応するノード（推論事実ノード）は黒色の円で示した．そしてノードの違いをわかりやすくするために仮定ノードは数字で，推論事実ノードは DN1 などのシンボルで名前付けした．アークの矢印は，依存されるノードから依存するノードの向きに付けられている．また複数の矢印なしのアークが結合して一本の矢印付きアークとなるときには，矢印のないアークの根元のノードの論理積に相当するものに，矢印付きアークの先のノードが依存することを示す．したがって依存情報は，ノード名と記号 & および =〉を用いて示すこともできる．たとえば前述の依存情報は，

1 & 2 =〉 DN 1

とも書ける．

依存情報グラフの各ノードには3種類の情報がしまわれる．ひとつは，そのノードに対応するリテラルそのもので，データムと呼ばれる．もうひとつは，そのノードが直接依存するノード群のリストである．これを正当化リストと呼ぶ．最後の情報はラベルと呼ばれ，そのノードが究極的に依存する仮定ノード群のリストである．このラベルについては後で詳しく述べる．これらは図3(2)には，

ノード名：〈データム，ラベル，正当化リスト〉

の形式で記述されている．たとえば前述のノード DN1 では，データムは DP1 (a, b) であり，直接依存するノード群は1と2なので正当化リストは {[1, 2]} である．またラベルは後述する理由から {(1, 2)} となり，

DN1 : 〈DP1(a, b)，{(1, 2)}，{[1, 2]}〉

と記録される．

ATMS では仮定はどのリテラルにも依存しない．これを「仮定は自分自身に依存する」とみなし，仮定に対応するノードには，その正当化リストとラベルにその仮定ノード自身だけを与えることにする．たとえば仮定 AP1 (a, b) に対応するノード1は，

1 : 〈AP1 (a, b)，{(1)}，{[1]}〉

とする．

仮定ノードが自分自身に依存することから，すべての推論事実ノードは，依存情報グラフのアークを矢印

とは逆向きに辿ることで，究極的に依存する仮定ノードの集合を得ることができる．これを環境と呼ぶ．一般にリテラルは複数の仮定の集合から導かれうるので，推論事実ノードは複数の環境を持つ．この環境の集合がラベルである．ラベルは基本的には次のように決められる．J_{ik} がノード DNn の正当化リストの k 番目のノード群中 i 番目のノードのラベルのとき，DNn のノードのラベルには，

$$\bigcup_k \{x \mid x = \bigcup_i x_i \text{ where } x_i \in J_{ik}\}$$

によって得られる環境のリストが与えられる．(その完全なアルゴリズムは文献 5) 参照.) 前述の DN 1 の場合には，その正当化リストはただ1つのリスト [1, 2] からなる．1と2はそれぞれ仮定ノードであり，そのラベルは {(1)} と {(2)} である．それぞれの環境(1)と(2)の集合和として得られる環境は (1, 2) なので，DN 1 のラベルは {(1, 2)} となる．図3(2)のノード DN 3 の場合には，その正当化リストは2つの要素 [DN 1] と [DN 2] からなる．そのそれぞれから環境 (1, 2) と (2, 3) を得るので，DN 3 のラベルには {(1, 2) (2, 3)} が与えられる．

推論の結果，矛盾が導かれることがある．また使用者がある目的から，意図的に矛盾を導かせることもある．図3(1)では仮定 AP 4 (a, b) と推論事実 DP 2 (a, b) から矛盾 (contradiction) が導かれている．(2)ではこの依存情報は，DP 2 (a, b) に対応するノード DN 2 と AP 4 (a, b) に対応するノード4を用いて，

DN 2 & 4 => FALSE

と記述される．このとき矛盾に対応するノード FALSE のラベルは前述のアルゴリズムによって決定され，{(2, 3, 4)} が与えられる．前述の環境の意味付けにしたがえば，この矛盾ノードのラベルを構成する環境 (2, 3, 4) は，矛盾を引き起こした究極的な原因となる仮定ノードの組合せと考えられる．ATMS はこの矛盾の原因となる環境を nogood と呼び特別に管理する．

ATMS では，コンテクストは nogood の組合せを含まない仮定ノードの集合として定義される．各ノードはそのノードを導いた原因となる環境をラベルに持つので，ラベル中の環境が1つでもコンテクストの部分集合ならば，そのノードとノードに対応するリテラルは与えられたコンテクストにおいて「正しい」と定義する．図3の例では，コンテクストを (2, 3) と定めると，仮定ノード 2, 3 と推論事実ノード DN 2, DN 3 は，このコンテクストの部分集合となる環境をラベルに持つので，これらのノードと対応するリテラル AP 2 (a, b), AP 3 (a, b), DP 2 (a, b), DP 3 (a, b) は，このコンテクストにおいて正しいことになる．しかし仮定ノード 1, 4 や推論事実ノード DN 1, 矛盾ノード FALSE などの環境は，このコンテクストには含まれないので正しくなく，また対応するリテラルも正しくないと判断される．

4. ATMS による操作の表現

本章では，2章で述べた前提規則の適用結果と操作の影響を依存情報のグラフで表現し，それを ATMS で管理する手法を述べる．

4.1 モデリング操作

図 4 に，図1に示した積木の操作を ATMS を用いて依存情報で表現した様子を示す．この図には，スペースの関係からノード名とデータムだけを示し，各ノードのラベルと正当化リスト，nogood の環境は**表 1** のほうに示した．図中の破線の上側に，操作 putdown (A), stack (C, D), stack (B, C), stack (A, B) の4操作を実行した直後の依存情報の様子を示す(図1の(5)のモデルの状態に対応する)．また破線の下側には，その後操作 putdown (A) の影響を取り消

表 1 putdown(A) 取消し後のラベル，正当化リスト，nogood 環境の情報
Table 1 Labels, justifications and nogood environments after cancel (putdown(A)).

Operations
ON0:<initial-state, {(0)}, {[0]}>
ON1:<putdown(A), {(0, 1)}, {[ON0, 1]}>
ON2:<stack(C,D), {(0, 2)}, {[ON0, 2]}>
ON3:<stack(B,C), {(0, 1, 3)}, {[ON0, ON1, 3]}>
ON4:<stack(A,B), {(0, 1, 4)}, {[ON0, ON1, 4]}>
ON5:<cancel(putdown(A)), {(5)}, {[5]}>

Other Literals
F1:<clear(A), {(0, 6)}, {[ON0, 6]}>
F2:<on(B,Table), {(0, 7)}, {[ON0, 7]}>
F3:<on(A,B), {(0, 8)(0, 1, 4, 17)}, {[ON0, 8][ON4, 17]}>
F4:<on(D,Table), {(0, 9)}, {[ON0, 9]}>
F5:<clear(C), {(0, 10)}, {[ON0, 10]}>
F6:<on(C,Table), {(0, 11)}, {[ON0, 11]}>
F7:<clear(D), {(0, 12)}, {[ON0, 12]}>
F8:<clear(B), {(0, 1, 13)}, {[ON1, 13]}>
F9:<on(A,Table), {(0, 1, 14)}, {[ON1, 14]}>
F10:<on(C,D), {(0, 2, 15)}, {[ON2, 15]}>
F11:<on(B,C), {(0, 1, 3, 16)}, {[ON3, 16]}>
F12:<height(A,20), {(0, 7, 8)}, {[F2, F3]}>

Nogood Environments
{(1, 8)
(2, 12)
(2, 11)
(3, 7)
(3, 10)
(4, 13)
(4, 14)
(1, 5)}

したときに付加された依存情報を示しており，図全体は図1の(6)に対応する依存情報を表している．本章ではまず破線の上側の場合について説明し，4.5節から破線以下も含めた全体の依存情報について説明する．なお表1は破線以下も含めた全体の依存情報に対応している．

まず各操作とその影響をATMSで管理するために，各操作の実行後のモデルの状態に対応する仮想的なATMSのノードを定義し，これを操作ノード(ON: Operation Node)と呼ぶ．初期状態についても，そのモデルの状態を実現した仮想的な操作を考え，操作ノードを与える．したがって図4では，初期状態に対してON$_0$，が，putdown (A) 実行後のモデルの状態に対応してON$_1$，stack (C, D) に対してON$_2$，stack (B, C) に対してON$_3$，stack (A, B) に対してON$_4$を与えた．これらの操作ノードはATMSの推論事実ノードとして扱う．

操作ノードは，操作実行後のモデルの状態全体を表している．そこで操作ノードとは別に，各操作の実行そのものを表すノードを考え，これをATMSの仮定ノードとして定義する．以後これを操作仮定と呼ぶ．初期状態，putdown (A), stack (C, D), stack (B, C), stack (A, B) の各操作に対して，0, 1, 2, 3, 4の5つの仮定ノードを操作仮定として定義した．操作後のモデルの状態は操作の実行によって得られる．したがって各操作ノードは，図4に示すように対応する操作

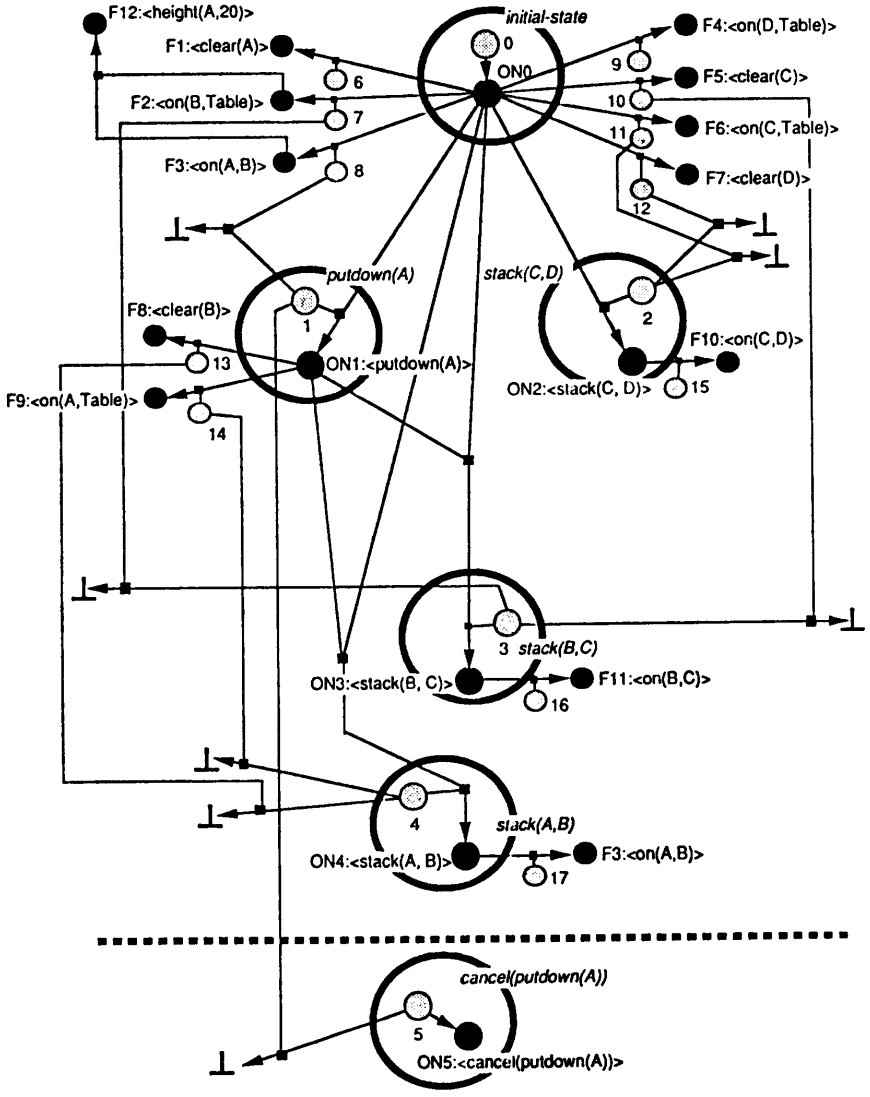


図4 操作間の依存情報を利用した積木の世界のモデリング過程の表現
Fig. 4 Representation of block manipulation process with operational data dependency

仮定に依存する．図4では，各操作について，関係する操作ノードと操作仮定を太い円でくくりその対応を強調した．

4.2 リテラルの付加と消去

操作の前提規則の適用結果の扱いを述べる前に，まず操作のモデルに対する影響の扱いを説明する．われわれは，操作のモデルに対する影響を，変数を含まない任意の数のリテラルのモデルへの付加と削除として扱う．

最初にリテラルの付加の，依存情報による表現を考える．操作 putdown(A) の実行にともない，リテラル on (A, Table) がモデルに付加される．この付加されたリテラルは，付加した操作の実行後のモデルの状態に依存すると考えられる．そこでこのリテラルに対応してノード F9 を定義し，これが操作 putdown (A) の実行後のモデルの状態に対応する操作ノード；ON_1 に依存するように，ATMS に定義すれば良いように思われる．しかしこの依存情報だけでは，F9 と ON_1 のラベルが同じになるため，ラベルを構成する環境を基にデータを管理する ATMS では，2つのノードに区別がつかなくなり面倒が生じる．そこで2つのノードが異なるラベルを持つように，非消去仮定と呼ばれる仮定ノードを各リテラルが付加される度に生成し，依存情報に含める．したがってノード F9 の依存情報は次のようになる，

ON_1 & 14 => F9.

14 が F9 の付加に関する非消去仮定である（図4参照）．この非消去仮定は，各リテラルの個別の付加操作を表すと考えられる．

次にリテラルの消去の依存情報による記録を考える．たとえば初期状態でモデルに付加されたリテラル on (B, Table) は，次の依存情報で ATMS に定義されている（図4参照）．

ON_0 & 7 => F2.

ここで ON_0 は初期状態に対応する操作ノード，F2 はリテラル on (B, Table) に対応する ATMS のノード，7 は F2 の付加に際して与えられた非消去仮定である．このリテラルは操作 stack (B, C) の実行にともない消去される．われわれはリテラルの消去を，そのリテラルの個別の付加操作とリテラルを消去する操作とが共には成立しないこと，すなわち矛盾することと考える．そこで消去したいリテラルの非消去仮定（この例では7）と，消去する操作の操作仮定（この例では stack (B, C) に対応する操作仮定 3) が矛盾を導くように，次の依存情報を ATMS に定義する．

3 & 7 => FALSE.

このような，非消去仮定を用いたリテラルの付加と消去の手法は文献 3) に基づいている．

4.3 前提規則の扱い

次に前提規則の適用結果の依存情報による記録について述べる．直感的には，適用結果にあたるリテラルの連言に対応するノードの集合を，実行する操作の操作ノードにそのまま依存情報として与えれば良いように思われる．しかしこれは正しくない．なぜなら操作の前提となっているリテラルが，その操作によって消去されてしまう可能性があるからである．たとえば図1の例で，積木Bを積木Cに載せる操作 stack (B, C) の前提となる条件のひとつは，積木Cの上面に何もないこと，すなわち clear (C)である．しかしこのリテラルは stack (B, C) の実行によって消去されてしまうので，依存情報に含められない．

このように前提規則の適用結果は，操作の実行直前には正しいことが要求されるが，操作の実行中や実行後にそれらが消去されても，実行された操作は影響を受けない．そこでこれを依存情報で扱うには，適用結果そのものの代りに，操作の実行直前には適用結果が正しければ必ず正しく，かつ操作実行中や実行後に適用結果のリテラルが消去されても影響を受けないノードを，依存情報として操作ノードに与えれば良い．なおここで「正しい」というのは，6章で述べるアルゴリズムにしたがってコンテクストを決定したとき，そのコンテクストにおいて正しいという意味である．

すでに述べたように，操作によってモデルに付加された各リテラルに対応するノードは，操作後の状態を表す操作ノードに依存するように定義される．したがって，付加されたリテラルが正しいとき，対応する操作ノードも必ず正しい．またリテラルが消去されても操作ノードは影響を受けない．そこで前提規則の適用結果を記録するときには，対応するリテラルの連言の代りに，それらのリテラルを付加した操作に対応する操作ノードの集合を，これから実行する操作の操作ノードに依存情報として与えることにする．同じリテラルが，複数の操作によって付加されている場合がある．たとえば，図1のリテラル on (A, B) は「初期状態」と操作 stack (A, B) の2つの操作で付加されている．このようなリテラルが，操作の前提規則の適用結果として用いられる場合には，複数の操作ノードか

ら1つを選ぶ必要がある．まず操作ノードの幾つかは，後述する操作の取消しによって正しくない場合があるので，それらを除く．そして残った操作ノードのうち，最も最近実行された操作に対応する操作ノードを依存情報として採用する．

図4の例では，操作 stack (B, C) の前提条件の適用結果は on (B, Table) & clear(C) & clear (B) である．初めの2つのリテラルは初期状態で，また最後のリテラルは操作 putdown(A) で付加されている．したがって操作 stack (B, C) に対応する操作ノード ON_3 には，操作仮定3に加えて，初期状態の操作ノード ON_0 と putdown (A) に対応する操作ノード ON_1 とを依存情報として与える；

ON_0 & ON_1 & 3 => ON_3.

すべての操作ノードは，幾つかの（なしの場合もある）すでに定義された操作ノードと1つの操作仮定とに依存するように定義されるが，その組合せはただ1つである．したがってすべての操作ノードのラベルは，環境を1つだけ持つ．

4.4 推論によって導かれるリテラルの扱い

操作によって付加されたリテラルを用いて，様々な推論を行うことができる．たとえば，各積木の高さを 10 とするとき，ある積木の上面のテーブルからの高さは，次のような前向きの推論規則として記述できる．

```
on (?x, Table)
->
height (?x, 10).

on (?x, ?y) ^
?y ≠ Table ^
height (?y, ?height-y)
->
height (?x, 10+?height-y).
```

この推論規則では記号 -> の前が推論の前提を，また後が結論を意味する．したがってこれらの規則は，?x がテーブル上にあるなら積木の高さがすなわち ?x 上面のテーブルからの高さであり，そうでなければ，?x の下の積木 ?y のテーブルからの高さに10を加えたものが ?x 上面のテーブルからの高さであることを意味する．

規則の適用に成功したとき，結論にあたるリテラルをモデルに付加することを考える．このとき付加されるリテラルに対応する ATMS のノードには，前提を満たすリテラルの集合を依存情報として与えることにする．たとえば，初期状態でこの推論規則を適用すると，height (A, 20) がモデルに付加されるが，このとき規則の前提を満たすリテラルは on(B, Table) と on(A, B) である．そこで F2, F3, F12 をそれぞれ on(B, Table), on(A, B), height(A, 20) に対応するノードとするとき，次の依存情報を ATMS に定義し，この推論を記録する，

F2 & F3 => F12.

この依存情報の様子を図4の左上に示した．

4.5 操作の取消しの表現

これまで一般の操作の取扱いについて述べたが，同じ依存情報を用いる手法で，操作の影響の取消しの記録も可能である．以後図4の破線以下の部分を含めて説明する．例として操作 putdown(A) を取り消す場合を考える．まず一般の操作の場合と同様に，取消し操作に対応する操作ノードと操作仮定を定義する．取消し操作には前提条件はない．したがってこの取消し操作の操作ノードは，依存する操作ノードを持たない．図4では，取消し操作の操作ノード ON_5 は，対応する操作仮定5にのみ依存するように定義されている．

5 => ON_5.

操作の取消しとは，ATMS ではリテラルの消去の場合と同様に，取り消される操作の実行と，取消し操作の実行とがともには成立しないこと，すなわち矛盾することと考えることができる．そこで取り消される操作 putdown(A) の操作仮定1と取消し操作の操作仮定5とが矛盾することを，ATMS に以下のように定義する（図4参照），

1 & 5 => FALSE.

5. 操作の実行順序の扱い

前章で述べた依存情報だけでは，操作の実行順序を表現することができない．そこで操作の実行順序を，対応する操作ノードの親子関係で記録する．この親子関係は双方向のポインタで記録され，ATMS を用いた依存情報の記録とは異なる．

個々の操作ノードには，この操作の直前に実行された操作の操作ノード（これを親ノードと呼ぶ）と，その操作の直後に実行された操作の操作ノード（これを子ノードと呼ぶ）へのポインタが記録される．この操作ノードの親子関係のポインタを利用して，ある操作ノードから，その操作の前に実行された全操作の操作ノードを実行とは逆順に検索することができる．図4

では実行順序は次のような親子関係で示される．

$ON_0 \leftrightarrow ON_1 \leftrightarrow ON_2 \leftrightarrow ON_3 \leftrightarrow ON_4 \leftrightarrow ON_5$.

ここで初期状態に対する操作ノード；ON_0 は親ノードへのポインタを持たない．また最後に実行された操作の操作ノード；ON_5 は，まだ子ノードへのポインタを持たない．この操作実行順序の記録は，次の章で述べるモデルの作成過程管理アルゴリズムにおいて利用される．

われわれは当初，この実行順序を文献 3) に基づき ATMS で管理し，さらにその上で操作間の依存情報を ATMS で管理することを考え文献 6) に報告した．しかしこの手法では，操作ノードのラベルが操作間の依存情報と実行順序の依存情報の両方に基づいて決まるため，(1)次章で述べるモデル作成過程管理アルゴリズムが複雑になる，(2)実行順序の依存情報は操作の実行回数に比例して大きくなり処理速度に影響を与える，などの問題があった．実行順序を，ATMS とは別に操作ノードの親子関係ポインタで記録した結果，これらの問題をある程度解決することができた．

6. 依存情報に基づくモデリング過程の管理

4 章で述べた依存情報に基づくモデル作成過程の記録手法では，依存情報は記録されているものの，一度付加されたリテラルは実際にはモデルから削除されることはない．そこで依存情報を基に，各リテラルを正しいリテラルなのか，それともモデルから本来削除されているはずのリテラルなのかを選別する必要がある．操作の影響の取消しを行ったときには，リテラルを適切に選別しモデルを再構築する必要がある．このような処理は，操作ノード間の依存情報と親子関係を基に，適切なモデルの状態に対応する ATMS のコンテクストを決定することで実現される．

以下，ある操作の実行後のモデルの状態を決定するための，コンテクストの構成アルゴリズムについて述べる．このアルゴリズムは，過去のモデルの状態の復元と，ある操作を取り消したときのモデルの再構成とを統一的に処理できる．

(1) 使用者は構成したいモデルの状態に対応する操作ノードを選択し，その操作ノードの環境をコンテクストにセットする．

(2) 操作ノード間の親子関係ポインタを用いて，過去に実行された操作ノードを遡る形で辿り，操作ノードを実行とは逆順にすべて検索する．そして検索された順に操作ノードの環境とコンテクストの集合和を新しいコンテクストとする，コンテクストの更新を繰り返す．操作ノードのラベルはひとつしか環境を持たないので，新しいコンテクストは一意に決まる．この際，加えようとする環境とコンテクストとの集合和が，なんらかの nogood の環境をその部分集合に含むときは，その操作ノードについては，コンテクストの更新を行わない．

(3) コンテクストを構成する仮定と矛盾しない非消去仮定をすべてコンテクストに加え，コンテクストを拡張する．この処理も，その非消去仮定とコンテクストとの集合和が，なんらかの nogood の環境を含むかどうかを確認することで実現される．

(4) 拡張されたコンテクストと各リテラルに対応するノードのラベルを比較し，もしもコンテクストに包含される環境が 1 つでもラベルに存在するならば，そのリテラルは正しいとする．それ以外のリテラルは正しくないとする．

図 4 の例で，stack (B, C) を実行した直後のモデル状態を復元する場合を説明する．この操作に対応する操作ノードは ON_3 であり，その環境 (0, 1, 3) をまずコンテクストとする（表 1 参照）．ON_3 の祖先にあたる操作ノードは ON_2, ON_1, ON_0 であり，その環境は (0, 2), (0, 1), (0) である．これらの環境とコンテクストの集合和をあたらしいコンテクストとしても，コンテクストが nogood の環境を含むことはない．したがってこれらとコンテクストとの集合和を順にとり，コンテクストの更新を繰り返すと，最終的にコンテクストは (0, 1, 2, 3) になる．次にこのコンテクストと矛盾しない非消去仮定をすべて加えてコンテクストを拡張すると，図 4 に示されている非消去仮定のうち，on (B, Table) に対応する 7 は 3 と，on (A, B) に対応する 8 は 1 と，clear (C) に対応する 10 は 3 と，on (C, Table) に対応する 11 と clear (D) に対応する 12 は 2 とそれぞれ nogood の環境を構成するためコンテクストに含むことはできず，最終的にコンテクストは，

{0, 1, 2, 3, 6, 9, 13, 14, 15, 16, 17}

となる（表 1 参照）．このコンテクストと，各リテラルに対応するノードのラベルとを比較し（表 1 参照），コンテクストの部分集合にあたる環境を含むノードを選択すると，

clear (A),
on (D, Table),

clear (B),
on (A, Table),
on (C, D),
on (B, C)

がこのコンテクストで正しいことになる．これは図1(4)の stack (B, C) 実行後のモデルの状態に適合する．

操作の影響の取消しにともなうモデルの再構成も，取消し操作直後のモデルの状態を復元すると考えることで同じアルゴリズムが適用できる．まず取消し操作の操作ノード ON_5 の環境(5)をコンテクストとして与え，さらに ON_5 の祖先の操作ノードの環境のコンテクストへの付加を順に試みる．このとき操作仮定1が最初にコンテクストに与えられた操作仮定5とnogood の環境を構成するため，操作仮定1を含む ON_4, ON_3, ON_1 の環境 (0, 1, 4), (0, 1, 3), (0, 1) はコンテクストに付加されない．したがって取り消された操作 putdown(A)と，その操作に依存する操作 stack (B, C), stack(A, B) の環境はコンテクストに付加されず，操作ノード ON_2 と ON_0 の環境 (0, 2) と(0)だけが付加される．最後に矛盾しない非消去仮定を加えてコンテクストを拡張すると，コンテクストは，

(0, 2, 5, 6, 7, 8, 9, 10, 13, 14, 15, 16, 17)

となる．このコンテクストにおいて正しいリテラルは，

clear (A),
on (B, Table),
on (A, B),
on (D, Table),
clear (C),
on (C, D),
height (A, 20)

である（表1参照）．これは図1の(6)の状態に対応しており，モデルは操作 putdown (A) の影響を取り消した後の状態に適切に変更された．

7. ソリッドモデリングへの適用

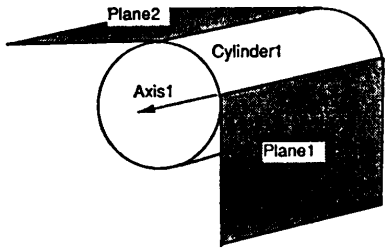
本手法の実用性を評価するために，ATMS を作成し，さらに境界表現法に基づくソリッドモデリングシステムを ATMS 上に開発した．

ソリッドモデリングでは，3次元のモデルを2次元の画面に表示しながら作業を進めるため，使用者が誤った操作を実行しモデルを破壊してしまう問題が指摘されている[1]．また曲面をともなうソリッドモデルの処理では，数値計算の誤差に起因するモデル破壊もしばしば起こる．さらにソリッドモデリングシステムは主に機械設計の支援のために利用されるが，設計作業は本質的に試行錯誤をともなう．以上の理由からソリッドモデルのモデル作成過程管理は重要である．なお本システムはコモンリスプ言語を用いて作成されており，現在リスプマシンで利用できる．

一般にソリッドモデルの幾何情報は，幾何要素間の基本的な制約を基に推論によって決めることができる[7]．たとえば2平面とそれらに接する円筒間の関係は，**図5**に示すような制約として記述できる．そこで次に示す前向きの推論規則によって，2平面の幾何情報が定まれば，円筒面の幾何情報を決めることができる．

```
tangent (? cyl, ? pl 1) ^
tangent (? cyl, ? pl 2) ^
? pl 1 ≠ ? pl 2 ^
axis (? cyl, ? axis) ^
side (? axis, ? pl 1, ? side 1) ^
side (? axis, ? pl 2, ? side 2) ^
radius (? cyl, ? rad) ^
plane-geometry (? pl 1, ? pnt 1, ? nml 1) ^
plane-geometry (? pl 2, ? pnt 2, ? nml 2) ^
set ((? cen ? ax)
     (cyl-tan-2-pls
       ? pnt 1, ? nml 1, ? side 1, ? pnt 2,
                                ? nml 2, ?side 2
       ? rad))
->
cylinder-geometry(? cyl, ? cen, ? ax, ? rad).
```

ここで，述語 tangent(?cyl, ?pl) は，円筒面 ?cyl が平面 ?pl に接することを示し，axis(?cyl, ?axis) は，



tangent(Cylinder1, Plane1),
tangent(Cylinder1, Plane2),
axis(Cylinder1, Axis1),
side(Axis1, Plane1, Below),
side(Axis1, Plane2, Below),
radius(Cylinder1, 50.0).

図5 幾何制約の例
Fig. 5 Some examples of geometric constraints.

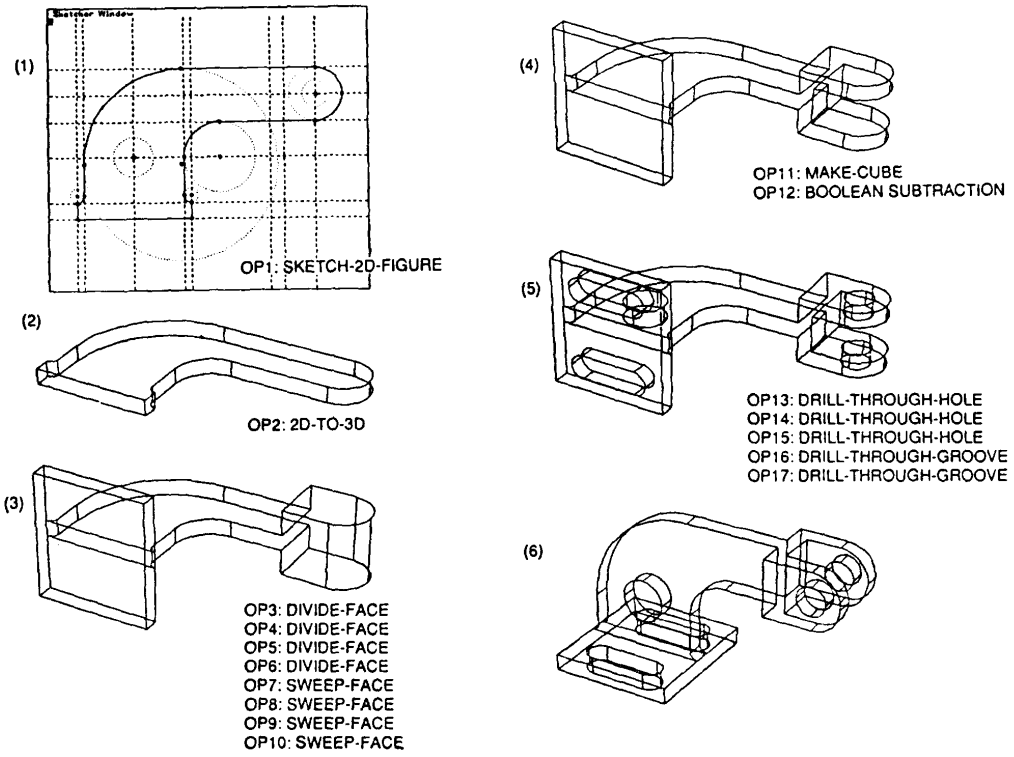


図 6 ソリッドモデル作成過程の例
Fig. 6 Example solid modeling process.

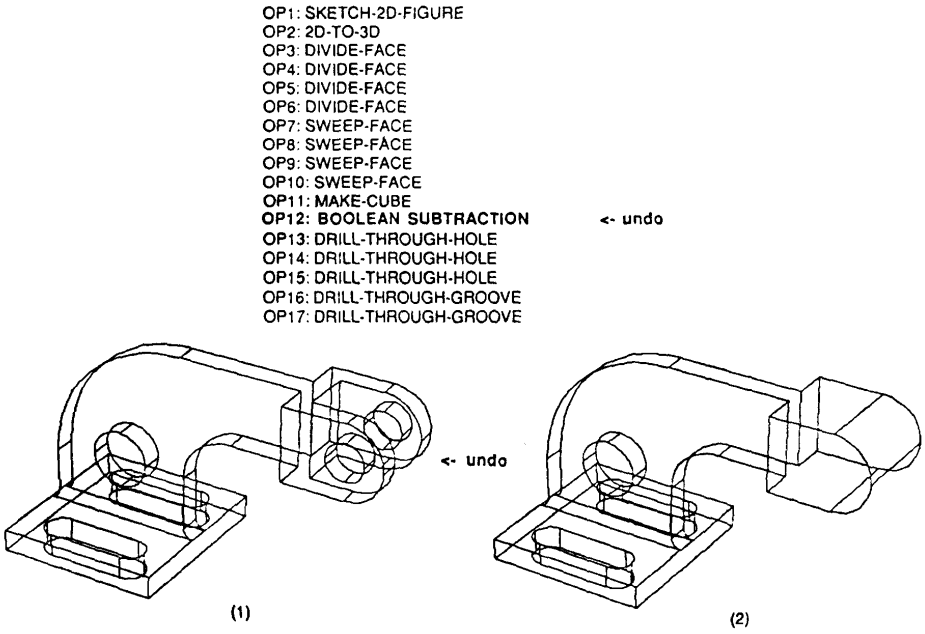


図 7 操作の取消し例
Fig. 7 Undo example.

?axis が円筒面 ?cyl の中心軸にあたる直線であることを，また side (?ln, ?pl, ?side) は，直線 ?ln が平面 ?pl の ?side 側 (Above または Below) にあることを示す．述語 plane-geometry は平面の幾何情報を記述するものである．述語 set は組込述語で，第二引数に与えられたパターンをリスプの関数として評価し，その結果を第一引数の変数にバインドする．この例では，関数 cyl-tan-2-pls を評価し，接平面の幾何情報と半径の情報から円筒面の幾何情報を求めている．この推論規則では，結論として得られた円筒面の幾何情報をモデルに付加している．その際 4.4 節で述べた手法で，前提を満たすリテラルに対応するノードが結論に相当するリテラルに対応するノードに依存情報として与えられる．われわれのソリッドモデリングシステムは，操作によって位相構造と幾何制約を決定し，幾何情報はこのように推論で決定する．

図 6 には，本システムを用いて簡単なソリッドモデルを作成した様子を示す．各操作はその実行に際して，幾つかの位相構造情報を参照する．たとえば穴明け操作では，生成される穴の入口と出口にあたる面の位相構造を参照する．そこでこれらの参照された情報を前提規則の適用結果に相当するリテラルとみなし，4.3 節にしたがい操作ノード間の依存情報を記録した．

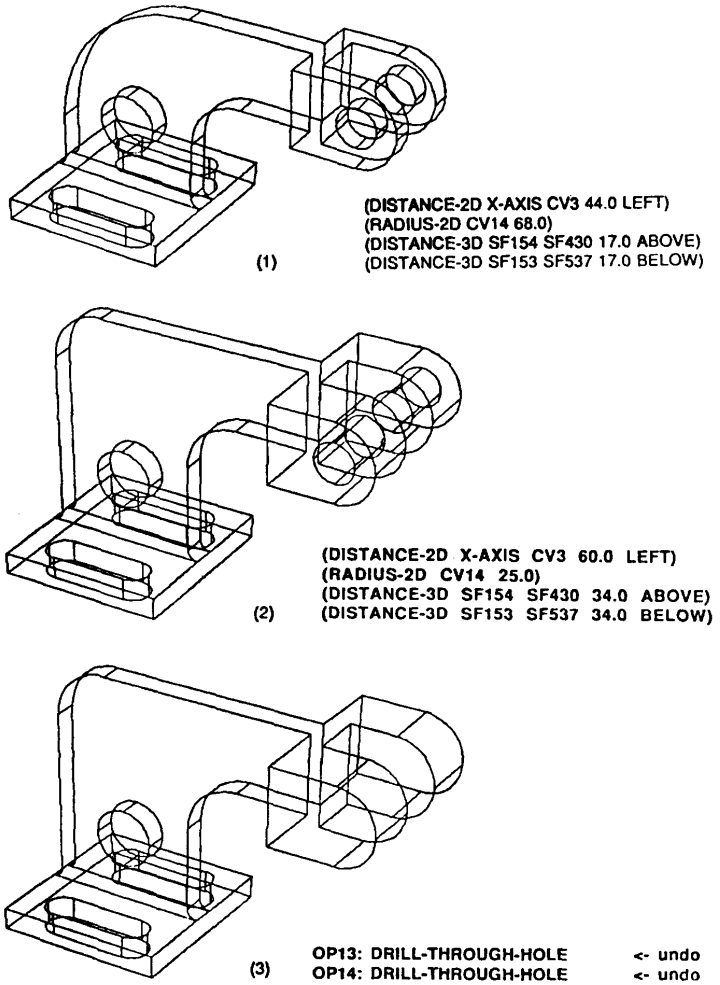


図 8 幾何制約の変更と操作の取消しとを組み合わせた形状変更例
Fig. 8 Shape modification according to geometric constraint changes and undoing.

本システムの機能を利用することで，モデル作成者は図 6 に示された任意のモデルの状態を復元することが可能である．また使用者は特定の作業の影響だけを取り消すことも可能である．**図 7** には，アーム先端部の溝の作成で利用された集合演算を取り消した例を示す．操作間の依存情報を基に，この集合演算とそれに依存する諸操作（この例では穴明け操作）だけが取り消されており，モデルの変化は最小限におさえられている．モデル復元に要する時間は 5 秒以内である．

さらに幾何制約の変更や操作の取消しによって，可変なソリッドモデルを容易に得ることができる．**図 8** (1) に示されているソリッドモデルの幾何情報は，幾何制約に基づいて推論によって決定されている．そこで幾何制約の幾つかを (2) のように変更すると，新たに加えられた制約に基づいて幾何情報が再決定され，幾何形状は (2) のように変化する．この処理に要する時間は 30 秒ほどである．さらにこの段階で穴形状を定義した操作を取り消すと，システムはモデルを再構成し (3) に示す形状を生成する．

8. おわりに

本論文では，モデル作成過程を，操作間の依存情報および操作と操作によって付加されたモデルデータ間の依存情報とで表現し，ATMS を用いて管理する手法を提案した．この手法に基づいて実際にソリッドモデリングシステムを開発し，その実用性の検討も行った．その結果，依存情報を基にモデルデータを操作することで，過去のモデルの状態の復元と，任意の操作

の取消しなど，これまでのシステムでは実現できなかった機能が統一的に実現できることが示された．さらに推論によって幾何情報を決定するシステムと結合することで，ソリッドモデルの可変形状の作成も可能となった．本研究ではソリッドモデリングを例として紹介したが，本手法はモデル操作をともなう様々な問題への適用が可能である．現在プラン生成問題や機械設計問題などへの本手法の適用を検討している．本手法ではモデルデータに加えて依存情報を記録するため，計算機内に保存するデータ量が膨大になる．そこでモデルが確定するにしたがい，不必要となった依存情報を消去する機能の実現も検討している．また本手法の応用として，例えば操作コマンド間の依存情報だけを管理することも可能である．依存情報に基づいて限定されたコマンドだけ再実行することで，処理速度に問題はあるものの，本研究とほぼ同様の機能が実現できると予想される．

参考文献

1) Chiyokura, H. and Kimura, F.: A Method of Representing the Solid Design Process, *IEEE Comput. Graph. & Appl.*, Vol. 5, No. 4, pp. 32-41 (1985).
2) McDermott, D.: A Temporal Logic for Reasoning about Processes and Plans, *Cognitive Science*, Vol. 6, pp. 101-155 (1982).
3) Morris, P. H and Nado, R. A.: Representing Actions with an Assumption-based Truth Maintenance System, *Proc. of 5th National Conference on Artificial Intelligence*, Philadelphia, pp. 13-17 (1986).
4) Fikes, R. E. and Nilsson, N. J.: STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving, *Artif. Intell.*, Vol. 2, pp. 189-208 (1971).
5) de Kleer, J.: An Assumption-Based TMS, *Artif. Intell.*, Vol. 28, pp. 127-162 (1986).
6) Inui, M. and Kimura, F.: Representation and Manipulation of Design and Manufacturing Processes by Data Dependency, *Proc. of IFIP W.G. 5.2 Workshop on Intelligent CAD, 1988* (to be published from North-Holland) (1990).
7) Kimura, F. et al.: Uniform Approach to Dimensioning and Tolerancing in Product Modelling, *Proc. of CAPE '86*, Copenhagen (North-Holland), pp. 165-178 (1986).

(平成元年9月25日受付)
(平成2年4月17日採録)

乾　正知（正会員）

昭和36年生．昭和59年東京大学工学部精密機械工学科卒業．昭和61年同大学大学院情報工学修士課程修了．昭和63年同大学院博士課程退学．同年より，東京大学先端科学技術研究センター助手．機械設計・生産の計算機支援に関する研究に従事．空間や形状に関する推論とその機械設計への応用に興味を持つ．精密工学会，ACM 各会員．

木村　文彦（正会員）

昭和20年生．昭和49年東京大学大学院博士課程修了．同年電子技術総合研究所パターン情報部入所．昭和54年より東京大学工学部精密機械工学科助教授．昭和62年より同教授．マン・マシン・システム，コンピュータ・グラフィックス，形状モデリング，CAD/CAM などの研究に従事．工学博士．IFIP-WG 5.2-5.2-5.3 委員．精密工学会，日本機械学会などの各会員．