



|              |                                                                                                                            |
|--------------|----------------------------------------------------------------------------------------------------------------------------|
| Title        | 関係データベースを用いた在庫管理プログラムの記述とその詳細化の正しさの証明                                                                                      |
| Author(s)    | 森岡, 澄夫; 岡野, 浩三; 東野, 輝夫 他                                                                                                   |
| Citation     | 情報処理学会論文誌. 1995, 36(5), p. 1091-1103                                                                                       |
| Version Type | VoR                                                                                                                        |
| URL          | <a href="https://hdl.handle.net/11094/50247">https://hdl.handle.net/11094/50247</a>                                        |
| rights       | ここに掲載した著作物の利用に関する注意 本著作物の著作権は情報処理学会に帰属します。本著作物は著作権者である情報処理学会の許可のもとに掲載するものです。ご利用に当たっては「著作権法」ならびに「情報処理学会倫理綱領」に従うことをお願いいたします。 |
| Note         |                                                                                                                            |

*The University of Osaka Institutional Knowledge Archive : OUKA*

<https://ir.library.osaka-u.ac.jp/>

The University of Osaka

## 関係データベースを用いた在庫管理プログラムの記述と その詳細化の正しさの証明

森 岡 澄 夫<sup>†</sup>    岡 野 浩 三<sup>†</sup>  
東 野 輝 夫<sup>†</sup>    谷 口 健 一<sup>†</sup>

代数的言語・手法の有用性を調べるために、プログラム設計の共通問題である酒屋在庫管理問題に対して、我々の代数的言語を用いて抽象的順序機械型モデルで階層的に設計し、その実現の正しさの証明を一部行った。ある制限されたスタイルで仕様(プログラムの各処理が満たすべき性質)を記述することによって、整数上の論理式の恒真性判定を用いた簡明な証明手続きにより、詳細化の正しさを半自動で高速に証明できる場合がある。そのスタイル制限のもとでは、例えば、集合を引数とし、その任意の要素がある性質を満たすことを表す述語を用いて、仕様を記述する。本設計例では、そのスタイル制限のもとで自然に記述を行うために、処理の対象となるコンテナ情報などを関係データベースとしてとらえ、また、検証で導入する基本関数の補題をなるべく簡単なものにするため、記述に用いる基本関数を関係データベースの基本的な演算に限った。その記述に対して、上述の証明手続きで詳細化の正しさの証明が比較的容易に行えることを確かめた。本実験の結果より、従来に比べ証明の実作業時間が大きく改善できたことが分かり、本論文で採用した記述スタイル・証明法が有効であることが確かめられた。

## Hierarchical Design of Stock Management Program using Relational Algebra and Its Correctness Proof

SUMIO MORIOKA,<sup>†</sup> KOZO OKANO,<sup>†</sup> TERUO HIGASHINO<sup>†</sup>  
and KENICHI TANIGUCHI<sup>†</sup>

In this paper we present an example of hierarchical design and the correctness proof of "Warehouseman Program for a Liquor Dealer", which was introduced as a common problem for program design. For an abstract level's specification, we refine the specification step by step and prove the correctness of each refinement. Those specifications and program are written in our algebraic language ASL as an abstract sequential machine model. By treating some data in the program as relational databases and using the relational algebra, we described the specifications and program naturally under a restricted style. In the restricted style, for example, we use a predicate which returns true if and only if a property is satisfied by any entry of a database, which is given as an argument of the predicate. Our verification support system can semi-automatically prove the correctness of a refinement, if specifications and programs are written in the restricted style. Using the system, to prove the correctness of a refinement, we have only to give the system some invariant assertions and some theorems for primitive functions/predicates. We have proved in a few days that some typical and important properties in the abstract level's specification hold in the program.

### 1. ま え が き

代数的言語は、意味定義が厳密で、任意の抽象レベルの記述を同じ言語で記述でき、記述の正しさの証明を形式的に行える等の特徴を持つ。従来、我々の研究グループでは、実用上の見地から代数的言語・手法の有用性を調べるための一つの例題として、プログラム

設計の共通問題である酒屋在庫管理問題<sup>1)~3)</sup>を選び、我々の代数的言語<sup>4)</sup>を用いて抽象的順序機械型モデルで階層的に設計し、実際に動作するプログラムを開発し<sup>5)</sup>、一部ではあるが詳細化の正しさの証明を行ってきた<sup>6)</sup>。

そこでは、処理の要求性質を記述する際に、例えば「コンテナリスト中の各コンテナ  $i$  について、 $i$  はある述語を満たす」といったように任意の要素を表す変数  $i$  を陽に用いていた。証明では「リスト  $list$  中の各要素  $i$  が述語  $p(i)$  を満たすならば、 $list$  に述語  $q(e)$  を満た

<sup>†</sup> 大阪大学基礎工学部情報工学科  
Department of Information and Computer Sciences,  
Faculty of Engineering Science, Osaka University

す  $e$  を追加したリスト  $append(list, e)$  中の各要素  $j$  は  $p(j)$  を満たす」というような推論が必要になってくる可能性があるが、上述のように変数を陽に用いた性質記述をしていると、

$$\{(\forall i \in list \supset p(i)) \wedge q(e)\} \\ \supset \{\forall j \in append(list, e) \supset p(j)\}$$

という形の論理式で表される補題が必要となる。しかしながら、限定子の使用を許さない等式論理では、このことをすべての変数が自由変数である等式として表現することはできないので、等式論理に基づく代数的手法では、上述のような形の補題を直接扱うことはできない。文献6)の場合には、帰納法で変数を定数と見なすなどの工夫をして証明したが、それでも検証者が良く考えて、帰納法、場合分け、および、項書換えなどを複雑に組み合わせて証明を行わなければならない。証明手続きは繁雑であった。

本論文では、証明で上述のような形の論理式を扱わなくても済むようにするため、リスト  $list$  に関する性質  $\forall i \in list \supset p(i)$  を、 $list$  のみを引数とする述語  $P(list)$  を用いて表すよう、仕様記述スタイルを制限する。このような述語  $P$  を用いると、前述の性質は

$$\{P(list) \wedge q(e)\} \supset P(append(list, e))$$

の形に書け、自由変数  $list$ ,  $e$  を使った等式で書ける。

また、そのような述語を用いた仕様記述のもとでは、加減・比較演算と論理演算からなる整数上の論理式(プレスブルガー文)の恒真性判定手続き<sup>7)</sup>を利用した簡明な証明手続きにより、詳細化の正しさの証明を行える<sup>8)</sup>。その手続きで証明が成功した場合、証明に用いた基本関数・述語に関する補題が正しいという前提のもとで、詳細化の正しさが保証される。我々の開発している検証支援系は、構造的帰納法に必要な不変表明と基本関数・述語に関する補題<sup>\*</sup>を検証者が与えれば、文献8)の証明法に基づき、証明を自動で高速に行える。

しかしながら、以上の記述法・証明法を実際の問題に適用する際、以下のような問題を避けるため、記述に用いる関数・述語をなるべく一般的かつ簡単なものにすべきである。まず、我々の証明手続きでは、検証者が基本関数・述語に関する補題を自由に導入することを許しているが、例えばプログラムが満たすべき性質をそのまま一つの述語で表現してしまうと、プログラム設計の正しさの議論の本質となるところを、補題として無条件に認めてしまうおそれがある。また、複雑な性質を表す述語を複数導入して仕様を記述した場

合、それらの述語間に成り立つ複雑な関係を補題として導入しなければならないとなり、補題の正しさを無条件に認めるのが困難になることが考えられる。

上述の記述スタイル制限のもとで、実用的なプログラムの仕様が自然に記述できるかどうか、また、証明をどの程度の手間でできるか調べるのが重要である。本論文では、酒屋在庫管理問題を例題として、その仕様記述と設計を記述スタイル制限のもとで行い、プログラムが満たすべきいくつかの重要な性質について、実際に検証支援系を用いて証明を行った。

今回は、コンテナ集合や出庫指示書などプログラムで取り扱う主なデータを、それぞれ関係データベースとしてとらえることにした。例えば、コンテナ集合を、コンテナ番号、日付、品名、および在庫数量を属性として持つ四つ組(エントリ)の集合(データベース)としてとらえる。

仕様にはプログラムの各処理の前後で各データベースが満たすべき幾つかの性質を記述するが、先のスタイル制限のもとで記述を行うために、それらの各性質を表す基本述語として、引数として与えられたデータベースの任意のエントリに対し、その属性値間に例えば大小関係などが成り立つとき、真となるようなものを用いることにした。このような述語は先の  $P(list)$  に相当するものである。

また、基本述語が表す性質を十分簡単なものにするために、一つのデータベースの性質を表す基本述語だけを用いることにした。複数のデータベース間に成り立つ関係を記述する場合<sup>\*\*</sup>には、それらのデータベースに関係データベースの演算<sup>10)</sup>(これらは基本関数とする)を適用して一つのデータベースを得て、そのデータベースを述語の引数に与える。

以上のような比較的単純な基本述語・関数を用いたが、記述はそれほど長くならず、自然に仕様を記述することができた。また、証明では、不変表明や基本述語・関数に関する補題を比較的簡単に考案することができ(各述語・関数の定義に相当するものなどを導入した)、文献6)で証明を行った場合と比べ実作業時間が数倍程度減り、本例題においても採用した記述スタイルが有効であることを確かめた。

本論文で検証を行った出庫処理の実現は、検証の易しさを意識して行ったものではない。往々にして、検証の易しさを目指すと、実行効率の悪いプログラムに

\* 正確には、等式として書かれた補題の変数に、ある制約のもとで値を代入したインスタンスを検証支援系に与える。  
4.2節参照。

\*\* 例えば、コンテナデータベースを変更する処理については、処理前のコンテナデータベースと処理後のコンテナデータベースの間の関係などを、その処理の要求性質として書く必要がある。

なる恐れがあるが、本論文で設計したプログラムはコンパイルされ、人が直接C言語で作成したものと同程度の速さで動作することが確認されている<sup>8)</sup>。

以下、2章で階層的詳細化の方法および記述上の制約について述べ、3章で在庫管理問題の記述と詳細化の例について述べる。4章で証明方法と、実際の証明例について述べる。

## 2. 階層的記述方法

### 2.1 諸定義

記述には、我々の研究グループが定義した代数的言語ASL<sup>4)</sup>のサブクラスである抽象的順序機械型(ASL/ASM)<sup>9)</sup>を用いる。

一般に代数的言語は、公理により表現式の等式集合を指定し、その等式集合から導かれる合同関係を、記述の意味定義としている。ASLでは、公理によって生成される等式集合から導かれる最小の合同関係を記述の意味定義とし、また表現式を定義するにあたって、関数の構文やソート間の包含関係を自然に定義できるように文脈自由文法を用いているのが、その特徴である<sup>4)</sup>。

ASL/ASMによる記述(以下、ASM型記述と呼ぶ)では、抽象状態を表すデータ型stateを導入し、以下のような関数を用いる。(A)状態成分関数は、抽象状態を引数とし、抽象状態におけるプログラム変数の値を表す。(B)状態遷移関数は、抽象状態を引数とし、遷移後の抽象状態を表す。(C)ループ関数は、抽象状態を引数とし、複数の状態遷移関数および他のループ関数間の部分的な実行順序を指定する。

ASM型記述は、一般に、有限個のレジスタ(データ型は任意)と制御部を持つレジスタ付き状態機械を代数的に記述しているものと思って良い<sup>\*</sup>。具体的には、各状態遷移の動作内容と、各遷移の実行順序・実行条件(以下、実行指定とも呼ぶ)を指定する。

一つの遷移 $T$ の動作内容を指定するには、その実行前後の状態において各状態成分関数の値が満たすべき関係を、公理で記述する(一般に、複数個の公理を用いる)。いま、あるレベルの記述において $p$ 個の状態成分関数が使われているとする。状態成分関数 $F_i(1 \leq i \leq p)$ の、遷移 $T$ の実行前後での値の関係を

$$F_i(T(s)) = C_i(F_1(s), \dots, F_p(s))$$

のように、 $T$ の実行後の $F_i$ の値が実行前の各 $F_i(1 \leq j \leq p)$ で定まるように記述すれば(各 $C_i$ の値は、各引数の取りうる値に対して、定義されていなければなら

ない)、その公理を代入定義と呼ぶ。ここで $s$ はstate型の変数である。また、各状態成分の値の関係を

$$P_k(F_1(s), \dots, F_p(s), F_1(T(s)), \dots, F_p(T(s))) \\ = \text{TRUE}$$

のように、 $T$ の実行前後の状態成分値を引数とするような述語 $P_k$ で指定するときは、その公理を性質記述と呼ぶ。

本論文では、まえがきで述べたように、証明の省力化のために、例えば「集合の任意の要素がある関係を満たす」ことを、任意の要素を表す変数( $i$ など)を陽に用いず、 $P(\text{list})$ のような述語を導入して書くことにする。これは厳密には、動作内容を指定する各公理を「state型の変数 $s$ 以外の変数を使用しない」という制約のもとで書く、ということである。

実行指定は、ループ関数( $L$ )を用いて記述するが、本論文では、実行指定の記述スタイルも以下の(C-1)または(C-2)の形に制限する。

$$(C-1) L(s) = \text{if } p(s) \text{ then } LT \text{ else } LT'$$

$$(C-2) L(s) = LT$$

ただし $L'$ を他のループ関数として、

$$LT, LT' = L'(T(s)) \text{ または } T(s) \text{ または } s.$$

ここで、(C-1)のif関数の条件部 $p(s)$ は、 $s$ 以外の変数を用いずに記述する。

### 2.2 抽象的順序機械型記述の階層的設計法

我々の提案する階層的設計法では、抽象度の高いレベルから実行可能なプログラムのレベルまで、逐次詳細化を行う<sup>9)</sup>。詳細化では、第 $k$ レベルに性質記述を用いて動作内容が指定された遷移 $T$ があるとき、それを第 $k+1$ レベルでより具体的な遷移 $t_1, \dots, t_n$ を用いて実現する( $T$ をループ関数として $t_1, \dots, t_n$ の実行順と実行条件を指定する)。必要であれば、第 $k+1$ レベルで新しい状態成分関数をいくつか導入する<sup>\*\*</sup>。遷移 $t_1, \dots, t_n$ の動作内容は、性質記述や代入定義を用いて指定する。第 $k$ レベルの複数の遷移 $T_1, \dots, T_m$ を、第 $k+1$ レベルで同時に詳細化しても良い。第 $k+1$ レベルの記述は、第 $k$ レベルの記述中の公理のうち、遷移 $T$ の動作内容の公理以外のものをすべて含む。詳細化の実例を3.3.2項で示す。

## 3. 在庫管理の階層的記述

酒屋在庫管理問題は、一つのコンテナ入庫票や新たな出庫依頼書の入力に対して、在庫情報や未出庫依頼書に関するデータを更新し、出庫可能であれば、各コンテナ中の酒の品名と数量等に関する在庫情報や、未

\* 一般には、制御部は状態数有限でなくてもよい。

\*\* 簡単のため、新たに導入される状態遷移関数は既存の状態遷移関数とは異なる名前であるとする。

処理の出庫依頼書(未出庫依頼書)に関するデータから、出庫依頼書に対する出庫指示書を作成・出力するといった、伝票業務をプログラムにする問題である<sup>3)</sup>。

### 3.1 抽象レベルの記述のためのデータ型と基本関数

まえがきで述べたように、本論文では、2.1節で述べたスタイル制限のもとで仕様を記述するため、コンテナ集合(在庫情報)や出庫指示書などプログラムで取り扱う主なデータを、それぞれ関係データベースとしてとらえることにし(詳しくは3.2節で述べる)、以下のような基本述語を用いることにした。

$EachPr_{p(X)}(R)$ : データベース  $R$  中の任意のエントリについて、その属性  $X$  のデータが述語  $p$  を満たすときのみ、真となる述語。 $EachPr_{p(X)}(NULL)$  は便宜上真とする。 $R$  の属性数および  $p(X)$  ごとに異なる述語記号を用いるべきであるが、簡便さのため、このような記法を用いる。

$Uniq_X(R)$ : データベース  $R$  の、どの二つのエントリについても、属性  $X$  の値が互いに異なるときのみ、真となる述語。先の  $EachPr_{p(X)}(R)$  と同様、これも簡易記法である。

まえがきで述べたように、これらはいずれも、一つのデータベースを引数として、そのデータベースの任意のエントリで、あるいは任意の二つのエントリ間で、ある性質が満たされるとき真となるような述語である。複数のデータベース間に成り立つべき関係を記述する場合は、それらのデータベースから、関係データベースの演算<sup>10)</sup>(関係代数)を用いて、上述の基本述語の引数に与える一つのデータベースを得る。関係データベースの演算は基本関数とする<sup>\*</sup>。演算としては、自然結合  $R_1 \bowtie R_2$ 、集合和  $R_1 \cup R_2$ 、集合差  $R_1 - R_2$ 、射影  $\pi_X R$ 、選択  $\sigma_{A=a} R$ 、属性名変換  $\rho_{A|B} R$  等があげられる。ここで、 $X$  は属性名の集合であり、 $A, B$  は属性名である。

自然結合  $R_1 \bowtie R_2$  は、 $R_1, R_2$  のそれぞれの属性集合  $X, Y$  の和を属性集合とし、以下の式を満たす( $t, t_1, t_2$  はエントリ、 $t[X]$  はエントリ  $t$  の属性  $X$  の値である)。

$$R_1 \bowtie R_2 \equiv \{t | t_1 \in R_1, t_2 \in R_2, \\ t[X] = t_1[X], t[Y] = t_2[Y]\}$$

同様に他の演算も定義できる。

$$R_1 \cup R_2 \equiv \{t | t \in R_1 \vee t \in R_2\},$$

$$R_1 - R_2 \equiv \{t | t \in R_1, t \notin R_2\},$$

$$\pi_X R \equiv \{t' | t' = t[X], t \in R\},$$

$$\sigma_{A=a} R \equiv \{t | t[A] = a, t \in R\},$$

$$\rho_{A|B} R \equiv R \text{ の属性名 } B \text{ を } A \text{ に変更したもの.}$$

$$M_{X,A} R \equiv \{t' [X \cup \{A\}]\}$$

$$t' [A] = \sum_{t \in R, t'[X] = t[X]} t[A].$$

すなわち、 $R$  の属性のうち  $X$  と  $A$  を属性として持つデータベースで、 $X$  の各値をキーとして  $R$  中の  $A$  の総和をとったもの。

$$AllOp_{op(X)}(R) \equiv op_{t \in R} (t[A]).$$

すなわち、 $R$  の全てのエントリ中の属性  $X$  の値を、演算  $op$  で結合し、その式の値を返す関数。 $R$  が NULL の場合は、値は未定義とする。 $EachPr_{p(X)}(R)$  などと同様、これも簡易記法である。

また、データベースの操作関数として、 $Open(R)$ ,  $Delete(R)$ ,  $Close(R)$ ,  $Insert(R, entry)$  などを用いる(これらも基本関数とする)。ここで、 $Delete(R)$  は、 $R$  から勝手な一つのエントリを削除した後のデータベースと、削除されたエントリとの組を値として返す関数とし(エントリの削除の順番は不定とする)、 $P1>Delete(R))$  が削除後のデータベース、 $P2>Delete(R))$  が削除の対象となったエントリの値をそれぞれ表すとする。 $R$  が NULL の場合の  $Delete(R)$ ,  $P1>Delete(R))$ ,  $P2>Delete(R))$  の値は、それぞれ未定義とする。これらの関数は SQL など幾つかの言語で標準的に用いられているものである。

### 3.2 階層的設計の概略

3.1節で述べた基本述語・関数を用いて、在庫管理プログラムを4段階で詳細化した。詳細化の概要を図1に示す。詳細化でのレベル分けは、文献5)と同様の方針で行った。

第一レベルでは「(入力された)コンテナ入庫票に対する処理」, 「(入力された)出庫依頼書に対する処理」, および「終了処理」の性質の記述を行い、これらの処理の実行制御<sup>\*\*</sup>をループ関数などにより記述した。「コンテナ入庫票に対する処理」では、一般に、複数の未出庫依頼書に対する出庫指示が作られる。このレベルの状態成分関数は、出庫依頼書、未出庫依頼書データベース、コンテナ入庫票、コンテナデータベース、出庫指示からなる。

ここで、出庫依頼書、未出庫依頼書データベースとともに、属性集合  $\{IDr, addr, name, qtyq\}$  を持つデ

<sup>\*</sup> 1 引数の演算については、引数が NULL の場合、演算結果は NULL とする。自然結合については、いずれか一方の引数が NULL ならば、演算結果も NULL とする。集合和、集合差で、引数が NULL の場合の演算結果は、通常の集合の演算の場合と同様に定義する。

<sup>\*\*</sup> プログラムの終了指示が来るまで、入力されるデータに応じて、出庫依頼書に対する処理またはコンテナ入庫票に対する処理を繰り返す。

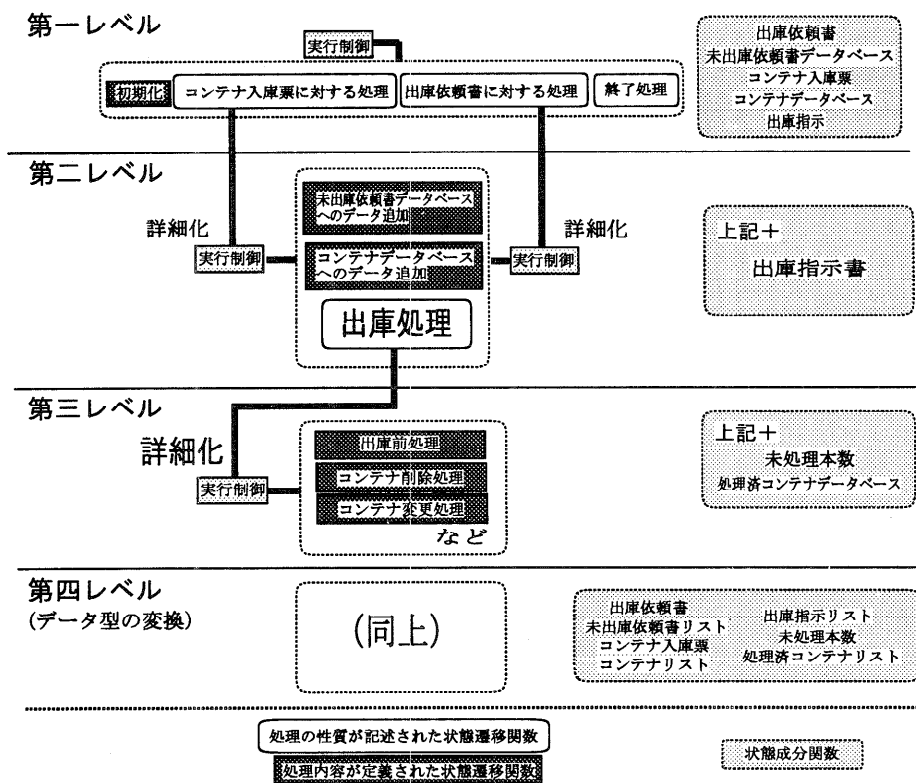


図1 在庫管理の階層的設計

Fig. 1 Hierarchical design of the stock management program.

ータベースである。各属性はそれぞれ、依頼書番号、送り先、出庫を依頼する品名、出庫を依頼する数量を表す。未出庫依頼書データベースは出庫依頼書の集合であり、データベースの一エントリが一枚の出庫依頼書に相当する。

コンテナ入庫票、コンテナデータベースはともに、属性集合  $\{IDc, date, name, qty\}$  を持つデータベースである。各属性はそれぞれ、コンテナ番号、日付、品名、(その品目の)在庫数量を表す。同一のコンテナは複数個の異なる品名を含むので、コンテナ番号が同じエントリ全体で一つのコンテナを表すことになる。

出庫指示(第二レベルでの「一枚の出庫指示書」の集合)は属性集合  $\{IDr, addr, name, IDc, qty, empty\}$  を持つデータベースである。各属性はそれぞれ、依頼書番号、送り先、品名、コンテナ番号、出庫数量、空コンテナマークを表す。データベースの一エントリは「どの依頼書に対応する出庫指示か、どこへ、どの品名を、どのコンテナから何本出すか、(そのエントリの表示指示に従って出庫したとき)そのコンテナは空になるか」を表す。一枚の出庫依頼書に対して、複数のコン

テナから出庫する場合は、同じ依頼書番号を持つ複数のエントリで、その出庫依頼書に対する出庫指示を表す。

第二レベルでは、第一レベルの「コンテナ入庫票に対する処理」と「出庫依頼書に対する処理」を詳細化した。

このために、一枚の出庫依頼書に対して、出庫可能などきに、それに対応する一枚の出庫指示書(これは第一レベルの出庫指示と同じ型のデータベースであり、同一の  $IDr$  を持つ複数のエントリからなる)を作成し、コンテナデータベースなどを変更する「出庫処理」を導入し、その性質を記述した(その記述例を 3.3.1 項で示す)。

この「出庫処理」と、未出庫依頼書データベースまたはコンテナデータベースにデータを追加する処理を用いて、上述の二つの処理をそれぞれ実現した。まず「コンテナ入庫票に対する処理」については、コンテナ入庫票の内容をコンテナデータベースに追加(処理)コンテナデータベースへのデータ追加を実行した後、未出庫依頼書データベースから出庫可能な出庫依頼書の一つずつ探して、それらに対する出庫指示書を出庫

処理」の実行により作成するように実現した。また、「出庫依頼書に対する処理」については、その出庫依頼書に対し、出庫可能であれば、それに対する出庫指示書を作成（「出庫処理」を実行）し、出庫不可能であれば出庫依頼書の内容を未出庫依頼書データベースに追加（処理「未出庫依頼書データベースへのデータ追加」を実行）するように実現した。

第三レベルでは、「出庫処理」を詳細化した。どのように詳細化したかについては 3.3.2 項で述べる。

第四レベルでは、データ型の変換を行った。第三レベルまでは、未出庫依頼書のコンテナの集合を、データベース型として扱ってきた。この第四レベルで、コンテナや未出庫依頼書を「古いものから処理していく」と（処理の順番を）定め、それに合わせて、これらのデータ型をデータベース型から、抽象リスト型へ変換した。この記述は C 言語に変換でき、実行できる<sup>8)</sup>。

### 3.3 記述と詳細化の例

ここでは、在庫管理プログラムのうちの重要な処理である「出庫処理」*Prc-Make-Order* の性質記述（第二レベル）と詳細化（第三レベル）の例について説明する。この処理では、一枚の出庫依頼書（処理前に、処理の引数として状態成分 *Req* に代入されている）に対して、出庫可能なときに、それに対応する一つの出庫指示書 *OrdD* を作成する。

#### 3.3.1 記述例

「出庫処理」に対する要求記述例を図 2, 3, 6 に示す。各公理（等式）では、例えば、出庫処理前のコンテナデータベース *PRE-AContD* (*AContD*(*s*)) のこと。図 2 の *define* 文を参照)、出庫処理後のコンテナデータベース *POST-AContD*、および出庫処理の実行により作られる出庫指示書 *POST-OrdD* などが満たすべき関係を、性質記述の形で書いている（記述すべき内容については文献 5) 参照）。

ここで「出庫処理」に対する要求では、処理の実行前に前提条件 *PRE-COND-MO* (図 2 に示す。出庫依頼書で指定された品目が出庫可能であることや、出庫指示書が NULL であることなどを表す) が成り立つことを仮定している。後に出庫処理を詳細化する際には、その条件 *PRE-COND-MO* が処理の実行前に成り立つという前提のもとで実現を行えば良い<sup>8)</sup>。

記述のうち、例えば公理 O1 (図 3) は、「出庫処理の実

```
(** 出庫処理 Prc-Make-Order の性質記述の一部 **)
define 'PRE-AContD' := 'AContD(s)';
(* 処理前のコンテナデータベース *)
define 'POST-AContD' := 'AContD(Prc-Make-Order(s))';
(* 処理後のコンテナデータベース *)
define 'POST-OrdD' := 'OrdD(Prc-Make-Order(s))';
(* 処理後の出庫指示書 *)
define 'POST-AContD_NAME' := ' $\pi\{IDc, name\} POST\_AContD$ ';
define 'POST-OrdD_NAME' := ' $\pi\{IDc, name\} POST\_OrdD$ ';
define 'PRE-AContD_QTY' := ' $\pi\{IDc, name, qty\} PRE\_AContD$ ';
define 'POST-AContD_QTY' := ' $\pi\{IDc, name, qty\} POST\_AContD$ ';
define 'POST-OrdD_QTY' := ' $\pi\{IDc, name, qty\} POST\_OrdD$ ';
define 'PRE-AContD_NAME_QTY'
  := ' $\pi\{IDc, qty\}(\sigma\{name=\pi\{name\} Req(s)\} PRE\_AContD)$ ';
define 'PRE-AContD_Total_QTY' := ' $M\{IDc, qty\} PRE\_AContD$ ';
(* 出庫前の各コンテナごとの総在庫数量 *)
(** 出庫処理の前提条件 PRE-COND-MO **)
define 'PRE-COND-MO' :=
   $\neg Req(s) = NULL$ 
   $\wedge \neg PRE\_AContD = NULL$ 
   $\wedge \neg OrdD(s) = NULL$ 
   $\wedge \neg (\sigma\{name=\pi\{name\} Req(s)\} PRE\_AContD) = NULL$ 
   $\wedge EachPr_{(qty > 0)}(PRE\_AContD)$ 
   $\wedge 0 < \pi\{qty\} Req(s)$ 
   $\leq AllOp_{+}(qty)(\sigma\{name=\pi\{name\} Req(s)\} PRE\_AContD)$ 
   $\wedge Uniq\{IDc\}(\pi\{IDc, qty\} PRE\_AContD)$ 
   $\wedge Uniq\{IDc, name\}(PRE\_AContD)$ ;
(* 処理前の出庫依頼書、コンテナデータベースは NULL でない。
  出庫前の (1 枚の) 出庫指示書は NULL である。
  処理前のコンテナデータベースに該当品目がある。
  処理前のコンテナデータベースの各在庫数量は正。
  出庫依頼書中の出庫数量は正。
  コンテナデータベース中に、出庫依頼以上の
  該当品目がある。
  コンテナデータベース中の IDc は Uniq.
  コンテナデータベース中の IDc と name の組は Uniq. *)
(** PRE-AContD, POST-AContD, POST-OrdD の関係 **)
define 'RC1' :=
   $(\pi\{IDc, qty\} PRE\_AContD\_QTY)$ 
   $\bowtie (\pi\{IDc, qty\} POST\_AContD\_QTY)$ 
   $\bowtie (\pi\{IDc, qty\} POST\_OrdD\_QTY)$ ;
(* 図 5 の RC1 *)
define 'RC2' :=
   $(\pi\{IDc, qty\} PRE\_AContD\_QTY)$ 
   $\bowtie (\pi\{IDc, qty\} POST\_AContD\_QTY)$ ;
(* 図 5 の RC2  $\cup$  RC1 *)
define 'RC3' :=
   $(\pi\{IDc, qty\} PRE\_AContD\_QTY)$ 
   $\bowtie (\pi\{IDc, qty\} POST\_OrdD\_QTY)$ ;
(* 図 5 の RC3  $\cup$  RC1 *)
```

図 2 第 2 レベルでの仕様の例

Fig. 2 ASL text for level 2.

行前に *PRE-COND-MO* が成り立つならば、*POST-OrdD* の各エントリについて、その出庫数量 *qtyr* の値は正で、かつ、対応する（属性 *IDc* の値が同じで、*Req*(*s*) で指定された品名を属性 *name* の値とする）*PRE-AContD* のエントリの在庫数量 *qtyc* の値以下であるべき」という要求を表している。

公理 O1 ではこの要求を、述語 *EachPr*<sub>(*qtyc* ≥ *qtyr* ∧ *qtyr* > 0)</sub> を用い、その引数に次のような一つのデータベース *R* を与えることで、記述している：*R* は {*IDc*, *qtyr*, *qtyc*} を属性として持つデータベースであり、そのエントリは *POST-OrdD* の各エントリと 1 対 1 に対応している。*R* の各エントリの属性 *IDc*, *qtyr*, *qtyc* の値はそれぞれ、対応する *POST-OrdD* のエントリ *entry* の属性 *IDc*, *qtyr* の値と、*entry* に対応する *PRE-AContD* のエントリの属性 *qtyc* の値である。

<sup>8)</sup> 出庫処理を実行する前に *PRE-COND-MO* が成り立つことは、第一レベルから第二レベルへの詳細化の際に証明すべきである。または、第二レベルの各状態で *PRE-COND-MO* が不変式として成り立つことを証明しても良い。

(\*\*\*\* 出庫指示書に関する性質 \*\*\*\*)

O1: *PRE\_COND\_MO*

⊃ *EachPr*<sub>(qty<0)</sub>(*PRE\_AContD.NAME\_QTY*  
 $\bowtie \pi_{\{IDc, qtyr\}} POST\_OrdD$ ) == TRUE;

(\* 出庫指示書中の各出庫数量は正かつ該当品の出庫前の各コンテナの在庫数量以下 \*)

O2: *PRE\_COND\_MO*

⊃ *EachPr*<sub>(empty iff (qtyr=qtyc))</sub>(*PRE\_AContD.Total\_QTY*  
 $\bowtie \pi_{\{IDc, qtyr, empty\}} POST\_OrdD$ ) == TRUE;

(\* 出庫指示書中の各出庫数量と出庫前の各コンテナごとの総在庫数量が一致すれば、そのときに限り empty の値が真である \*)

O3: *PRE\_COND\_MO*

⊃ *AllOp*<sub>(+ (qtyr))</sub>(*POST\_OrdD*) =  $\pi_{\{qtyr\}} Reg(s)$  == TRUE;

(\* 出庫指示書中の総出庫数量と出庫依頼書中の数量は同じ \*)

など

(\*\*\*\* コンテナデータベースに関する性質 \*\*\*\*)

C1: *PRE\_COND\_MO*

⊃ ( *EachPr*<sub>(qty<0)</sub>(*POST\_AContD*)  
 $\wedge Uniq\{IDc\}(\pi_{\{date, IDc\}} POST\_AContD)$   
 $\wedge Uniq\{IDc, name\}(POST\_AContD)$  ) == TRUE;

(\* 出庫後のコンテナデータベースの各在庫数量は正  
 出庫後のコンテナデータベースの IDc は Uniq  
 出庫後のコンテナデータベースの IDc と name の組は Uniq \*)

など

図3 第2レベルでの仕様の例一続きー

Fig. 3 ASL text for level 2—Continued.

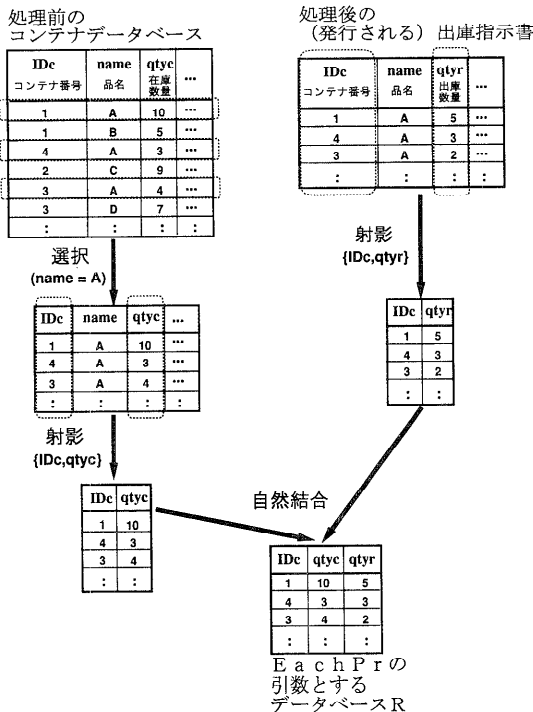


図4 公理 O1 の性質記述における関係代数の使用

Fig. 4 Use of relational algebra at describing the axiom O1.

このデータベース *R* を、*POST\_OrdD* と *PRE\_AContD* から関係データベースの演算を用いて構成した。図4に演算をどのように組み合わせて *R* を

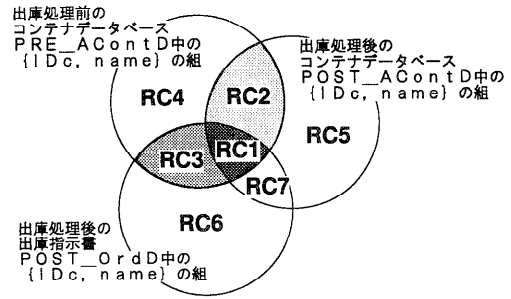


図5 「出庫処理」前後における各データベースの関係

Fig. 5 Relations between databases before and after an execution of the transition “Prc-Make-Ordem” in level 2.

SET1: *PRE\_COND\_MO*

⊃ *EachPr*<sub>(qty=qty1+qty2)</sub>(*RC1*) == TRUE;

(\* 各コンテナごと、各品名ごと、出庫後の在庫数量と出庫指示書の出庫数量を合わせると出庫前の在庫数量に等しい (図5のRC1に関する性質) \*)

SET2: *PRE\_COND\_MO*

⊃ *EachPr*<sub>(qty=qty1)</sub>(*RC21* -  $\pi_{\{IDc, name, qty, qty1\}} RC1$ ) == TRUE;

(\* 各コンテナごと、各品名ごと、出庫指示書にあらわれないエントリの在庫数量は変化しない (図5のRC2に関する性質) \*)

SET3: *PRE\_COND\_MO*

⊃ *EachPr*<sub>(qty=qty2)</sub>(*RC31* -  $\pi_{\{IDc, name, qty, qty2\}} RC1$ ) == TRUE;

(\* 各コンテナごと、各品名ごと、処理後に在庫がなくなったら、出庫前の在庫数量と出庫指示書の出庫数量は等しい (図5のRC3に関する性質) \*)

SET4: *PRE\_COND\_MO*

⊃ *PRE\_AContD.NAME* = *PRE\_AContD.NAME*  
 $\bowtie (POST\_OrdD.NAME \cup POST\_AContD.NAME)$   
 == TRUE;

(\* IDc と name の組についてみたとき、処理前のコンテナは処理後のコンテナと出庫依頼書の Union のサブセット (図5のRC4は NULL) \*)

SET5: *PRE\_COND\_MO*

⊃ *POST\_AContD.NAME* = *POST\_AContD.NAME*  $\bowtie$  *PRE\_AContD.NAME*  
 == TRUE;

(\* IDc と name の組についてみたとき、処理後のコンテナは処理前のコンテナのサブセット (図5のRC5 ∪ RC7は NULL) \*)

SET6: *PRE\_COND\_MO*

⊃ *POST\_OrdD.NAME* = *POST\_OrdD.NAME*  $\bowtie$  *PRE\_AContD.NAME*  
 == TRUE;

(\* IDc と name の組についてみたとき、出庫指示書は処理前のコンテナのサブセット (図5のRC6 ∪ RC7は NULL) \*)

図6 第2レベルでの仕様の例一続きー

Fig. 6 ASL text for level 2—Continued.

得たかを示す。

また、*PRE\_AContD*、*POST\_AContD*、*POST\_OrdD* の間で満たされるべき性質の記述では、以下のことに着目した。それらのデータベースは、そ



それぞれ属性  $IDc$  と  $name$  を含むが、各データベース中の各エントリの属性値の組  $\{IDc, name\}$  は、図5の RC 1, ..., RC 7 のいずれかの部分に属する(実際には、RC 4, RC 5, RC 6, RC 7 に属するような  $\{IDc, name\}$  の組は存在しない)。そこで、図5の各 RC 1, ..., RC 7 ごとに、その  $\{IDc, name\}$  を持つエントリが満たすべき性質を、公理 SET 1~SET 6(図6)にそれぞれ記述している。

### 3.3.2 詳細化例

第三レベルで、未処理本数  $Qty$  や処理済コンテナデータベース  $NewContD$  など、新しい状態成分関数として導入し、それらを使って「出庫処理」を以下のようなアルゴリズムで実現した。

まず  $NewContD$  を NULL,  $Qty$  を  $Req$  の(属性)出庫数量  $qtyq$  の値に初期化し、以下を、 $Qty$  が取り出したエントリ中の在庫数量より大きい間、繰り返す。 $AContD$  のエントリを一つずつ取り出し、取り出したエントリ  $entry$  中の品名が  $Req$  の品名と異なる場合は、 $entry$  をそのまま  $NewContD$  に追加し、そうでない場合には、 $Qty$  を  $entry$  中の在庫数量  $qtyc$  分だけ減じ、かつ  $entry$  からその在庫を全て出庫するような指示(エントリ)を出庫指示書  $OrdD$  に追加する。

以上を繰り返して  $Qty$  が  $entry$  の在庫数量以下になった場合、 $entry$  から  $Qty$  分だけ出庫する指示を  $OrdD$  に追加し、 $entry$  の在庫数量から  $Qty$  を減じたエントリを  $NewContD$  に追加して、繰り返しを終了する。その後、 $NewContD$  と残りの  $AContD$  を結合したものを、新しコンテナデータベースとして、処理を終了する。

上述の処理を行うために、幾つかの新しい遷移を導入し、その動作内容を記述した(記述の一部を図7に示す)。遷移  $Open\_AContD$  は初期化を行う。遷移  $Inc\_AContD$ (または遷移  $Delete\_AContD\_and\_Make\_OrdD$ )では、 $AContD$  からの一つのエントリの削除(操作関数  $Delete$  を使う)と、そのエントリの  $NewContD$  または  $OrdD$  への追加(操作関数  $Insert$  を使う)を行う。遷移  $Modify\_AContD\_and\_Make\_OrdD$  の動作内容は遷移  $Delete\_AContD\_and\_Make\_OrdD$

$OrdD$  とほぼ同じであるが、遷移の実行で  $OrdD$  へ追加するエントリの出庫数量に書かれる内容が  $Qty$  の値である点異なる。遷移  $Union\_AContD$  では、 $AContD$  と  $NewContD$  との Union をとったものを新しい  $AContD$  とする。

最後に、導入した各遷移の実行側(図8参照)を指定

```

define 'IDR'      := ' $\pi(IDr)Req(s)$ ';
define 'ADDR'     := ' $\pi(addr)Req(s)$ ';
define 'NAME'     := ' $\pi(name)Req(s)$ ';
define 'DATE'     := ' $\pi(date)P2(Delete(AContD(s)))$ ';
define 'IDC'      := ' $\pi(IDc)P2(Delete(AContD(s)))$ ';
define 'QNTY'     := ' $\pi(qtyc)P2(Delete(AContD(s)))$ ';
define 'EMPTY'    := ' $\pi(Search\_key(QTYList(s), \pi(IDc)P2(Delete(AContD(s))))$ 
                      =  $\pi(qtyc)P2(Delete(AContD(s)))$ ';

(** 以下、各遷移の動作内容。
  遷移の実行によって状態成分の値が
  変わらないことなどの記述は、省略 **)
AContD(Open_AContD(s)) == Open_AContD(s);
NewContD(Open_AContD(s)) == NULL;
Qty(Proc_Open_AContD(s)) ==  $\pi(qtyq)Req(s)$ ;
(* 遷移 Open_AContD の動作内容 (代入定義) *)
¬ (AContD(s) = NULL)
  ⊃ AContD(Delete_AContD_and_Make_OrdD(s))
  == P1(Delete(AContD(s))) == TRUE;
¬ (AContD(s) = NULL)
  ⊃ OrdD(Delete_AContD_and_Make_OrdD(s))
  == Insert(OrdD(s), [IDR, ADDR, NAME, IDC, QNTY, EMPTY])
  == TRUE;
¬ (AContD(s) = NULL)
  ⊃ Qty(Delete_AContD_and_Make_OrdD(s))
  == Qty(s) - QNTY == TRUE;
(* 遷移 Delete_AContD_and_Make_OrdD の動作内容
  (性質記述) *)
¬ (AContD(s) = NULL)
  ⊃ AContD(Inc_AContD(s)) = P1(Delete(AContD(s)))
  == TRUE;
¬ (AContD(s) = NULL)
  ⊃ NewContD(Inc_AContD(s))
  == Insert(NewContD(s), P2(Delete(AContD(s))))
  == TRUE;
(* 遷移 Inc_AContD の動作内容 (性質記述) *)
¬ (AContD(s) = NULL)
  ⊃ AContD(Modify_AContD_and_Make_OrdD(s))
  == P1(Delete(AContD(s))) == TRUE;
¬ (AContD(s) = NULL)
  ⊃ OrdD(Modify_AContD_and_Make_OrdD(s))
  == Insert(OrdD(s), [IDR, ADDR, NAME, IDC, Qty(s), FALSE])
  == TRUE;
¬ (AContD(s) = NULL)
  ⊃ NewContD(Modify_AContD_and_Make_OrdD(s))
  == Insert(NewContD(s), [IDC, DATE, NAME, QNTY - Qty(s)])
  == TRUE;
(* 遷移 Modify_AContD_and_Make_OrdD の動作内容
  (性質記述) *)
AContD(Union_AContD(s)) == NewContD(s) ∪ AContD(s);
(* 遷移 Union_AContD の動作内容 (代入定義) *)

```

図7 第3レベルの遷移の動作内容の記述(一部)

Fig. 7 Requirements for each transition in level 3.

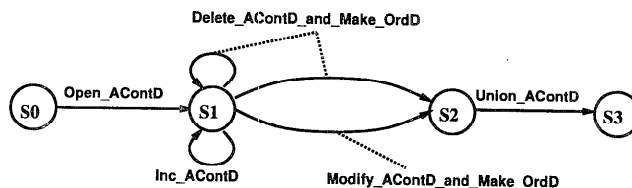


図8 第3レベルの状態図(一部)

Fig. 8 State diagram of level 3.

```

Prc_Make_Order(s) == Making_Order(Prc_Open_AContD(s));
Making_Order(s) ==
  if ( $\pi_{\{name\}}P2(Delete(AContD(s))) = \pi_{\{name\}}Req(s)$ )
  then modcontainer(s)
  else Making_Order(Inc_AContD(s));
modcontainer(s) ==
  if ( $Qty(s) > \pi_{\{qty\}}P2(Delete(AContD(s)))$ )
  then Making_Order(Delete_AContD_and_Make_OrdD(s))
  else terminate(s);
terminate(s) ==
  if ( $Qty(s) = \pi_{\{qty\}}P2(Delete(AContD(s)))$ )
  then union(Delete_AContD_and_Make_OrdD(s))
  else union(Modify_AContD_and_Make_OrdD(s));
union(s) == Union_ACont(s);

```

図9 第3レベルの遷移の実行指定(一部)

Fig.9 Execution order of transitions in level 3 and their execution conditions.

した、その記述を図9に示す(図中の Making\_Order, modcontainer, terminate, union は、それぞれループ関数である)。

## 4. 正しさの証明

### 4.1 正しい詳細化の定義

項の組  $\langle \xi, \eta \rangle$  がテキスト  $t$  のもとで定理として成り立つのは、 $\xi, \eta$  に許される任意の代入  $\rho$  に対して  $\rho\xi \equiv \rho\eta$  が成り立つこと、すなわちテキスト  $t$  上の帰納的定理であることとし、これを  $\xi \sim_t \eta$  と表すことにする<sup>4)</sup>。

上位のテキストで公理として表された各等式  $\xi == \eta$  に対して、下位のテキスト  $t$  の公理系のもとで定理  $\xi \sim_t \eta$  が成り立つとき、詳細化は正しいとする<sup>11)</sup>。厳密には下位の公理が無矛盾であることも必要であるが、今回は議論しない<sup>\*</sup>。

### 4.2 証明法の概略と検証支援系

$\xi \sim_t \eta$  を示すには、項書換え、場合分け、整数上の決定問題への帰着、補題の利用、帰納法などを用いる<sup>11)</sup>。これらの証明法自体は健全である。

今回のスタイル制限を満たす ASM 型記述に対しては、整数上の論理式の恒真性判定手続きを利用した簡明な証明手続きにより、証明を行える<sup>8)</sup>。その証明手続きは、次のような方法である。まず、下位のテキストの公理系のもとで定理となることを示すべき上位のテキストの公理(等式)、下位レベルの各遷移の動作内容、それらの実行制御の公理、構造的帰納法で用いる不変表明、および基本関数・述語に関する補題(それらは等式として書かれる)の変数・状態成分関数などの値を代入して得られるインスタンスから、構造的帰納法の

各段階に対応する論理式を作る(4.3.1項で例を示す)。次に、その論理式が、整数の加減算・比較演算および論理演算は通常の解釈、それ以外の関数・述語は自由解釈としたときに、真となるかどうかを調べる。論理式の真偽は、全ての変数が全称記号  $\forall$  で束縛された冠頭標準形のプレスブルガー文の真偽判定手続き<sup>7)</sup>により調べる。

以上の方法で証明ができた場合には、検証者が用いた基本関数・述語の補題が正しいという前提のもとで、詳細化の正しさが保証される。

この証明法に基づく我々の検証支援系では、検証者は構造的帰納法で用いる不変表明、基本関数・述語に関する補題、および補題の各変数への代入値を支援系に与えるだけで良く、論理式の合成やその真偽判定などは支援系が自動で行う。本支援系は、帰納法のどの段階が既に証明できているかの自動管理なども行う。

また、本証明法では真偽を判定する論理式が大きくなる場合があるため、証明の高速化のために、上述の形のプレスブルガー文の真偽判定ルーチンに、高速化の工夫をしている<sup>12)</sup>。

### 4.3 証明の実際

我々の検証支援系を用いて、3.3.2項で述べた「出庫処理」の詳細化の正しさを一部証明した。ここでは、その証明の概略と、証明にかかった手間などについて述べる。

#### 4.3.1 証明例1

3.3.2項で述べた「出庫処理」の実現が、要求性質のうち公理(等式)O1(図3)で表された性質を満たすことを証明した。証明で用いた各状態への不変表明を図10に、基本関数・述語に関する補題のうち主なものを図11にまとめる。

証明ではまず、図8の状態  $S_1$  と  $S_3$  に対して不変表明を与えた。このうち、 $S_1$  に対する表明  $Inv(s, S_0)$  を検証者が考案した。ここで、 $S_0$  は図8の状態  $S_0$  を表

$\langle S_1 \text{ に与えた表明 } (Inv(s, S_0)) \rangle$

```

( OrdD(s) = NULL
   $\vee (\neg (OrdD(s) = NULL)$ 
     $\supset EachPr_{(qty \geq qtyr \wedge qtyr > 0)} (\pi_{\{IDc, qtyr\}} OrdD(s))$ 
       $\bowtie (\pi_{\{IDc, qty\}} \sigma_{\{name = \pi_{\{name\}} Req(S_0)\}} AContD(S_0))$ 
    )
   $\wedge AContD(s) \bowtie AContD(S_0) = AContD(s)$ 
   $\wedge \neg (AContD(s) = NULL)$ 
   $\wedge Qty(s) > 0$ 
   $\wedge Qty(s) \leq AllOp_{+}(qty) (\sigma_{\{name = \pi_{\{name\}} Req(S_0)\}} AContD(s))$ 
   $\wedge Req(s) = Req(S_0)$ 
 $\langle S_3 \text{ に与えた表明 } \rangle$ 

```

```

EachPr(qty <= qtyr & qtyr > 0) (  $\pi_{\{IDc, qtyr\}} OrdD(s)$  )
 $\bowtie (\pi_{\{IDc, qty\}} \sigma_{\{name = \pi_{\{name\}} Req(S_0)\}} AContD(S_0))$  )

```

図10 性質 O1 の証明に用いた表明

Fig.10 Assertions used for proof of the axiom O1.

\* プログラムレベルでは、性質記述ではなく代入定義の形の公理で書かれるので、その形より無矛盾は保証される。従って、各レベルの詳細化が正しければ、各レベルの記述も無矛盾となる。

```

prim1:  $\neg (db_3 = NULL)$ 
 $\supset \{ ( \text{EachPr}_{\{qty \geq qtyr \wedge qtyr > 0\}} ( \pi_{\{IDc, qtyr\}} db_1 )$ 
 $\quad \bowtie ( \pi_{\{IDc, qtyr\}} \sigma_{\{name=name1\}} db_2 ) )$ 
 $\quad \wedge 0 < i_1 \leq \pi_{\{qtyr\}} P2(Delete(db_3))$ 
 $\quad \wedge i_2 = \pi_{\{IDc\}} P2(Delete(db_3))$ 
 $\quad \wedge name_1 = \pi_{\{name\}} P2(Delete(db_3))$ 
 $\quad \wedge db_3 \bowtie db_2 = db_3$ 
 $\quad )$ 
 $\supset \text{EachPr}_{\{qty \geq qtyr \wedge qtyr > 0\}} ($ 
 $\quad ( \pi_{\{IDc, qtyr\}} Insert(db_1, \{i_3, addr_1, name_1, i_2, i_1, b_1\})$ 
 $\quad \bowtie ( \pi_{\{IDc, qtyr\}} \sigma_{\{name=name1\}} db_2 ) ) )$ 
prim2:  $\neg (db_1 = NULL)$ 
 $\supset \{ (db_1 \bowtie db_2) = db_1$ 
 $\supset (P1(Delete(db_1)) \bowtie db_2) = P1(Delete(db_1)) \}$ 
prim3:  $\neg (db_1 = NULL)$ 
 $\supset \{ ( \pi_{\{name\}} P2(Delete(db_1)) = name1$ 
 $\quad \wedge 0 \leq \pi_{\{qtyr\}} P2(Delete(db_1))$ 
 $\quad \quad < AllOp_{\{qtyr\}} (\sigma_{\{name=name1\}} db_1)$ 
 $\quad )$ 
 $\supset \neg (P1(Delete(db_1)) = NULL) \}$ 

```

図 11 公理 O1 の証明に用いた基本関数に関する補題(一部)

Fig. 11 Theorems for primitive functions/predicates used for proof of the axiom O1.

す定数である。S<sub>3</sub> に対する表明は O1 の結論部に対応する式であり、検証支援系が自動的に状態 S<sub>3</sub> に与えた。

このもとで証明では、PRE-COND-MO (O1 の前提部に対応する式) の変数  $s$  に定数 S0 を代入した式が成り立つという仮定のもとで、S<sub>0</sub> から遷移の実行を始めて S<sub>3</sub> に到達したときに S<sub>3</sub> の表明が成り立つことと、S<sub>1</sub> の不変表明が実際に不変式として成り立つことを示す。

S<sub>1</sub> の不変表明はいくつかの条件の論理積からなり、「この状態での(作成途中の)出庫指示書  $OrdD(s)$  は、NULL であるか、NULL でなければ O1 の要求性質を満たしていること(条件 1)、かつ、この状態でのコンテナデータベース  $AContD(s)$  は S<sub>0</sub> でのデータベース  $AContD(S0)$  の部分集合であること(条件 2)、かつ、 $AContD(s)$  は空でないこと(条件 3)、かつ、この状態での未出庫本数  $Qty(s)$  は正(条件 4)、かつ、 $Qty(s)$  の値は  $AContD(s)$  中の出庫品目の総在庫量以下であること(条件 5)、かつ、この状態での出庫依頼書  $Req(s)$  は S<sub>0</sub> での出庫依頼書  $Req(S0)$  と同じであること(条件 6)」を表す。

検証者は、この不変表明を次のようにして考案していった。通常、不変表明の考案の際には、証明に用いる基本関数の補題の考案も行っていく。

まず、条件 1 を導入した。S<sub>1</sub> で常に条件 1 が成り立つことを示すことができれば、S<sub>1</sub> での遷移の繰り返し実行を抜けて状態 S<sub>3</sub> に到達したとき、作成された出庫指示書が O1 の要求性質を満たすことを示すことができる(このことはプログラムの設計者の意図にも合っていると思われた)。このために、「データベース  $db_1$  と  $db_2$  が(O1 の)要求性質を満たしていれば、 $db_1$  に要求

性質を満たすような( $db_3$  中の)エントリを加えたデータベースと  $db_2$  は、要求性質を満たす」ことを表す補題 prim 1(図 11)を導入した。

補題 prim 1 を使うためには、状態 S<sub>1</sub> からの遷移  $Delete-AContD-and-Make-OrdD$  の実行で出庫指示書  $OrdD$  に加えられるエントリ entry が、要求性質を満たすことを示さなければならない。ここで、不変表明の条件 1 では  $OrdD(s)$  と  $AContD(S0)$  との関係を書いているが、遷移  $Delete-AContD-and-Make-OrdD$  の実行で出庫指示書に加えられるエントリは、 $AContD(s)$  から得られるものである。従って、 $AContD(s)$  中の各エントリが  $AContD(S0)$  中の各エントリと同じものでなければ、entry が要求性質を満たすことは示せない。このために、不変表明に条件 2 を追加した。

以下、条件 3 は補題 prim 1 などの前提条件(補題の引数となるデータベースが NULL でないこと)を満たすために導入した。条件 4 は、状態 S<sub>1</sub> での繰り返しを抜けて遷移  $Modify-AContD-and-Make-OrdD$  を実行するときに  $OrdD(s)$  に追加されるエントリについて、その属性「出庫本数」の値が正であることを示すために導入した。条件 5 は、この条件と、S<sub>1</sub> で遷移が繰り返し実行されるための条件( $Qty$  の値が、 $AContD$  から削除したエントリ中の在庫数量より大きい)などから、条件 3 が保存することを示すために導入した。条件 6 は、条件 2 と同様に、S<sub>1</sub> での出庫依頼書  $Req(s)$  が出庫処理前の出庫依頼書  $Req(S0)$  と同じものであることを示すために導入した。

証明で用いた他の基本関数・述語に関する補題のうち主なものとしては「データベース  $db_1$  が  $db_2$  の部分集合なら、 $db_1$  から一つのエントリを削除したデータベースも  $db_2$  の部分集合であること」(prim 2. 不変表明の条件 2 の成立を示すために使う)、「品目  $name1$  のデータベース  $db_1$  中の在庫の総和が正、かつ、エントリ  $P2(Delete(db_1))$  中の品目が  $name1$  で、その在庫数量(非負とする)が  $db_1$  中の在庫の総和よりも小さければ、 $P1(Delete(db_1))$  は空ではないこと」(prim 3. 条件 3 の成立を導く)などがある。

不変表明  $Inv(s, S0)$  に加えて、検証者が上述のような基本関数・述語に関する補題と、それらの補題の変数への代入値(例えば補題 prim 1 の変数  $db_1$  には  $OrdD(S)$ 、 $db_2$  には  $AContD(S0)$ 、 $db_3$  には  $AContD(S)$  を与えた)を検証支援系に与えると、支援系は、例えばパス「状態 S<sub>1</sub> → (遷移  $Inc-AContD$ ) → S<sub>1</sub>」(帰納法の帰納段階に対応する)に対しては、PRE-COND-MO の変数  $s$  に定数 S0 を代入した式



*Disjoint* であり、かつ  $ACondD(S_0)$  は、 $AContD(s)$  と  $NewContD(s)$  との和、 $AContD(s)$ 、 $OrdD(s)$  を、それぞれ含むこと」などを与えた。

また、証明に用いた基本関数の補題は「データベース  $db_1$  (NULL でないとする) と  $db_2$  が *Disjoint* ならば、 $db_1$  から一つのエントリを削除したデータベース  $P1(Delete(db_1))$  と、 $db_2$  にそのエントリ  $P2(Delete(db_1))$  を追加したデータベースが *Disjoint* であること」(prim 4)、「データベース  $db_1$  が  $db_2$ 、 $db_3$  をそれぞれ含むなら、 $db_3$  に  $db_2$  中の一つのエントリ (のコピー) を追加したデータベースも、 $db_1$  に含まれること」(prim 5)、「データベース  $db_1$ 、 $db_2$ 、 $db_3$  が {IDc, name} の値について互いに *Disjoint* ならば、 $db_1$  中のエントリ  $P2(Delete(db_1))$  の (在庫) 数量を分割して得られる二つのエントリ (それらは分割前のエントリと同一の {IDc, name} を持つ) の一方を  $db_2$  に、他方を  $db_3$  に追加して得られるデータベースをそれぞれ  $\alpha$ 、 $\beta$  としたとき、同一の {IDc, name} を持つような  $db_1$ 、 $\alpha$ 、 $\beta$  中の任意のエントリ間に対して、 $db_1$  のエントリ中の (在庫) 数量 =  $\alpha$  のエントリ中の数量 +  $\beta$  のエントリ中の数量であること」(prim 6) などである。

以上のような基本関数・述語に関する補題や不変表明を検証支援系に与えて証明を行った (証明にかかった手間などを表 1 に示す)。この例ではプレスブルガー文の長さは最大 430 程度で、その真偽は 0.2 秒程度で判定できた。

#### 4.3.3. 証明例 3

この詳細化例のように、状態遷移の繰り返し実行がある場合は、停止性を保証する必要がある。このためには、例えば、繰り返しによって常に減少し、かつ 0 以上の値をとるものを見つけ、そのことを証明すれば良い。この例では、コンテナデータベースのサイズを与える関数を考えて、証明を行った (証明にかかった手間などを表 1 に示す)。その関数の値は繰り返しの間、Delete により確実に一つ減る。また、0 以下にならないことも、コンテナデータベースが NULL にならないこと (証明例 1 で、 $S_i$  で常に成り立つことが既に証明されている) より保証できる。

#### 4.4 考 察

文献 6) では、上位レベルで公理として表された四つの等式に対し、それらが下位レベルの公理系のもとで定理となることの証明 (および停止性の証明) に数週間の労力をかけているが、今回は前節の各証明を合計 1 週間程度で行うことができた。文献 6) の場合とは証明した性質は異なる (公理も 1 対 1 に対応しているわけではない) が、今回の方がより複雑な性質を証明してい

ることから、本論文で述べたような方法によって、従来より証明を短期間で効率的に行えと考えられる。

この主な理由としては、(1) 記述スタイルの制限により文献 8) の証明法が使えるため、証明法 (項書換え等の適用順) を考える必要がなくなり、検証者が証明の道筋 (どのような不変表明や基本関数の補題を用いて証明を行うか) の考案に専念できたこと、(2) 高速なプレスブルガー文真偽判定ルーチンにより、大きな論理式の真偽を短時間で証明できるようになったこと、(3) 真偽を判定する論理式を手で入れなくても済むようになったこと (支援系が不変表明などから論理式を自動生成する) や、証明過程の自動管理機能があることが挙げられる。

今回は、出庫処理の正しさの証明において、一部の公理 (15 個中 4 個) しか証明していないが、証明しなかった公理は今回証明した公理と同様な性質の記述がほとんどであることから、それらについても今回と同程度かそれ以下の時間で証明できると思われる。また、一部の証明であっても、それによりプログラムの信頼性を高めることができるとと思われる。

4.3.1 項で示したように、通常、検証作業で用いる不変表明や基本関数に関する補題は、試行錯誤により決定していく。今回の作業時間のほとんどは、それらの考案時間である。今回はプログラムの設計者と検証者は別であったが、設計者自身が検証を行う場合には、不変表明の考案などにかかる時間はより少なくて済むと考えられる。

本論文で用いた証明支援系では、検証者が用いた基本関数に関する補題が正しいという仮定のもとで詳細化が正しいことを保証する。補題が正しいかどうかを議論したいときには、さらにプリミティブなレベルで証明を行えば良いが、プログラムの正しさを議論するのに本質的でないところは、証明は省くのが現実的であろう。

#### 5. あとがき

本論文では、制限されたスタイルのもとで抽象的順序機械型プログラムの仕様を記述し、プログラムの正しさの証明を高速・半自動で行う方法を示し、例題として在庫管理プログラムの仕様記述や詳細化の正しさの証明を実際に行った。その実験により、今回のように、プログラム (の入出力データ) が満たすべき関係がかなり複雑な場合であっても、本手法により自然に処理の要求性質を記述でき、その実現の正しさの証明も短期間に行えることが分かった。

今後の課題の一つとしては、証明作業の支援の工夫

等が挙げられる。そのための具体的な方法としては、証明が失敗したときに、式の構造が限られていることを利用して、どの部分が失敗の原因か指摘する機能の追加等が考えられる。

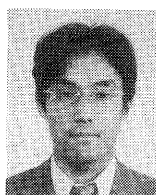
## 参 考 文 献

- 1) 山崎利治：共通問題によるプログラム設計技法解説—他，情報処理，Vol. 25, No. 9, pp. 934-962 (1984)。
- 2) 山崎利治：共通問題によるプログラム設計技法解説(その2)—他，情報処理，Vol. 25, No. 11, pp. 1219-1268 (1984)。
- 3) 二村良彦，雨宮真人，山崎利治，淵 一博：新しいプログラミングパラダイムによる共通問題の設計—他，情報処理，Vol. 26, No. 5, pp. 458-520 (1985)。
- 4) 嵩 忠雄，谷口健一，杉山裕二，関 浩之：代数的言語 ASL/\*—意味定義を中心に—，信学論(D)，Vol. J 69-D, No. 7, pp. 1066-1074 (1986)。
- 5) 岡野浩三，北道淳司，東野輝夫，谷口健一：順序機械型プログラムの階層的設計法と在庫管理プログラムの開発例，信学論 D-I, Vol. J 76-D-I, No. 7, pp. 354-363 (1993)。
- 6) 岡野浩三，北道淳司，東野輝夫，谷口健一：代数的言語 ASL で記述した在庫管理プログラムとその正しさの証明，信学报報，SS 90-29 (1990)。
- 7) Cooper, D. C.: Theorem Proving in Arithmetic without Multiplication, *Machine Intelligence*, No. 7, pp. 91-99 (1972)。
- 8) 森岡澄夫，岡野浩三，東野輝夫，谷口健一：整数上の論理式の恒真性判定手続きを利用した順序機械型仕様記述の詳細化の正しさの半自動証明，日本ソフトウェア科学会第 11 回大会論文集，pp. 361-364 (Nov. 1994)。
- 9) 大蘆雅弘，杉山裕二，谷口健一：代数的言語 ASL における抽象的順序機械型プログラムとその処理系，信学論(D 1)，Vol. J 73-D-I, No. 12, pp. 971-978 (1990)。
- 10) Kanellakis, P. C.: Elements of Relational Database Theory, Ch. 17, *Handbook of Theoretical Computer Science Vol. B—Formal Models and Semantics*, ed. Leeuwen, J. V., Elsevier Sci. Pub. (1990)。
- 11) 東野輝夫，関 浩之，谷口健一：代数的仕様から関数型プログラムの導出とその実行，情報処理，Vol. 29, No. 8, pp. 881-896 (1988)。
- 12) Kitamichi, J., Morioka, S., Higashino, T. and Taniguchi, K.: Automatic Correctness Proof of the Implementation of Synchronous Sequential Circuits using an Algebraic Approach, *Proc. 2nd Int. Conference on Theorem Provers in Circuit Design (TPCD 94)* (Kropf, T. and Kumar, R. eds.), Vol. 901 of Lecture Notes in

Computer Science, pp. 165-184, Springer Verlag (1995)。

(平成 6 年 8 月 17 日受付)

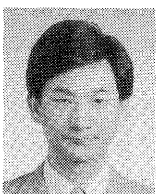
(平成 7 年 3 月 13 日採録)



森岡 澄夫 (学生会員)

昭和 43 年生。平成 4 年大阪大学基礎工学部情報工学科卒業。平成 6 年同大学院博士前期課程修了。現在、同大学院博士後期課程在学中。代数的手法を用いたソフトウェア・ハー

ドウェアの形式的検証などに興味を持つ。平成 6 年度より日本学術振興会特別研究員。日本ソフトウェア科学会会員。



岡野 浩三 (正会員)

昭和 42 年生。平成 2 年大阪大学基礎工学部情報工学科卒業。平成 5 年同大学院博士後期課程中退。同年同大学基礎工学部情報工学科助手。現在に至る。博士(工学)。代数的手法

によるソフトウェア設計開発法，分散システム等の研究に従事。電子情報通信学会会員。



東野 輝夫 (正会員)

昭和 31 年生。昭和 54 年大阪大学基礎工学部情報工学科卒業。昭和 59 年同大学院博士課程修了。工学博士。同年同大学助手。平成 2 年，6 年モ

ントリオール大学客員研究員。平成 3 年大阪大学基礎工学部情報工学科助教授。現在に至る。分散システム，通信プロトコル等の研究に従事。電子情報通信学会，IEEE-CS，ACM 各会員。



谷口 健一 (正会員)

昭和 17 年生。昭和 40 年大阪大学工学部電子工学科卒業。昭和 45 年同大学院基礎工学研究科博士課程修了。工学博士。同年同大学基礎工学部助手。現在，同情報工学科教授。

計算理論，ソフトウェアやハードウェアの仕様記述・実現・検証の代数的手法および支援システム，関数型言語の処理系，分散システムや通信プロトコルの設計・検証法などに関する研究に従事。