ソフトウェア基礎論　37－23
プログラミング言語　27－23
（1990. 12. 14）

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

岡野　浩三　北道　淳司　東野　輝夫　谷口　健一

大阪大学基礎工学部情報工学科

あらまし

情報処理学会誌においてプログラム作成法の比較研究のための共通問題として取り上げられた酒屋在庫管理プログラムを我々の研究グループが設計した代数的言語ASLを用いて記述した．そのプログラムはASLの部分言語である抽象的順序機械型言語ASL/ASMを用いて階層的に記述されており，ASLシステムのコンパイラを用いて翻訳･実行できる．

また，出庫依頼書に対する処理について，満たすべき要求をASLで記述し，作成したプログラムがその要求を満たしていることをASLシステムの検証支援機能を用いて証明した．

本報告では，そのプログラムおよび検証した性質，証明の仕方，証明に要した手間等について説明されている．

A Stock Management Program written in Algebraic Language ASL and Its Correctness Proof

Kouzou OKANO　Junji KITAMICHI　Teruo HIGASHINO　Ken'ichi TANIGUCHI

Department of Information and Computer Sciences, Faculty of Engineering Science, Osaka University

Abstract

A *Stock Management Problem for a wholesale liquor dealer* was introduced as a common problem for program design. In this paper, an ASL program for the stock management is presented. ASL is an algebraic language designed by our group. The program is described hierarchically in Abstract Sequential Machine (ASM) style and can be compiled to an executable one by ASL system.

Requirements of a procedure which deals with a shipping request is described in ASL. We proved that the program satisfies the requirements by using a verification support system for ASL. The proof method and the working load for the proof are also explained.

1. まえがき

代数的言語は,意味定義が厳密であり抽象度の高いレベルから低いレベルまで同じ言語で記述することができ,記述の正しさの証明を形式的に行なえる等の利点をもつ.

更に任意のレベルを自然に記述でき,かつ実行プログラムの効率が悪くないこと,正しさの証明が計算機の支援のもとで行えることなどが望まれる.

これらの点を考慮し筆者らのグループでは代数的言語ASL[3]を定め,記述支援系,検証支援系,実行系からなるASLプログラム開発システムをつくり,代数的手法の有用性について調べてきた.

実際にASLシステムを使って今までに,クイックソートの仕様記述,プログラムへの詳細化,正しさの証明[6],[7],[8],スクリーンエディタの仕様記述[4],プログラムへの詳細化などを行ってきた.

しかしながらASLシステムを用いて仕様からプログラムを製作し,さらにはプログラムの正しさを証明するといった経験が少ない.

そこで,在庫管理問題を例題として,ASLシステムを実際に用いてプログラムを作り,その正しさの証明を一部行った.以下**2.**では在庫管理問題について簡単に説明し,**3.**ではASLによる記述例について述べる.

更に**4.**ではプログラムの正しさの証明について説明する.

2. 在庫管理問題

在庫管理問題は文献"情報処理,**26**,5"[1]の特集にプログラム開発の共通問題として提供された問題である.概略は次の通りである.

(1). (どの商品を何個出庫するかを記載した)"出庫依頼書"を受け付けたとき,在庫があれば,(その商品をどのコンテナから何個ずつ出庫するかを記載した)"出庫指示書"を出力する.在庫がなければ"出庫できない"旨を出力する.

(2). (そのコンテナにどの商品を各々何個格納しているかを記載した)"積荷表"を受け付けたとき,過去に在庫不足で出庫できなかった"出庫依頼書"の中で,その入荷により出庫できるものが生じれば,それらに対してそれぞれ"出庫指示書"を出力する.

(3). "終了"を受け付けたとき,在庫情報等の内部情報を適当なファイルにセーブし,プログラム再起動時にはこのファイルからデータをロードし,前回終了前の状態に復帰させる.

(4). 以上"終了"を受け付けるまで(1),および(2)を繰り返す.

なお,(3)における終了時,再起動時の処理の仕様は我々が追加したものである.

以上の要求には

- 在庫情報をどのように格納するのか
- 過去に出庫出来なかった出庫依頼書の中からどのような方針で依頼書を選択するのか
- 出庫するときどのような方針でコンテナを選択するのか

等の要求は含まれていない.このようなことは,データ構造の決定などと一緒に必要な時点で決定していけばよい.

3. 抽象的順序機械型プログラムによるプログラム

上述の在庫管理を行うプログラムをASLの部分クラスである抽象的順序機械型言語ASL/ASM[2]を用いて作成する.

ASMプログラムでは,抽象状態を表すデータタイプをあらたに導入し,その抽象状態を引数にする関数として,状態遷移関数と状態成分関数を定義する.状態成分関数は抽象状態での各状態成分の値を表し,状態遷移関数は各状態成分の値を変化させる.

ASMプログラムは各状態遷移関数で各状態成分関数の値がどのように変わるかの記述と,状態遷移関数をどのような順で適用し目的の出力値を求めるかという実行指定の記述からなる.

階層的設計のアプローチで詳細化を行う.その概略を**3.1**に述べる.そして実際のプログラム例を**3.2**以下に示す.

3.1 階層的設計の概略

図1, 2, 3は今の問題に対する階層的設計の概念図を表している.四角は処理,縦線は実行指定に対応する.実行指定は,上に描かれている処理を下に描かれている処理を用いて定義する.

例えば,在庫管理は,"初期化","出庫依頼書入力に対する処理","積荷表入力に対する処理","データ格納後終了する処理"を用い,これらを繰り返し選択実行指定することによって定義される.

更に,"出庫依頼書入力に対する処理"は(一つの出庫依頼書に対する)"出庫処理","指示書出力処理","在庫不足処理"を用いて実行指定が定義される.

ここで"初期化","データ格納後終了する処理","指示書出力処理","在庫不足処理"等太い四角で囲っている処理は,基本関数を用いて処理内容が定義されている処理である.一方,2重の四角で囲まれた"出庫処理"では今の時点(第1レベル)では出庫処理は,どういう性質の処理であるかという処理の性質をASLで書いておく.

それ以外の処理例えば"出庫依頼書入力に対する処理","積荷表入力に対する処理"は理解,記述しやすさのために導入した名前(状態遷移関数)である.

ある処理を,いくつかのより基本的な処理(内容が定義されている,あるいは,性質が記述されている)を用いて実現するときには,その記述は一般に次の三つの部分からなる.

- 処理内容が定義できている処理の記述
- 処理の性質が記されている処理の記述

- 上記二つの処理をどのように実行指定するかの定義の記述

ここで, 性質が記されている処理の記述に関しては, 目的に応じて次の 2 通りの立場が考えられる.

(1). その処理に対する要求仕様を書く. つまりこれを満たすように作るのであればどのように作っても良いと言う立場で記述する. 考えられるすべての性質を洩れなく記述する必要がある.

(2). プログラムの正しさを議論するために, 検証したい性質あるいは検証に必要な性質だけを記述する.

上述の (2)の検証したい性質は一般に (1) の要求仕様の部分集合となることが多い.

次のステップでは処理の性質が記されている処理を同様の手法で詳細化していく. 今の例では出庫処理を詳細化することになる.

以上に述べたような詳細化を繰り返し, すべて基本関数を使って定義できる処理を用いて実現できたときに詳細化は終了する.

今回この在庫管理プログラムの例では 3 段で記述した.

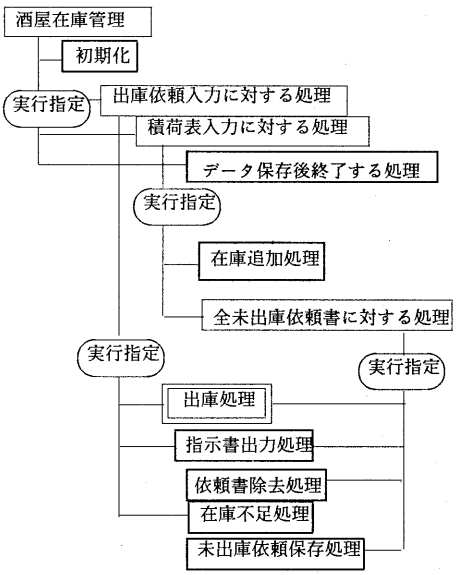

図 1: 詳細化の概念図 第 1 レベル

図 1 では, 在庫管理は,“出庫処理”,“指示書出力処理”,“依頼書除去処理” などを用いて定義されている.

出庫処理は外部から今受け付けた出庫依頼書あるいは過去に出庫できなかったので保存しておいた出庫依頼書のいずれにかかわらず, 着目している出庫依頼書に対して在庫があるとき, これに対する出庫指示書を作成し, 在庫情報を更新する処理である.

出庫処理の記述は, 出庫処理に対する要求仕様の立場で記述されている. すなわちどのようにして指示書を作成するか (どのコンテナから優先して出庫するのか, そのコンテナから出す本数はどのように決めるか等) にはふれずに, どのような指示書を作るべきか, 内部情報がどのように変わるべきかが記述されている.

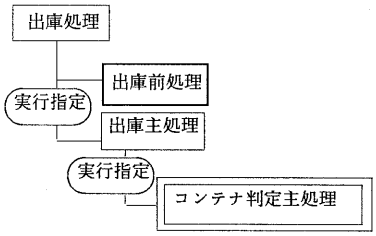

図 2: 詳細化の概念図 第 2 レベル

図 2 では, 出庫処理は,“出庫前処理”,“コンテナ判定主処理” を用いて実現されている. 出庫前処理の後, コンテナリストを先頭からたどりながら, 要求された本数がそろうまで, コンテナ判定主処理を繰り返し実行する.

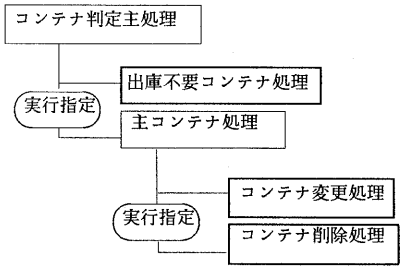

図 3: 詳細化の概念図 第 3 レベル

図 3 では, コンテナ判定主処理は “出庫不要コンテナ処理”, “コンテナ変更処理”,“コンテナ削除処理” を用いて実現されている. それらは, 基本関数を用いて定義するので, 詳細化はこれで終了する.

以上をまとめると, 記述全体は図 4 のように表せる.

便宜上, 図 4 中の各記述を text1a, text1b, ... ,text3b と呼ぶことにする.

在庫管理の ASM プログラムは上記の階層的設計で得られた記述の

- 処理内容が定義されている処理の記述
  図 4 の text1b, 2b, 3b

- 実行指定の定義の記述
  図 4 の text1a, 2a, 3a

の部分だけを集めたものとなる.

単にプログラムを作るだけならば, 処理の性質 (text1c,2c) などを書く必要はない.

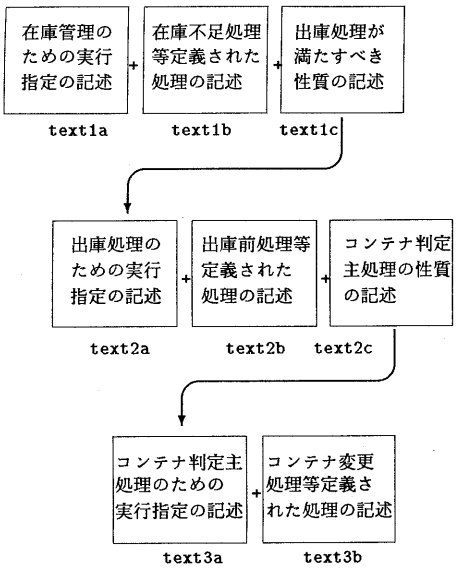

図 4: テキストの構成

3.2 第 1 レベルのプログラム

図 5 は,“出庫処理”,“指示書出力処理”,“在庫不足処理” 等をどのように組み合わせて在庫管理を実行指定しているかを記した部分 (text1a) である.

例えば, 未出庫依頼書リストに対し出庫できるものをすべて出庫するための処理 “全未出庫依頼書に対する処理” は,“未出庫依頼書に対する処理” を繰り返し行うことと記述されている.

更に,“未出庫依頼に対する処理” は未出庫依頼書リスト中の着目している出庫依頼書に対し, もし出庫可能 ならば “出庫処理”, “依頼書の削除処理”,“指示書出力処理” をこの順で実行し, 出庫可能でなければ “未出庫依頼保存処理” を実行することと記述されている.

ここで, 出庫可能は関数 “在庫有り (在庫品名数量表(S),pr_3(依頼書),pr_4(依頼書))’; ” の略記で, 該当依頼書に対して在庫があるかどうかを在庫品名数量表をもちいて判定する関数である. 在庫品名数量表というのは, 品名とその品名に対する (全在庫における) 数量の表である. 各コンテナを順次検索して判定するよりも, この表を用いて調べる方が, 効率的である.

```
define ’出庫可能’ := ’在庫有り
(在庫品名数量表(S),pr_3(依頼書),pr_4(依頼書))’;
define ’番号’:=’Get_i_num (未出庫依頼書リスト(S))’;
define ’宛先’:=’Get_i_adr (未出庫依頼書リスト(S))’;
define ’品名’:=’Get_i_name(未出庫依頼書リスト(S))’;
define ’本数’:=’Get_i_suu (未出庫依頼書リスト(S))’;
酒屋() == 出力ファイル( 初期化 処理( stdin ) ) ;
S 処理( input_sq ) ==
  if      Get_cmnd( input_sq ) = ’quit’
  then    ( S SAVE )
  else if Get_cmnd( input_sq ) = ’import’
  then    ( S 積荷表入力に対する処理
              ( Get_tumini( Tail(input_sq) ) )
              処理( Tail( Tail(input_sq) ) ) )
  else if Get_cmnd( input_sq ) = ’export’
  then    ( S 依頼書入力に対する処理
              ( Get_irai( Tail(input_sq) ) )
              処理( Tail( Tail(input_sq) ) ) )
  else    ( S 処理( Tail( input_sq ) ) ) ;
(* 依頼書に対する処理 *)
S 依頼書入力に対する処理( 依頼書 ) ==
  if   出庫可能
  then ( S 出庫処理( 依頼書 ) 指示書出力処理 )
  else ( S 在庫不足処理( 依頼書 ) ) ;
(* 入庫時の処理 *)
S 積荷表入力に対する処理( 積荷表 ) ==
  S 在庫追加処理( 積荷表 )
    全未出庫依頼書に対する処理 ;
(* 全未出庫分の処理の実現 *)
S 全未出庫依頼書に対する処理   ==
  if   End_list_i( 未出庫依頼書リスト( S ) )
  then S
  else ( S 未出庫依頼に対する処理(
                  [番号, 宛先, 品名, 本数] )
         全未出庫依頼書に対する処理 ) ;
(* 未出庫依頼に対する処理の実現 *)
S 未出庫依頼に対する処理 ( 依頼書 ) ==
  if     出庫可能
  then   ( S 出庫処理 ( 依頼書 )
           依頼書の削除処理  指示書出力処理 )
  else   ( S 未出庫依頼保存処理 ) ;
```

図 5: 在庫管理のための実行指定 text 1a

“依頼書入力に対する処理” は, 状態と依頼書を引数にとる関数で, ここでは, mixfix の形で書かれている. 他の処理も mixfix の形で書かれているが,ASL ではこのように, 関数など表現式の構文を文脈自由文法で, 自由に指定できる.

図 6 は, 基本関数を用いて定義されている処理について記述されている部分 (text1b) である.

例えば, 在庫不足処理後の状態成分 未出庫依頼書リストは依頼書が追加されるということや, 在庫不足処理後の出力ファイルは連絡文字列化 (依頼書) の値が追加されることが記述されている. 関数連絡文字列化は引数の依頼書が出庫できない旨の連絡を示す文字列を返す関数である.

また, 図 5 の記述では, 未出庫依頼書に関する全データは依頼書のリストの形で保持し, 例えば “全未出庫依頼書に対する処理” では先頭から検索するというように記述している (もとの要求にはないが, このように決めて記述した).

```
(* 初期化 *)
コンテナリスト( 初期化 ) ==
  Load_zaiko( ’stock.save’ ) ;
未出庫依頼書リスト( 初期化 ) ==
  Load_irai( ’waitprc.save’ ) ;
在庫品名数量表( 初期化 ) ==
  Load_naiyou( ’table.save’ ) ;
出力ファイル  ( 初期化 ) == stdout ;
指示書        ( 初期化 ) ==
  [ 0, snill, snill, Create_list_c(anill) ] ;
(* データ格納後終了する処理 *)
在庫_save_file    ( S SAVE ) ==
  Save_zaiko( ’stock.save’, コンテナリスト(S) ) ;
依頼書_save_file  ( S SAVE ) ==
  Save_irai( ’waitprc.save’, 未出庫依頼書リスト(S) );
内容_save_file    ( S SAVE ) ==
  Save_naiyou( ’table.save’, 在庫品名数量表(S) );
(* 指示書出力処理 *)
出力ファイル( S 指示書出力処理 ) == Out_string
  ( 出力ファイル(S), 指示書文字列化(指示書(S)) ) ;
(* 在庫不足処理 *)
未出庫依頼書リスト( S 在庫不足処理( 依頼書 ) ) ==
  Add_last_i  ( 未出庫依頼書リスト( S ) , 依頼書 ) ;
出力ファイル      ( S 在庫不足処理( 依頼書 ) ) ==
  Out_string( 出力ファイル(S), 連絡文字列化(依頼書) );
(* 在庫追加処理 *)
コンテナリスト    ( S 在庫追加処理(積荷表) ) ==
  Add_last_z( コンテナリスト(S), 品名単一化(積荷表) ) ;
```

```
未出庫依頼書リスト( S 在庫追加処理(積荷表) ) ==
  Set_Top_i( 未出庫依頼書リスト( S ) ) ;
在庫品名数量表    ( S 在庫追加処理(積荷表) ) ==
  品目更新( 在庫品名数量表( S ) , pr_3( 積荷表 ) ) ;
(* 依頼書の削除処理 *)
未出庫依頼書リスト( S 依頼書の削除処理 ) ==
  Delete_i( 未出庫依頼書リスト( S ) ) ;
(* 未出庫依頼保存処理 *)
未出庫依頼書リスト( S 未出庫依頼保存処理 ) ==
  Inc_i( 未出庫依頼書リスト( S ) ) ;
   (* 内容の変わらない状態成分は省略している *)
```

図 6: 初期化, 在庫不足処理等の定義 text 1b

3.3 基本関数

このプログラムを作るにあたり, 基本関数として仮定している関数の一覧を図 7, 8, 9 に示す. ここでは, ポインタ込みの抽象リストを基本データ構造として用いている.

ある *item*(整数, 文字列等) の組を cell と呼ぶことにする. *item* 自体が, 抽象リストであることも許す. ここで考えている抽象リストとは, cell の (いわゆる) リストと cell をさすポインタの 2 項組である.

抽象リストに関する基本関数は次の通りである.

Create_list(anill)	空の抽象リスト
Set_Top(list)	list の内蔵ポインタを先頭にセットした抽象リスト
Inc(list)	list の内蔵ポインタを一つ進めた抽象リスト
Delete(list)	list の内蔵ポインタのさす cell を取り除き, ポインタを一つ進めた抽象リスト
Add_last(list, cell)	list の最後に cell を追加した抽象リスト
Get_*item*(list)	list の内蔵ポインタのさす cell 中の該当 *item* の値
Cng_*item*(list, *value*)	list の内蔵ポインタのさす cell の中の該当 *item* の値を値 *value* に変更し, ポインタを一つ進めた抽象リスト
End_list(list)	list の内蔵ポインタが list の最後を指しているかどうかの判定

プログラム中ではこの抽象リストをコンテナリスト, 未出庫依頼書リスト, コンテナの内容リスト, 出庫指示書の指示リストに用いている. そこで, 上記の関数を, 各データ構造の分だけ用意してある. 図 7 にこれを示す.

抽象リストはコンテナリスト

Get_z_int(コンテナリスト)	*item* はコンテナ番号
Get_z_list(コンテナリスト)	*item* はコンテナの内蔵品リスト
Set_Top_z(コンテナリスト)	
Delete_z(コンテナリスト)	
Inc_z(コンテナリスト)	
Add_last_z(コンテナリスト, セル)	
Cng_z_list(コンテナリスト, 内蔵品リスト)	*item* はコンテナの内蔵品リスト

抽象リストは未出庫依頼書リスト

Get_i_num(未出庫依頼書リスト)	*item* は依頼書番号
Get_i_adr(未出庫依頼書リスト)	*item* は依頼書の送り先
Get_i_name(未出庫依頼書リスト)	*item* は依頼書の品名
Get_i_suu(未出庫依頼書リスト)	*item* は依頼書の本数
Inc_i(未出庫依頼書リスト)	
Delete_i(未出庫依頼書リスト)	
Add_last_i(未出庫依頼書リスト, セル)	
Set_Top_i(未出庫依頼書リスト)	
End_list_i(未出庫依頼書リスト)	

抽象リストは指示リスト

Create_list_c(anill)	
Add_last_c(指示リスト, セル)	

抽象リストはコンテナ内容リスト

Create_list_n(anill)	

図 7: 抽象リストに関する基本関数

図 8 に入出力関係の基本関数を示す. コマンド, 積荷表, 依頼書の 3 つからなる系列を抽象系列と呼ぶことにする.

図 9 は, その他の基本関数の一覧である.

Get_cmnd(input)	抽象系列 input からコマンドを得る. 但し抽象系列の先頭がコマンドでなくてはいけない.
Get_tumini(input)	抽象系列 input から積荷表を得る. 但し抽象系列の先頭が積荷表でなくてはいけない.
Get_irai(input)	抽象系列 input からから依頼書を得る. 但し抽象系列の先頭が依頼書でなくてはいけない.
Tail(input)	抽象系列 input の tail(先頭を取り除いた残り) を得る
Out_string(出力ファイル, 文字列)	出力ファイルに文字列を出力する
Load_zaiko(ファイル)	コンテナリストの内容をファイルからロードする
Load_irai(ファイル)	未出庫依頼リストの内容をファイルからロードする
Load_naiyou(ファイル)	在庫品名数量表の内容をファイルからロードする
Save_zaiko(ファイル, コンテナリスト)	コンテナリストの内容をファイルへセーブする
Save_irai(ファイル, 未出庫依頼書リスト)	未出庫依頼書リストの内容をファイルへセーブドする
Save_naiyou(ファイル, 在庫品名数量表)	在庫品名数量表の内容をファイルへセーブする

図 8: 入出力基本関数

コンテナ変更(コンテナ内容, 品名, hon)	コンテナ内容の該当品の本数を hon だけ減らす
数量更新(在庫品名数量表, 品名 , hon)	在庫品名数量表の該当品の本数を hon だけ減らす
品目更新(在庫品名数量表, セル)	在庫品名数量表をセルのエントリーで変更追加する
品名単一化(積荷表)	積荷表の品名の同一項目を一つにまとめる
積荷本数(コンテナ, 品名)	コンテナの 該当品の本数を返す

カーゴ本数(コンテナ)	コンテナの総本数を返す
在庫有り (在庫品名数量表, 品名, hon)	在庫品名数量の該当品に hon 以上の本数があるかどうかの判定
指示書文字列化(指示書)	指示書の内容を文字列に変える
連絡文字列化(依頼書)	依頼書は出庫できない旨の通知の文字列を返す

図 9: その他の基本関数

基本関数は

(1). C 言語で直接実現した関数

(2). ASL/ASM コンパイラが提供する組込み関数, あるいはこれらを組み合わせて ASL で実現した関数

がある.

3.4 出庫処理の満たすべき性質

図 10 は出庫処理の満たすべき性質の記述 (text 1c) である.

今回は出庫処理の記述に関しては, 要求仕様, すなわちこれさえ満たせばどのように実現しても良いと言う立場で記述した.

満たすべき性質は "条件 imply 項1 == 項2" の形のいくつかの公理で記述する. ここで条件は, この出庫処理が実行される直前に各データがそれらを満たしているとして出庫処理を実現してもよいという条件に相当する. 今の場合は, 出庫依頼本数は正, コンテナリスト中のコンテナ番号はすべて異なる, コンテナリスト中の各コンテナの出庫依頼書品名の数量は負でない, 出庫依頼品名に対する在庫がある (コンテナリスト中の全コンテナに関する出庫品名の数量の総本数が出庫依頼本数以上であること, 出庫可と略記) の 4 つの論理積である (まとめて入力条件と略記).

例えば S1 は そのような条件が満たされているときに出庫処理を行った後の出庫指示書の番号は依頼書の番号に一致していること, すなわち出庫指示書が依頼書に対応するものであることを示している.

以下,S2 は送り先が一致すること,S3 は品名が一致することを表している.

S4 は指示書のコンテナの記述は一度に限る (同じコンテナに関する記述が 2 回以上現れない) ことを規定している.

S5 は指示書に書かれた (各々のコンテナから何本づつ出すかをあらわす) 本数の合計が出庫依頼書本数と一致することを表している.

S6 は指示書とコンテナの関係が健全であることを規定している. 指示書正当と言う関数は, 指示書に記載された各コンテナに対し, コンテナ正当という関数が真であることと定義されている. コンテナ正当という関数内で, 出庫本数は正で, かつ該当コンテナ中の (その品名の) 在庫本数を越えてはいけない, あるいは, 空きコンテナの指示があるときは実際にそのときに限ることが記されている.

以下, 同様に出庫処理後のコンテナリストがどうあるべきか, 在庫品名数量表がどうあるべきかについて記述してある (但し図 10 では省略).

```
define 出庫指示書 := '指示書( S 出庫処理( 依頼書 ) )' ;
define 出庫可      :=
'本数_在庫. 品名(Set_Top_z(コンテナリスト(S)),品名(依頼書))
   >= 本数( 依頼書 ) ' ;
define 前提条件   :=
 'Unique_id(Set_Top_z(コンテナリスト(S))) and
 '本数(依頼書)>0 and ( Memberof( id, コンテナリスト(S) )
 imply 本数_在庫. 番号. 品名( Set_Top_z(コンテナリスト(S)),
                              id, 品名(依頼書) ) >= 0 )' ;
define 入力条件   :='前提条件 and 出庫可' ;
  (* 指示書に関する性質 *)
S1: 入力条件 imply
  番号 (出庫指示書) = 番号 (依頼書) == TRUE ;
S2: 入力条件 imply
  送り先(出庫指示書) = 送り先(依頼書) == TRUE ;
S3: 入力条件 imply
  品名 (出庫指示書) = 品名 (依頼書) == TRUE ;
S4: 入力条件 imply Unique.id( 出庫指示書 )     == TRUE ;
S5: 入力条件 imply
  総本数_指示書(出庫指示書) = 本数 (依頼書) == TRUE ;
S6: 入力条件 imply
  指示書正当( 出庫指示書, コンテナリスト(S), 品名(依頼) )
        == TRUE ;
  (* コンテナリストの性質 *)
                              ⋮ (省略)
  (* 在庫品名数量表の性質 *)
                              ⋮ (省略)
  (*  補助関数  *)
指示書正当( 指示書, コンテナ list, 品名 ) ==
  全コンテナ正当( Set_Bottum( 指示list( 指示書 ) ),
                  コンテナ list, 品名 ) ;
全コンテナ正当( 指示list , コンテナ list , 品名 ) ==
  if   Empty_list( 指示list ) then TRUE
  else( コンテナ正当( Get_c_id( 指示list ),
          Get_c_suu( 指示list ), Get_c_bool( 指示list ),
          コンテナ list , 品名 )
 and 全コンテナ正当(Deque(指示list), コンテナ list, 品名) );
コンテナ正当( id , 本数 , bit , コンテナ list , 品名 ) ==
  0 < 本数 and
  本数 <= 本数_コンテナ.id.品名( コンテナ list , id , 品名 )
 and ( bit iff (本数_コンテナ.id(コンテナ list, id)=hon) );
```

図 10: 出庫処理の満たすべき性質 text 1c

なお, 図 10 で Unique_id(コンテナリスト (S)) は, 状態 Sでのポインタ以降のコンテナリスト中のコンテナ番号がすべて異なるとき真, そうでないとき偽を返す述語である. 本数_在庫. 品名 (コンテナリスト (S),*hin*) は状態 Sでのコンテナリストの現状ポインタ以降のコンテナすべてに対して品名 *hin* に対する数量の合計を返す関数である. 本数_在庫. 番号. 品名 (コンテナリスト (S),*id*,*hin*) は, 状態 Sでのコンテナリストにおける番号 *id* のコンテナ中の品名 *hin* の数量を返す関数である. Memberof(*id*, コンテナリスト) はコンテナリスト中に *id* をコンテナ番号としてもつコンテナがあるかどうかのブール値を返す. Set_Bottum は ポインタをリストの最後のセルにセットする関数であり, また Deque はリストの最後のセルを取り除く関数である. iff は *if and only if* すなわち恒等を表す基本関数である.

3.5 出庫処理の実現

図 11 は出庫処理の実現 (text 2a) を表している. ここでは, コンテナリストの先頭 (古い在庫) から順に調べるという方針を採用する. 出庫処理は出庫前処理を行った後, コンテナリストをたどりながらコンテナ判定主処理を用いて実現されている.

出庫前処理は処理内容がこのレベルで定義されている処理である.

今回, コンテナ判定主処理の記述に関しては, 出庫処理の性質の証明に必要な性質を書く. 出庫処理の正しさの証明を行う際に, 必要であると判断した性質を逐次記入していくことにする. 次のレベルではこのコンテナ判定主処理を処理内容が定義された処理だけを用いて実現する.

```
define ’該当品数’ :=
  ’積荷本数( Get_z_list(コンテナリスト(S)), hin ) ’ ;
(* 出庫処理の実現 *)
S 出庫処理 ( 依頼書 ) ==
  S 出庫前処理( 依頼書 )
    出庫主処理( 本数( 依頼書 ) , 品名( 依頼書 ) ) ;
(* 出庫主処理の実現 *)
S 出庫主処理( hon , hin ) ==
  if      hon <= 0
  then    S
  else    ( S コンテナ判定主処理( hon , hin )
              出庫主処理( hon - 該当品数 , hin ) ) ;
```

図 11: 出庫処理のための実行指定 text 2a

図 12 は出庫前処理の定義の記述 (text 2b) である. この時点で記述できる指示書の内容 (依頼書番号, 送り先, 品名および空の指示リスト) を定義している. また, コンテナリストのポインタを先頭にセットしている.

```
(* 出庫前処理 *)
コンテナリスト( S 出庫前処理( 依頼書 ) ) ==
  Set_Top_z( コンテナリスト( S ) ) ;
指示書          ( S 出庫前処理( 依頼書 ) ) ==
  [ 番号(依頼書), 送り先(依頼書), 品名(依頼書),
    Create_list_c(anill) ] ;
```

図 12: 出庫前処理の定義 text 2b

```
define ’該当品数’ :=
  ’積荷本数( Get_z_list(コンテナリスト(S)), hin )’;
define ’カーゴ内総数’ :=
  ’カーゴ本数( Get_z_list(コンテナリスト(S)) )’ ;
S コンテナ判定主処理( hon , hin ) ==
  if      該当品数 = 0
  then    ( S 出庫不要コンテナ処理 )
  else if 該当品数 >= hon
  then    ( S 出庫コンテナ処理( hon , hin) )
  else    ( S 出庫コンテナ処理( 該当品数 , hin ) ) ;
S 出庫コンテナ処理( hon , hin ) ==
  if      hon < カーゴ内総数
  then    ( S コンテナ変更処理( hin , hon ) )
  else    ( S コンテナ削除処理( hin ) ) ;
```

図 13: コンテナ判定主処理のための実行指定 text 3a

図 13 では, コンテナ判定主処理の実現 (text 3a) を表している.

この実現では, あと何本出庫すべきかが与えられて, 当該コンテナにその品目の在庫が 1 本以上あるかどうかを判定し, あるときは, その在庫量でまかなえるときは必要量, まかなえないときは全量を出庫するように定めている.

ここで,“出庫不要コンテナ処理” は当該コンテナに出庫対象となる品目がない, あるいは,0 本であるときの処理であり,“コンテナ変更処理” は当該コンテナから該当品を必要量出してなおかつ, コンテナが空にならないときに行われる処理である.

それに対して “コンテナ削除処理” はコンテナが空になるときの処理である.

これらの処理の内容は, 図 14 の text3b で定義されている.

```
define ’該当品数’ :=
  ’積荷本数( Get_z_list( コンテナリスト(S) ), hin )’ ;
define ’CargoID’    := ’Get_z_int(コンテナリスト(S) )’ ;
define ’CargoCONT’:= ’Get_z_list(コンテナリスト(S) )’ ;
(* コンテナ変更処理 list の要素で出庫できるものがある  *)
コンテナリスト( S コンテナ変更処理( hin, hon ) ) ==
  Cng_z_list(コンテナリスト(S),
             コンテナ変更( CargoCONT, hin, hon ) ) ;
指示書          ( S コンテナ変更処理( hin , hon ) ) ==
  [ 番号( 指示書(S) ) , 送り先( 指示書(S) ), hin ,
    Add_last_c(pr_4(指示書(S)), [CargoID,hon,FALSE]) ];
在庫品名数量表( S コンテナ変更処理( hin , hon ) ) ==
  数量更新( 在庫品名数量表(S), hin, hon ) ;
(* コンテナ削除処理 *)
コンテナリスト     ( S コンテナ削除処理( hin ) ) ==
  Delete_z( コンテナリスト( S ) ) ;
指示書          ( S コンテナ削除処理( hin ) ) ==
  [ 番号( 指示書(S) ) , 送り先( 指示書(S) ), hin,
    Add_last_c(pr_4(指示書(S)),[CargoID, 該当品数,TRUE]) ];
在庫品名数量表 ( S コンテナ削除処理( hin ) ) ==
  数量更新( 在庫品名数量表(S), hin, 該当品数 ) ;
(* 出庫不要コンテナ処理 *)
コンテナリスト ( S 出庫不要コンテナ処理 ) ==
  Inc_z( コンテナリスト(S) ) ;
```

図 14: コンテナ変更処理等の定義 text 3b

3.6 プログラムのサイズ

以上述べた text 1a, 1b...3b の各サイズを表 1 に示す. 表 1の中身は, 上段公理数, 下段行数を表す.

表 1: ASL テキストのサイズ

text1a	text1b	text1c
6 個	47 個	21 個
53 行	96 行	123 行
text2a	text2b	text2c
2 個	5 個	7 個
28 行	14 行	25 行
text3a	text3b	
2 個	15 個	
27 行	54 行	

プログラムは, 実行指定部と処理の定義部を合わせたもの (text1a,1b,2a,2b,3a,3b) である. 更には, 式の構文指定のための文法部も必要である. これらのサイズは次のとおりである.

- 文法部 書換え規則 140 個 280 行
- プログラムサイズ 公理 80 個 280 行 + 文法部 280 行
- 要求部サイズ 公理 35 個 220 行 + 文法部 280 行

(注) 表 1の text2c の部分では **4.**で述べる正しさの証明において “出庫処理” の性質を証明するのに使った “コンテナ判定主処理” に関する補題の量を載せている.

4. 正しさの証明

ここでは, 出庫処理のレベル 2 以降の実現が, 出庫処理の要求性質の正しい実現であるということを証明する. それには,

(1). text2a, text2b, text2c のもとで text1c の各式が成り立つことを示す

(2). text3a, text3b のもとで text2c の各式が成り立つことを示す

これらの証明を,検証支援系を用いて行う.

4.1 検証支援系の提供する機能

ASL システムの検証支援系[6],[7],[8]には次の機能が用意されている. (1) 項書換え機能, (2) 数学的恒真性判定機能, (3) 補題追加機能, (4) 場合分け管理機能.

項書換え機能は公理を左項から右項への書き換え規則と見なして,項を書き換えていくものである.

数学的恒真性判定機能は整数上の述語プレスブルガー文の恒真性を判定する機能である.

4.2 検証例

ここでは S5 の検証を例に,どのような方法で検証を行うかについて説明する.

4.2.1 証明の方法

出庫処理の実現は再帰関数を用いているので,証明には,コンテナ判定主処理の適用に関する帰納法を用いる. 以下便宜上次のように関数を読み替える.

$$S\text{出庫主処理}\ (hon, hin) \rightarrow f(S, hon, hin)$$
$$S\text{出庫前処理}\ (\text{依頼}) \rightarrow g(S, irai)$$
$$S\text{コンテナ判定主処理}\ (hon, hin) \rightarrow r(S, hon, hin)$$

すると出庫処理は次のように書ける.

$$S_0\text{出庫処理}(irai, hin) = f(g(S_0, irai), \text{本数}(irai), \text{品名}(irai))$$

更に関数 f は次のように書ける.

$$f(S, hon, hin) = \textbf{if}\ c(S, hon, hin)\ \textbf{then}\ p(S, hon, hin)\ \textbf{else}\ f(q(S, hon, hin))$$

但し

$$c(S, hon, hin) \equiv hon \leq 0$$
$$p(S, hon, hin) \equiv S$$
$$q(S, hon, hin) \equiv \langle r(S, hon, hin),\ hon - Hon_{hin}(S, hon, hin),\ hin \rangle$$

ここで $q(S, hon, hin)$ は,引数 S, hon, hin から右辺の 3 つ組を返す関数である.

また $Hon_{hin}(S, hon, hin)$ は該当品数すなわち状態 S におけるコンテナリストのポインタのさすコンテナ一つに対して品名 hin に関する本数を返す関数である.

関数 "出庫処理" の "入出力" に関する述語 Q を導入し,次のように定める.

$$Q(S_0, irai, S) \equiv \text{総本数}(\text{指示書}(S)) = \text{本数}(irai)$$

ここで "総本数" は関数 総本数_指示書の略記とする. これは作成中の指示書も含め,指示書に現れた本数をすべて足し合わせる関数である.

更に出庫処理の入力 (実行直前の状態 S_0 と出庫依頼書 $irai$) と f の引数 S, hon, hin との間の関係を表す述語 $R(S_0, irai, S, hon, hin)$ を次のように定める.

$$R(S_0, irai, S, hon, hin) \equiv$$
$$\textbf{if}\ \ hon > 0$$
$$\textbf{then}\ \text{総本数}(\text{指示書}(S)) = \text{本数}(irai) - hon$$
$$\textbf{else}\ \text{総本数}(\text{指示書}(S)) = \text{本数}(irai) \quad (\alpha)$$
$$\wedge\ \text{品名}(irai) = hin \quad (\beta)$$
$$\wedge\ hon > 0 \Rightarrow$$
$$\text{総本数}_{\text{品}}(\text{コンテナリスト}(S), hin) \geq hon \quad (\gamma)$$

但し総本数$_{\text{品}}$(コンテナリスト(S), hin) は状態 S でのコンテナリストの現状ポインタ以降のコンテナすべてに対して hin に対する数量の合計を返す関数つまり本数_在庫.品名 (コンテナリスト (S),hin) の略記である.

入力条件のうち出庫可の条件と,前提条件の中の本数 (依頼書) > 0 を S5 の証明に使う. それら二つの条件の論理積を $PreQ(S,$ 依頼書) と略記する.

さて, S5 の証明を次のような手順で行う.

(1). $PreQ(S_0, irai)$ を仮定する.

(2). $R(S_0, irai, g(S_0, irai), \text{本数}(irai), \text{品名}(irai))$ が真になることを示す.

(3). $R(S_0, irai, S, hon, hin)$ を仮定する.

(4). $c(S, hon, hin)$ の値が定まることを示す.

(5). (a) $\neg c(S, hon, hin)$ を仮定する.

(b) $q(S, hon, hin)$ の値が定まることを示す.

(c) $R(S_0, irai, q(S, hon, hin))$ が真になることを示す.

(6). (a) 今度は $c(S, hon, hin)$ を仮定する.

(b) $p(S, hin, hon)$ の値が定まることを示す.

(c) $Q(S_0, irai, p(S, hon, hin))$ が真になることを示す.

(7). $c(q^n(g(S_0, irai), \text{本数}(irai), \text{品名}(irai)))$

== TRUE を満たす非負整数 n が存在すること,すなわち,再帰がいつかは停止することを示す.

以上より以下のことを結論する.

「関数出庫処理の再帰が停止し，指示書の総本数等の関数の値が求まり，かつ

$PreQ(\ S,$ 依頼書 $)\ \Rightarrow$
$\quad Q(S,$依頼書,
$\quad\quad$出庫処理 $(S,$本数 (依頼書), 品名 (依頼書)))

が真である.」

S5 の入力条件 $\Rightarrow\ PreQ(S,$ 依頼書) であるので，以上のことより，S5 のいわゆる Total Correctness の証明を行ったことになる (S5 の式は，総本数_指示書 (出庫指示書) の値が定まり，かつ入力値である本数 (依頼書) と等しいことを要求している).

(2)が帰納法の基底段階に相当し,(5)が帰納段階に相当する.(6)は最終段階である.

この証明で用いたコンテナ判定主処理すなわち $r(\ldots)$ の性質は次の二つである.

総本数 (指示書 ($r(\ S,\ hon,\ hin\ )$)) =
$\quad$ **if** $\quad Hon_{hin}(\ S,\ hon,\ hin\ ) \geq hon \qquad (C_1)$
$\quad$ **then** 総本数 (指示書 (S)) $+ hon$
$\quad$ **else** 総本数 (指示書 (S)) $+ Hon_{hin}(\ S,\ hon,\ hin\ )$

総本数$_{品}$(コンテナリスト ($r(\ S,\ hon,\ hin\ )$), $hin\ )$ =
$\quad$ 総本数$_{品}$(コンテナリスト $(S),\ hin\ )$
$\quad - Hon_{hin}(\ S,\ hon,\ hin\ ) \qquad (C_2)$

実際に証明支援機能を用いるのは上述の (2), (5), (6)の部分である. 値が定まるか，停止性は保証されているかといった Totall Correctness の議論で用いる補題を使うために必要な性質は不変式 R中に入っている (今の Rではγ).

Totall Correctness の議論で用いた補題は例えば次のようなものである.

任意の正整数 N, 整数列 $n_1, n_2, \ldots$ に対して

$$\forall i\ [\ \exists \ell(i)\ :\ n_i + n_{i+1} + n_{i+2} + \cdots + n_{i+\ell(i)} > 0]$$
$$\Rightarrow \exists k\ :\ N - n_1 - n_2 - \cdots - n_k < 0$$

$hon > 0 \wedge$ 総本数$_{品}$(コンテナリスト $(S),\ hin\ ) > 0 \Rightarrow$

$$\exists n\ :\ \sum_{i=0}^{n} Hon_{hin}(\ q^i(S,\ hon,\ hin)\ ) > 0$$

4.2.2 検証の実際

ここでは検証支援機能をどのように使うかについて述べる. 例として証明手順中の (5)すなわち帰納段階を取り上げる. その中の $R(\ S_0,\ irai\ ,\ q(\ S,\ hon,\ hin\ )\ )$ の第 1 項 αが真であることは次のようにして証明する.

この項は実際には次の形をしている.

if $\quad hon\ -\ Hon_{hin}(S,\ hon,\ hin) > 0$
$\quad$ **then** 総本数 (指示書 ($r(S,\ hon,\ hin\ )$)) =
$\quad\quad$本数 $(irai)\ -\ (hon\ -\ Hon_{hin}(S,\ hon,\ hin))$
$\quad$ **else** 総本数 (指示書 $(r(S,\ hon,\ hin\ )$)) =
$\quad\quad$本数 $(irai)$

(1). まず text2a, text2b, text2c(C_1, C_2は入っている) のもとで検証支援機能を起動する.

(2). 次に補題追加機能を用いて，前述の証明手順中の (1)での仮定 $PreQ(S_0,\ irai)$ と (3)での仮定の $R(S_0,\ irai,\ S,\ hon,\ hin)$ を追加する.

(3). 更に$\neg\ c(\ S,\ hon,\ hin\ )$ も仮定する.

(4). 証明項αをロードする.

(5). 項αの if 文の条件式を，場合分け管理機能を用いて，真偽の両方を確認することにする.

(6). 条件式が真のときは次のようになる.

(7). 条件式==真と言う補題は自動的に追加される.

(8). 証明項を項書換え機能を用いて書き換えていく. αは総本数 (指示書 ($r(S,\ hon,\ hin\ )$)) =
本数 $(irai)\ -\ (hon\ -\ Hon_{hin}(S,\ hon,\ hin))$ となる

(9). 項書換えによって得られた数式に対して,text 中の公理や追加された補題 (前提条件等) を使い数学的恒真性判定機能を用いて恒真であることを確認する.

(10). 条件式が偽の場合,条件式==真と言う補題は自動的に取り除かれる.

(11). このあと条件式が偽の場合も同様に行う.

恒真性を判定するときに基本関数の性質をあらかじめ，公理の形で追加する必要がある場合もある.

S5 を証明するために使った基本関数の性質は次のようなものである.

総本数 ($[\ id,\ adr,\ hin,\ Create\_list\_n(anill)\ ]$)
$\quad =\ \ 0\ ==$ **TRUE**

これは，指示書中の指示リストが空なら総本数は 0 に等しいことを表している.

また，コンテナ判定主処理に関する性質 (C_1) の証明では次の基本関数の性質等を使用した.

総本数 $([\ id_1,\ adr,\ hin,\ Add\_last\_c(cs,\ [id_2, hon, bit])\ ])$
$\quad =\ \ $ 総本数 $([\ id_1,\ adr,\ hin,\ cs\ ])\ +\ hon\ ==$ **TRUE**

これは，指示書にどのコンテナから何本という情報が加わるとその本数分だけ総本数が増えることを表している.

4.3 検証作業量

ここにあげているのは出庫処理の性質のうち指示書に関する性質 (図 10 の (S1)~(S6)) を証明したときの，検証支援系のコマンド操作回数に関するデータである.

表 2は,S5 の証明に用いた検証支援系コマンド操作回数をまとめたものである. 1 行目は S5 が text2a, 2b および text2c(2 つの補題 $(C_1), (C_2)$ を含む) の上で成り立つことを証明するのに要した回数である. 2 行目は 2 つの補題 (C_1),

表 2: S5 の証明に用いた検証支援系コマンド操作回数

項書換え	補題追加	場合わけ	恒真性判定	その他	合計
9	21	8	12	15	65
18	12	18	12	6	66

表 3: S3 の証明に用いた検証支援系コマンド操作回数

項書換え	補題追加	場合わけ	恒真性判定	その他	合計
4	5	0	0	7	16
10	1	9	0	2	22

表 4: S4 の証明に用いた検証支援系コマンド操作回数

項書換え	補題追加	場合わけ	恒真性判定	その他	合計
3	7	0	7	10	27
33	7	30	15	6	91

表 5: S6 の証明に用いた検証支援系コマンド操作回数

項書換え	補題追加	場合わけ	恒真性判定	その他	合計
17	17	11	6	10	61
13	7	10	4	6	40

(C_2) が text3a,3b の上で成り立つことを証明するのに要した回数である.

S3 の証明では text2c へ補題を 1 個追加した. S4 の証明では text2c へ補題を 3 個追加した. S6 の証明では text2c へ補題を 1 個追加した.

表 3, 表 4, 表 5の 1 行目はそれぞれ S3, S4, S6 の証明に要したコマンド操作回数であり, 2 行目はその補題の証明に要した回数である.

S1, S2 は S3 とほぼ同様の回数と思われる. また S3 等の証明で関数が値をもつか, 再帰が停止するかなどの議論は S5 のそれと同様なので, S5 以外は Partial Correctness でのみ証明した. 明らかに同一の証明になる項の証明も省いている.

5. 終りに

ASL プログラム開発システムによるプログラム開発と, プログラムの正しさの証明について述べた.

このプログラムは効率を考えて作成されている. 例えば, 次のような点があげられる.

(1). 出庫可能かどうか判断するときに,(本来の意味である text1c 中の出庫可の定義にしたがって) コンテナリストを全部見て調べることをせずに, プログラムで用意した在庫品名数量表を見て判断するようにしている.

(2). 未出庫依頼リストやコンテナリストに対する処理において, 不必要な検索をしたり, 新しいリストを作るなどという操作をしない.

しかし, そのために, プログラムの正しさの証明では, (1) の判断で正しいのかの証明が余分に必要になるし, またリストについて (2)で述べたような操作を許すなら検証がより簡単になる可能性もある.

なお, 出庫処理の S1~S6 中の入力条件のうち外部入力に関するもの以外のものは, 外部入力に対する一定の条件 (出庫依頼本数が正, 積荷表のコンテナ番号が現在在庫中のコンテナ番号と異なる, 積荷表の各品目の本数が 0 本以上で, かつ積荷表全体では正, 等) のもとで, 出庫処理を行う状態では満たされているということを示す必要がある.

もちろん, 証明は検証者が必要と思うものだけを行なえばよい. 一部の証明であっても, それにより, プログラムの信頼性は非常に向上する. 今回の経験から,ASL を用いれば, 証明が可能であり, この程度の検証であれば作業量もそれほど多くないことが分かった.

今回は出庫処理の要求性質のみを記述して, 出庫処理の実現が, その要求に対する正しい実現であることの証明を一部行ったが, より上位レベルの要求記述, 例えば全未出庫依頼書の処理の満たすべき性質 (例えば未出庫依頼書をどの順番で優先して処理を行うかを指定しないような要求) をどのように記述するか, それに対してどのように証明を行うか, 等についても今後検討したいと考えている.

文献

[1] 二村, 雨宮, 山崎, 淵:"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", 情報処理,**26**, 5, pp.458-459(1985-05).

[2] 大蘆, 杉山, 谷口:"代数的言語 ASL における抽象的順序序機械型プログラムとその処理系", 信学論 (D1),**J73-D-I**,12(平 2-12) 掲載予定.

[3] 嵩, 谷口, 杉山, 関:"代数的言語 ASL/* –意味定義を中心に–", 信学論 (D), **J69-D**, 7, pp.1066-1075(昭 61-07).

[4] 大蘆, 谷口:"スクリーンエディタの仕様記述と抽象的順序序機械型プログラム", 信学技報,**SS**89-24, pp.55-64(1989-11).

[5] 東野, 関, 谷口:"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, **29**, 8, pp.881-896(昭 63-08).

[6] 日高:"代数的言語 ASL で記述されたプログラムの正しさの証明", 大阪大学大学院 基礎工学研究科 修士論文 (1990-02).

[7] 日高, 東野, 谷口:"ASL システムを用いたプログラム開発と証明書作成", 平 1 秋季信学全大, SD-7-1, pp. 6·248-6·249 (1989-09).

[8] 日高, 東野, 谷口:"代数的言語 ASL で記述されたプログラムの正しさの証明と証明書作成支援", 信学技報,**SS**89-21, pp. 45-54 (1989-11).