プログラミング－言語・基礎・実践－研究報告　9－4
（1992．10．30）

仕様記述言語ＣＳＰによるプロセス合成とＡｄａ並列処理プログラム開発の経験と評価

辻ヶ堂　信　　疋田　輝雄　　銀林　純
白鷗大学　　明治大学　　富士通

並列処理ソフトウェアシステムの仕様を形式的に記述するための言語として，CSP やLOTOS が試用されている．並列処理システムの開発方法として，CSP による形式的な仕様記述とプロセス計算を用いる仕様の検討を行い，次に，仕様からAda プログラムを開発するという，二段階からなる方法が考えられる．本論文はこの開発方法の，各段階における利点と問題点を明らかにすることを目的とする．このため，ある程度の規模と複雑さをもつ実例として，自動販売機の組込みソフトウェアの仕様を CSPで記述し検討する．次にその約95行からなる仕様記述を人手で書き直すことによって，約 500行からなる Adaプログラムを得る．この経験から得られた結論として，CSP による仕様記述の有効性，問題点，改善の方向，Ada プログラムとの整合性について述べる．

Experience and Evaluation of Specification and Process Calculation in CSP and Concurrent Program Implementation in Ada

Makoto Tsujigado　　Teruo Hikita　　Jun Ginbayashi
Hakuhoh Univ　　Meiji Univ　　Fujitsu Ltd

Specification languages CSP and LOTOS are recently beginning in use for the formal specification of parallel software systems. In this paper we study a conceivable development method of parallel systems, in which one first gives the formal specifications in CSP and manipulates them in process algebra, and, as the second step, develops Ada programs from the specifications by hand. The purpose of this paper is to clarify the advantages and disadvantages of this two-step method.

For this, we describe in CSP an embedded software for a vending machine, as an example with moderate size and complexity. Then by using process algebra, we prove the absence of the system deadlock. Lastly the specifications of 95 lines are translated to Ada programs of 500 lines.

As conclusions gained in this experience, we state the effectiveness and demerits of CSP, suggest its improvement, and describe the conformance and differences of CSP with Ada.

1 はじめに

並列処理システムの仕様を記述するための言語として，CSPやLOTOSが開発され，試用されている．CSP [4] は Hoareによって設計された．またLOTOS [7] は通信やOSI(Open Systems Interconnection) のための国際規格として開発された．これら二つはともに，プロセスの記述を基本とし，プロセスはイベントからなるという点で本質的には似ている．イベントとは複数のプロセスの参加する単一の事象で，同期と通信の役割を果す．

並列処理のプログラムを設計するのに，ほかに，Buhr[2]による構造グラフを利用する方法がある．イベントに着目する点では CSPも構造グラフも同じであるが，グラフを段階的に精細化していくのは人間の直観ないし発想を必要とするのに対して，CSP による表記はプロセスの代数的な処理が可能であり，仕様と設計の検討やデッドロックがないことの確認を論理的・代数的に行なえるという利点がある．

本論文の目的は，CSP による形式的な仕様記述およびプロセス計算を用いての仕様の検討と，その仕様から Adaプログラムを開発するという，二段階からなる並列処理システムの開発方法の，各段階における利点と問題点を明らかにすることである．なお，最終的な Ada プログラムの掲載は紙面の制約から省略する．

2 並列処理のための仕様記述言語CSP

2.1 CSP とは

CSP(Communicating Sequential Processes) に関して，1978年の最初の論文[3]はプログラム言語ないし計算モデルというべきもので，1985年の文献[4]とは内容が異なり，本論文では後者を指すものとする．

CSP はプロセスとイベントを基本概念とする．LOTOS と同じく，Milnerの並列計算モデル CCS(Calculus of Communicating Systems)の影響を受けている[8]．

並列処理の仕様記述の単位はプロセスであり，プロセスはイベント(event) からなる．プロセスを並置することにより，並列処理システムが構成される．一つのプロセスの動作の内容は逐次的なものであるが，外部の状況で選択されるプロセス選択肢や，非決定的に選択されるプロセス選択肢を持つプロセスもある．イベントは，同じ名前のイベントを持つ総てのプロセスの参加を要求するもので，プロセスの外部で観察でき，これ以上は分割できない単一の動作である．イベントは２個以上のプロセスの間の同期と通信に用いる．イベント名は英小文字，プロセス名は大文字で表記する．

イベントからプロセスを組み立てるため，イベントの選択｜，プロセスの並置‖などがある．演算子；は二つのプロセスの逐次的な実行を表す．繰返しはループまたは再帰によって表す．なお次の並置されたプロセスの合成プロセスで，$\mu X.\phi(X)$は $X=\phi(X)$ となるプロセスを表す．

$$P \| Q = (e1 \rightarrow P \mid e2 \rightarrow P) \| (e1 \rightarrow Q)$$
$$= (e1 \rightarrow (P \| Q) \mid e2 \rightarrow (P \| Q))$$
$$= \mu X.(e1 \rightarrow X \mid e2 \rightarrow X)$$

2.2 チャネルによる通信

CSP はプロセス間の通信をチャネルを介して行う．チャネルはプロセスの外部にある．チャネルｃからの値vの入力はc?vで表し，cへの出力はc!vとする．これらを総称して c.vと書く．これらもイベントである．

例． P=(c1?v→P｜c2!w→P)

2.3 並列処理以外の機能

逐次的な制御の機能としては，プロセスに局所的な変数とそれへの代入，論理式による分岐，ループが用意されている．変数は英小文字で表す．if b then P else Q にあたる分岐は次のように記述する．

P≮b≯Q

ここでｂは論理式，ＰとＱはプロセスである．

2.4 追加機能

CSP は，並列処理の記述に焦点を絞っているので，通常の逐次処理における種々の言語機能は最小限であるといってよい．それゆえ，欠けている機能の追加をある程度行う必要がある．本論文の仕様記述では次のものを追加した．

(1) 変数ｘが列挙型のとき，x(i)はその ｉ番目の値を表す．

(2) 列挙型の紙幣および硬貨 x(後出)の金額を示すために，v(x)と書くことにする．たとえば，v(Y1000)は千円札の値 1000 を表し，v(HALF) は 50 を表す．

(3) システムの外部(ユーザ)との通信手段として，論理型の大域的変数を導入する．例えば，第ｉ種の商品の排出されたことを，dispenseg(i)=true によって示す．この変数はプロセス間の通信・同期のためには使用しない．

3 課題とする自動販売機システム

自動販売機の組込みソフトウェアを設計し，それを Ada で実現する．貨幣制度，販売品およびその値段の変更は，自動販売機の設置時に，システムのパラメータの変更で済むような仕様を持つものとする．すなわち，

(1) 貨幣制度は，円，ドルなど，十進貨幣制度の国ならどこでも使えること．

(2) 販売品とその値段の変更が容易であること．

なお，並列処理の問題を検討するという目的から，次の条件をつける．

(3) 紙幣，硬貨の投入，商品の指定，釣銭（紙幣，硬貨）の排出，商品の排出は，並列動作が可能であること．（但し，論理的制約から，以上のすべてが並列に動くわけではない．）

ここではソフトウェアとしてこのシステムを実現す

```
select B/C/G/R/T: B
Y10000/Y5000/Y1000/:Y10000
select B/C/G/R/T: C
Y500/Y100/Y50/Y10/Y5/Y1/:Y100
select B/C/G/R/T: G
WATERMELON/CANDY/:CANDY
  dispense:CANDY  Change Total=  9870
  return:Y500  return:Y5000  return:Y100  return:Y1000
  return:Y100  return:Y1000  return:Y100  return:Y1000
  return:Y1000  return:Y50  return:Y10  return:Y10
select B/C/G/R/T: B
Y10000/Y5000/Y1000/:Y10000
select B/C/G/R/T: C
Y500/Y100/Y50/Y10/Y5/Y1/:Y500
select B/C/G/R/T: R
  Return All Your Money:Total Return= 10500
  return:Y10000  return:Y500
```

注　網かけ部分の文字はユーザの入力．また，紙幣と硬貨の排出は並列に行われている．

図 1　自動販売機の入出力

ることにする．端末上でのシステムの使用感は図１のようなものである．

システムからのselect B/C/G/R/T:というプロンプトに対してB(紙幣)と応答すると，システムからY10000/Y5000/Y1000/:というプロンプトが出る．これに対してY10000からY1000までのいずれかで答える．再びB/C/G/R/T:というプロンプトに対して C(硬貨)と応答すると，Y500/Y100/Y50/Y10/Y5/Y1/:のプロンプトがでる．これに対してはY500からY1のいずれかで答える．さらに，Gに対しては，watermelon,candyのいずれかで答える．この結果として，指定した商品と釣銭が排出される．Rは返金の指示，Tはシステムの終了の指示である．

4　仕様の設計

３節の課題に対して，CSP を用いた仕様記述を構成することによって，CSP の使用感を具体的に示し，またその長所と欠点を検討するための材料とする．

4.1 公開するデータ型

CSP にはデータ型を記述する機能は用意されていない．しかしこれは一般に仕様記述として最初になすべきことであり，ここで，システムの外部へ公開する型について定める．

用意する型は，紙幣の種類としてBILL_T，硬貨の種類としてCOIN_T，自動販売機の提供する商品の種類としてGOODS_T，商品の値段の型としてPRICE_LIST_Tとする．いずれも列挙型である．

例　BILL_T={Y10000,Y5000,Y1000}　　円紙幣の種類
　　COIN_T={Y500,Y100,Y50,Y10,Y5,Y1}円硬貨の種類
　　GOODS_T={watermelon,candy}　　商品の種類
　　PRICE_LIST_T={320,230}　　値段の単位は円

4.2 プロセスの概略設計

以下のようにシステムのモジュール化を行う．各モジュールは CSPのプロセスに対応する．

(1) 紙幣関係の考察(B_HDLR:Bill Handler)

ハードウェアの想定としては，投入された一枚の紙幣の種類を識別して，その値を通知し，将来の釣銭としての返却に備えて，紙幣の種類ごとに分類された貯蔵所に蓄える．紙幣の返却の要求があるときには，対応する種類の紙幣一枚を排出する．特定の紙幣の枚数が一定数以下になったら，アラームの表示をする．従って紙幣を扱うこのプロセスB_HDLRは，紙幣を読込む部分と，排出する部分より構成される．

(2) 硬貨関係の考察(C_HDLR:Coin Handler)

紙幣の場合と同様である．この部分は，上のB_HDLRと並列動作が可能であるものとする．

(3) 販売品関係の考察(その1) 商品識別ボタン関係(G_BTN:Goods Button):

ボタンにより指定された商品を識別し，「制御部分」へ通知する．商品識別ボタンが押された後は，商品を排出するか，投入金額を全額返却するまでは，紙幣も硬貨も投入されないように，B_HDLRとC_HDLRに通知する．この部分をプロセスG_BTN とする．

(4) 販売品関係の考察(その2) 商品の出力関連(G_DSP:Goods Dispenser):

「制御部分」の指示により，一個の商品を排出する．ある商品が無くなったときはアラームの表示をする．アラーム表示中の商品に対して，その供給が行われたらアラーム表示を消す．この部分をプロセス G_DSPと名づける．

(5) 紙幣と硬貨それぞれの勘定を行う部分 (B_COUNT:Bill Counter,C_COUNT:Coin Counter)

プロセスB_HDLRとC_HDLRから，投入された紙幣あるいは硬貨の一枚の値が通知されるので，その加算を行う．それぞれ，プロセスB_COUNTとC_COUNTと名づける．

(6) 紙幣と硬貨の返却 (B_RTRN:Bill Return,C_RTRN:Coin Return)

「制御部分」より，返却すべき紙幣あるいは硬貨の総額が通知されるので，その指示に従い，B_HDLRあるいはC_HDLRに１枚の紙幣ないし硬貨を排出するように，必要な回数だけ指示を出す．なお，例えば1000円の返却を行うとき，1000円札が無ければ 500円硬貨２枚，それもなければ 500円硬貨１枚と 100円硬貨５枚というように，なるべく大きい単位の通貨を用いて返却するものとする．釣銭排出後は，通貨の投入が可能となるように，紙幣および硬貨の投入口を開く．B_RTRNとC_RTRNとは並列動作が可能なものとする．

(7) 制御部分(V_CTRL:Vending Control)

上記で「制御部分」と呼んでいたものを，V_CTRLとして一つのプロセスとする．商品に対応する正確な金額が投入されたときと，釣銭があるときに，商品の排出の指示と，必要な釣銭の返却の指示を行う．釣銭を出すために十分な紙幣と硬貨の枚数があるかどうかを調べるプロセスCHANGE_COUNTをも用意する．

以上で必要なモジュールとその概要を定めたので，以下ではこれらのモジュールの間を結ぶ関係すなわちチャネルを明示しながら，設計を進める．

4.3 大域的変数

システムと外部の連絡のために次の大域的な論理型変数を用意する．通信と同期のためには使用しない．

alarmfb(i):alarm few bills
　第ｉ種の紙幣が一定枚数以下になったことの表示．
alarmfc(i):alarm few coins
　第ｉ種の硬貨が一定枚数以下になったことの表示．
alarmng(i):alarm no goods
　第ｉ種の商品がなくなったことの表示．
dispenseg(i):dispense goods
　第ｉ種の商品が排出されたことの表示．
dispenseb(i):dispense bill
　第ｉ種の紙幣が排出されたことの表示．
dispensec(i):dispense coin
　第ｉ種の硬貨が排出されたことの表示．

4.4 プロセス結合図

プロセスの結合図(connection diagram)を描く．これは，プロセスと，それらの間を結ぶチャネルとを記述したものである（図２）．矢印はデータの流れる方向を示す．次の 4.5節で各チャネルの説明をする．

4.5 チャネルおよびイベントの一覧

(1) 紙幣関係

bi.b(i)　チャネル bi(bill_input)上の，紙幣の種類の情報（例　Y5000 ）．
bm.v(b(i))　チャネル bm(bill_middle)上の，紙幣の値の情報（例 5000 ）．
bd.b(i)　チャネル bd(bill_dispense)上の，紙幣の種類の情報(例　Y5000)．

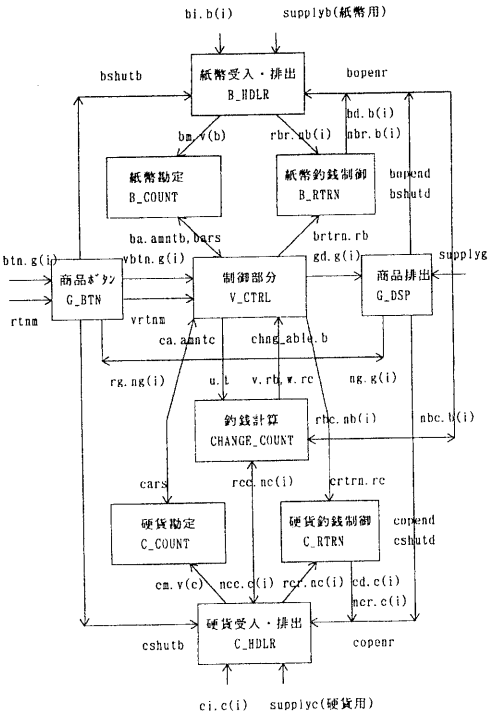

図2 プロセスの結合図
Fig. 2 Connection Diagram

nbc.b(i) チャネル nbc(number_of_bills_from_CHANGE_COUNT)上での，第 i 種紙幣の保存枚数の問合せ．
nbr.b(i) チャネル nbr(number_of_bills_from_B_RTRN)上での，第 i 種の紙幣の保存枚数の問合せ．
rbc.nb(i) チャネル rbc(reply_bill_number_to_CHANGE_COUNT)上での，紙幣保存枚数nb(i)の通知．
rbr.nb(i) チャネル rbr(reply_bill_number_to_B_RTRN)上での，紙幣保存枚数nb(i)の通知．
brtrn.rb チャネル brtrn(bill_return)上の，紙幣の値の情報(例 9000)．
ba.amntb チャネル ba(bill_amount)上の，紙幣の総額の情報 (例 10000)．
bars イベント bars(bill_amount_reset)で，カウンターをリセットする．
bopend,bopenr 紙幣投入孔を開く．これによりb_open=trueとなる．bopendはG_DSPからのイベントで，bopenrはB_RTRNからのイベントである．
bshutb,bshutd 紙幣投入孔を閉じる．これによりb_open=falseとなる．bshutbはG_BTNからのイベントで，bshutdはG_DSPからのイベントである．
supplyb 釣銭の供給を示す．各種類の紙幣の一定枚数が供給されるものとする．

(2) 硬貨関係
ci.c(i) チャネル ci(coin_input)上の，硬貨の種類の情報（例 HALF）．
cm.v(c(i)) チャネル cm(coin_middle)上の，硬貨の値の情報 (例 50)．
cd.c(i) チャネル cd(coin_dispense)上の，硬貨の種類の情報(例 HALF)．
ncc.c(i) チャネル ncc(number_of_coins_from_CHANGE_COUNT)上の，第 i 種硬貨の保存枚数の問合せ．
ncr.c(i) チャネル ncr(number_of_coins_from_C_RTRN)上の，第 i 種の硬貨の保存枚数の問合せ．
rcc.nc(i) チャネル rcc(reply_coin_number_to_CHANGE_COUNT)上で，硬貨保存枚数nc(i)の通知．
rcr.nc(i) チャネル rcr(reply_coin_number_to_C_RTRN)で，硬貨保存枚数nc(i)の通知．
crtrn.rc チャネル crtrn(coin_return)上の，硬貨の値の情報（例 90 ）．
ca.amntc チャネル ca(coin_amount)上の，硬貨の総額の情報 (例 125)．
cars イベント cars(coin_amount_reset)で，カウンターをリセットする．
copend,copenr 硬貨投入孔を開く．これによりc_open=trueとなる．copendはG_DSPからのイベントで，copenrはC_RTRNからのイベントである．
cshutb,cshutd 硬貨投入孔を閉じる．これによりc_open=falseとなる．cshutbはG_BTNからのイベントで，cshutdはG_DSPからのイベントである．
supplyc 釣銭の供給を示す．各種類の硬貨の一定枚数が供給されるものとする．

(3) 商品関係
btn.g(i) チャネル btn(button)上の，商品の種類goods(i)の情報（例 chocolate ）．
vbtn.g(i) V_CTRLに対するチャネル vbtn (V_CTRL.button)上の， goods(i)の情報．
gd.g(i) チャネル gd(goods_dispense)上の，goods(i)の情報．
ng.g(i) チャネル ng(number_of_goods)上で，商品g(i)の保存個数を尋ねる．
rg.ng(i) チャネル rg(remaining_goods)上で，第 i 種の商品の保存個数ng(i)を返す．
supplyg 商品の供給を示す． 各種類の商品の一定個数が供給されるものとする．
rtnm イベントrtnm(return money)による，投入通貨返却の指示．
vrtnm V_CTRLに対する投入通貨返却の指示．

(4) 釣銭関係
u.t チャネル u 経由，釣銭の総額を与える．
chng_able.b チャネル chng_able(change_able)経由，釣銭の存否を論理型の値で示す．
v.rb チャネル v 経由，紙幣の返却値の通知(rb=return_bill_value)．
w.rc チャネル w 経由，硬貨の返却値の通知(rc=return_coin_value)．

4.6 各プロセスのＣＳＰによる設計
4.2 節の概略に基づき，各プロセスの記述を以下に与える．各プロセスのかなり複雑な動作が CSPでは簡潔に記述できる．

4.6.1 B_HDLR, C_HDLR の記述
αB_HDLR ={supplyb,bopend,bopenr,bshutb,bshutd}
∪{bi.b(i),bd.b(i),bm.v(b(i)),nbc.b(i),nbr.b(i),
rbc.nb(i),rbr.nb(i) | i∈BILL_T}
以下これを次のように略記する．
αB_HDLR={supplyb,bopend,bopenr,bshutb,bshutd,
bi.b(i),bd.b(i),bm.v(b(i)),nbc.b(i),nbr.b(i),
rbc.nb(i),rbr.nb(i)}
B_HDLR
=(supplyb→(∀i:BILL_T・nb(i):=med_bill);B_HDLR
| bi?b(i)→(nb(i):=nb(i)+1);bm!v(b(i))→B_HDLR
| bshutb →(b_open:=false);B_HDLR

```
| bshutd →(b_open:=false);B_HDLR)
 ≮b_open≯
 (nbc?b(i)→rbc!nb(i)→B_HDLR
| nbr?b(i)→rbr!nb(i)→B_HDLR
| bd?b(i)→
          (dispenseb(i):=true);(nb(i):=nb(i)-1);
               (((alarmfb(i):=true);SKIP)
                ≮nb(i) ≦ min_bill ≯
                SKIP); B_HDLR
| bopend→(b_open:= true);B_HDLR
| bopenr→(b_open:= true);B_HDLR)

αC_HDLR={supplyc,copend,copenr,cshutb,cshutd,
  ci.c(i),cd.c(i),cm.v(c(i)),ncc.c(i),ncr.c(i),
  rcc.nc(i)rcr.nc(i)}
```

C_HDLRはB_HDLRと同様である．

注．∀i:BILL_T・nb(i):=med_billは，列挙型BILL_Tのすべての要素iに対して，nb(i):=med_billを行うという意味であり，ここで導入した記法である．

4.6.2 G_BTN の記述

```
αG_BTN={rtnm,bshutb,cshutb,vrtnm,btn.g(i),
        ng.g(i),rg.ng(i),vbtn.g(i)}
G_BTN
=(rtnm→bshutb→cshutb→vrtnm→G_BTN
| btn?g(i)→bshutb→cshutb→ng!g(i)→rg?ng(i)
           → (( vbtn!g(i)→ G_BTN)
               ≮ng(i)＞0≯
               ( vrtnm → G_BTN)))
```

4.6.3 G_DSP の記述

```
αG_DSP={supplyg,ng.g(i),rg.ng(i),gd.g(i),bopend,
        copend,bshutd,cshutd}
G_DSP
=(supplyg→(∀i:GOODS_T・ng(i):=max_goods);
   (no_goods:=false);bopend→copend→G_DSP
| ng?g(i)→rg!ng(i)→G_DSP
| gd?g(i)→
          (dispenseg(i):=true);(ng(i):=ng(i)-1);
    (((alarmng(i):=true);SKIP)
     ≮ng(i) = 0≯
     SKIP);
    (((no_goods:=true);bshutd→cshutd→SKIP)
    ≮∀i:GOODS_T・ng(i)=0≯
     SKIP);G_DSP)
```

注．∀i:GOODS_T・ng(i)=0 は，型 GOODS_Tのすべての要素iに対して ng(i)=0 が成り立つという意味である．

4.6.4 B_COUNT, C_COUNTの記述

```
αB_COUNT={bm.v(b(i)),ba.amntb,bars}
B_COUNT
=(bm?v(b(i))→(amntb:=amntb+v(b(i)));B_COUNT
| ba!amntb→B_COUNT
| bars→(amntb:=0);B_COUNT)
αC_COUNT={cm.v(c(i)),ca.amntc,cars}
```

C_COUNTはB_COUNTと同様である．

4.6.5 B_RTRN, C_RTRN の記述

```
αB_RTRN={brtrn.rb,bd.b(i),bopenr,nbr.b(i),
         rbr.nb(i)}
B_RTRN=(brtrn?rb→BR_SIGNAL;bopenr→B_RTRN)
```

ここで BR_SIGNAL は Bill Return Signalの意であり，紙幣による釣銭の，総額から種類を決めてB_HDLRに通知する逐次的な手続きである．CSP の逐次的な制御構造の記述能力には不自由な点があり，ここでは直接に言語 Adaを用いて BR_SIGNALの仕様(例)を記述する．

```
  type BILL_T is (Y10000,Y5000,Y1000);
  BILL_LIST : array(BILL_T) of PRICE_T
            := (10_000,5_000,1_000);
  BT := rb;
  B := BILL_T'FIRST;
  B_HDLR.NB(B,N);           -- nbr!b(i)→rbr?nb(i)
  while BT >= BILL_LIST(BILL_T'LAST) loop
    while BT >= BILL_LIST(B) and N /= 0 loop
      B_HDLR.BD(B);         -- bd!b(i)
      BT := BT - BILL_LIST(B);
      N := N-1;
    end loop;
    exit when B = BILL_T'LAST;
    B := BILL_T'SUCC(B);
    B_HDLR(B,N);            -- nbr!b(i)→rbr?nb(i)
  end loop;
```

```
αC_RTRN={crtrn.rc,cd.c(i),copenr,ncr.c(i),
         rcr.nc(i)}
C_RTRN=(crtrn?rc→CR_SIGNAL;copenr→C_RTRN)
```

ここで CR_SIGNALは Coin Return Signal の意であり，その内容は上記の BR_SIGNAL と同様である．

4.6.6 V_CTRLとCHANGE_COUNTの記述

```
αV_CTRL={vbtn.g(i),vrtnm,ba.amntb,ca.amntc,bars,
  cars,gd.g(i),brtrn.rb,crtrn.rc,u.t,chng_able.b,
  v.rb, w.rc}
V_CTRL
=((vbtn?g(i)→(rtnm_1:=false)
| vrtnm→(rtnm_1:=true));
  ba?amntb→bars→ca?amntc→cars
  →(amnt:=amntb+amntc);
  (((t:=amnt-price(g(i)));u!t→chng_able?b→SKIP)
  ≮not rtnm_1≯
    SKIP);
  ((gd!g(i)→ SKIP)
  ≮(t=0 or t>0 and b=true) and not rtnm_1≯
   (u!amnt → chng_able?b → SKIP));
    v?rb→w?rc→brtrn!rb→crtrn!rc→V_CTRL)
```

```
αCHANGE_COUNT={u.t,nbc.b(i),rbc.nb(i),ncc.c(i),
               rcc.nc(i),chng_able.b,v.rb,w.rc}
CHANGE_COUNT
=(u?t→(∀i:BILL_T・(nb!b(i)→rb?nb(i)→SKIP));
      (∀i:COIN_T・(nc!c(i)→rc?nc(i)→SKIP));
      calculate(t,b,rb,rc);
      chng_able!b→CHANGE_COUNT
| v!rb→w!rc→CHANGE_COUNT)
```

ここで手続きcalculateは，釣銭の総額tを与えられて，それを払うことが可能かどうか(b)，可能のときは紙幣による額(rb)と硬貨による額(rc)とを返すものとする．

4.7 プロセス間でのチャネルの共有について

CSP では，例えば，第i種の紙幣の枚数を問合わせるのに，イベントnb.b(i) が三つのプロセスで記述されていると，これら三つのイベントが同時に発生することを要求する．従って，三つのうちの二つだけのイベントの同期を要求する場合は，ここで行ったように，nbc.b(i)とnbr.b(i)としてチャネルを分ける必要がある．同様のことはイベントbopenとbshutについてもいえる．

５．プロセス合成の計算からの考察

本節では，CSPにおけるプロセス合成の規則を用いていくつかのプロセスを合成することにより，より少ない個数のプロセス群にまとめることができることと，それにより例題のシステムにデッドロックの生じないことを示す．次にプログラム開発面からの CSPの問題点を述べる．

5.1 プロセス合成に関する一般的な規則

文献[4]の規則のうちでよく用いるものをここで用意する．

(1) 文献[4] p.71の次の規則を使う．これを三つ以上のプロセスの合成のために拡張することは可能である．
P=(x:A→P(x)),Q=(y:B→Q(y)) のとき，
(P ‖ Q)=(z:C→(P' ‖ Q'))
ただし，C=(A∩B)∪(A-αQ)∪(B-αP)とし，
P'=もし z∈A ならば P(z) さもなければ P
Q'=もし z∈B ならば Q(z) さもなければ Q

(2) 文献[4] p.188 の次の公式を利用する．
(((x:=e);P) ‖ Q)=((x:=e);(P ‖ Q))
但し，x⊆var(P)-acc(Q) かつ acc(e)∩var(Q)={}とする．ここで，var(P)はプロセスP内で値の変更を受ける可能性のある変数の集合とし，acc(P)はP内で参照される可能性のある変数の集合を表す．

(3) 文献[4] p.189 の次の公式を用いる．
P‖(Q≪b≫R)=(P‖Q)≪b≫(P‖R)
但し， acc(b)∩var(P)={}．

(4)隠蔽(concealment)とは，一つのシステムを構成する成分であるプロセスPとQに共通のイベントおよびチャネルを，このシステムの外部からは見えないものに変形することを言う．なお文献[4] p.113のように，隠蔽を行うと一般に非決定性選択Π(文献[4] p.102)や汎選択[](文献[4] p.106)が出現し，複雑化の原因になるので，本論文では限られた範囲でのみ使用する．

5.2 B_HDLRとB_COUNTの合成

4.6.1節のB_HDLRと，4.6.4節の B_COUNTの仕様を，上記の規則を用いて合成することにより，プロセス合成計算の一端を示す．

B_HDLR=BH1≪b_open≫BH2
とおく．
```
B_HDLR ‖ B_COUNT
=(BH1≪b_open≫BH2) ‖ B_COUNT
=(BH1 ‖ B_COUNT)≪b_open≫(BH2 ‖ B_COUNT)
```
となる．ここで，
```
BH1=(supplyb→(∀i:BILL_T・b(i):=med_bill);BH1
 | bi?b(i)→(nb(i):=nb(i)+1);bm!v(b(i))→BH1
 | bshutb →(b_open:=false);BH2
 | bshutd →(b_open:=false);BH2)
BH2=(nbc?b(i)→rbc!nb(i)→BH2
 | nbr?b(i)→rbr!nb(i)→BH2
 | bd?b(i)→
        (dispenseb(i):=true);(nb(i):=nb(i)-1);
     (((alarmfb(i):=true);SKIP)
     ≪nb(i) ≦ min_bill ≫
      SKIP); BH2
 | bopend→(b_open:= true);BH1
 | bopenr→(b_open:= true);BH1)
```
さらに，
```
BH11=bm!v(b(i))→BH1
BH21=rbc!nb(i)→BH2
BH22=rbr!nb(i)→BH2
```
とおくと，
```
BH1 ‖ B_COUNT
=(supplyb→(∀i:BILL_T・nb(i):=med_bill);
                                (BH1 ‖ B_COUNT)
 | bi?b(i)→(nb(i):=nb(i)+1);(BH11 ‖ B_COUNT)
 | bshutb →(b_open:=false);(BH2 ‖ B_COUNT)
 | bshutd →(b_open:=false);(BH2 ‖ B_COUNT)
 | ba!amntb→(BH1 ‖ B_COUNT)
 | bars→(amntb:=0);(BH1 ‖ B_COUNT))
```
ここで
```
BH11 ‖ B_COUNT
=(bm.v(b(i))→(amntb:=amntb+v(b(i)));
                                (BH1 ‖ B_COUNT)
 | ba!amntb→(BH11 ‖ B_COUNT)
 | bars→(amntb:=0);(BH11 ‖ B_COUNT))
```
一方，
```
BH2 ‖ B_COUNT
=(nbc?b(i)→(BH21 ‖ B_COUNT)
 | nbr?b(i)→(BH22 ‖ B_COUNT)
 | bd?b(i)→
        (dispenseb(i):=true);(nb(i):=nb(i)-1);
     (((alarmfb(i):=true);SKIP)
     ≪nb(i) ≦ min_bill ≫
      SKIP);(BH2 ‖ B_COUNT)
 | bopend →(b_open:= true);(BH1 ‖ B_COUNT)
 | bopenr →(b_open:= true);(BH1 ‖ B_COUNT)
 | ba!amntb→(BH2 ‖ B_COUNT)
 | bars→(amntb:=0);(BH2 ‖ B_COUNT))
```
ここで，
```
BH21 ‖ B_COUNT
=(rbc!nb(i)→(BH2 ‖ B_COUNT)
 | ba!amntb→(BH21 ‖ B_COUNT)
 | bars→(amntb:=0);(BH21 ‖ B_COUNT))

BH22 ‖ B_COUNT
=(rbr!nb(i)→(BH2 ‖ B_COUNT)
 | ba!amntb→(BH22 ‖ B_COUNT)
 | bars→(amntb:=0);(BH22 ‖ B_COUNT))
```
このように，プロセスを合成すると，一般にプロセスの状態数は急激に増加する．

5.3 デッドロック不在の証明の概要

本節では，プロセス合成の規則を用いて，本自動販売機ではデッドロックの起らないことの証明の概略を示す．ユーザの動作も仕様化しておく必要がある．次のように仮定する．
(1) ユーザは最低１つは通貨を投入した後，商品ボタンか返却ボタンを押す．
(2) supplyb, supplyc, supplyg は起らない．
以上の条件はユーザ（＝外界）の動作として次のように記述できる．

```
αUSER={bi.b(i),ci.c(i),btn.g(i),rtnm,supplyb,
       supplyc,supplyg}
USER=(bi!b(i)→USER1) Π (ci!c(i)→USER1)
USER1=USERΠ(btn!g(i)→SKIP)Π(rtnm!→SKIP)
```

演算子Πはプロセス間の非決定性を表す．ここで最後に SKIP としたのは，一人のユーザが商品や釣銭を受け取る前に次のユーザが通貨を入れ始めようとする並列性を避けるために，ユーザ動作の１サイクルだけを記述するためである．

ここでは，USERを含めたすべてのプロセスを一挙に合成する．

```
(B_HDLR ‖ B_COUNT ‖ B_RTRN)
‖ (C_HDLR ‖ C_COUNT ‖ C_RTRN)
‖ (G_BTN ‖ G_DSP)
‖ (V_CTRL ‖ CHANGE_COUNT)＼{u,chng_able,v,w}
‖ USER
```

ここで＼は隠蔽を表す．

この式を前節と同様に展開していき，展開の途中で STOP が現れなければ，デッドロックの生じないことが証明されたことになる．この展開はここでは割愛するが，デッドロック不在を証明できる．すなわち，ユーザの動作を前述のUSERに限定したときには，デッドロックの起ることなしに1サイクルを終了することを証明できる．この計算には約 1,200行のプロセス変形の計算を要した．

これと同様なやり方で，supplyg などを含んだ保守動作に対しても，デッドロック不在を証明することが可能である．逆に，デッドロックの起るようなユーザの動作を見出すこともできる．

CSP の代数法則を用いてプロセスを次々に合成していく時，最初の間はプロセス相互の間の動作の制約があまり作用しないため，式の大きさは急激に増大する．しかしやがて相互の制約が作用し合って，大きさは減少していく．

5.4 プログラム開発面から見た CSPの問題点

並列処理の本質として，二つの並置されたプロセスを合成する時，イベントについて，許される順序のすべてを考慮する必要がある．従って，例えばB_HDLRとB_COUNTの合成プロセスは，5.2節のとおり，複雑なものとなる．一方，当初から B_HDLR の中に B_COUNTの機能を含めて設計すると，チャネル bm がなくなるなど，次のより簡単な仕様で十分である．この仕様はプロセス合成とbmの隠蔽によってだけでは導くことはできない．

```
B_HC=
((supplyb→(∀i:BILL_T・nb(i):=med_bill);B_HC
| bi?b(i)→
   (nb(i):=nb(i)+1);(amntb:=amntb+v(b(i)));B_HC
| bshut?x→(b_open:=false);B_HC)
≮b_open≯
 (nbc?b(i)→rbc!nb(i)→B_HC
| nbr?b(i)→rbr!nb(i)→B_HC
| bd?b(i)
  →(dispenseb(i):=true);(nb(i):=nb(i)-1);
      (((alarmfb(i):=true);SKIP)
      ≮ nb(i)≦min_bill ≯
       SKIP);B_HC
| bopend→(b_open:=true);B_HC
| bopenr→(b_open:=true);B_HC
| ba!amntb→B_HC
| bars→(amntb:=0);B_HC))
```

多数の小さなプロセスの仕様をまず記述し，それらを合成して適当な大きさのプロセスを作り出すという開発方法を用いるとすると，現在の CSPの合成規則に，イベントの順序について一定の制限を加える手段を追加する必要がある．

6 Ada プログラムの開発

CSPによる記述から人手によりAdaプログラムに書き直す．Ada プログラムとしては，自動販売機のプログラムを汎用パッケージとして作成する．

6.1 CSP のプロセスとAda のタスクの対応

CSP のプロセスやイベントはAda のタスクおよびランデブーによって自然に実現できる．CSP とAda の対応の一例を示す．

```
CSP :   P=(e1→P | e2→P)
Ada :   task P is
          entry E1;
          entry E2;
        end task;
        task body P is
        begin
          loop
            select
              accept E1;
            or
              accept E2;
            end select;
          end loop;
        end P;
```

チャネルを介した入出力は， Adaのランデブーによって実現できる．プロセス結合図において矢の出発側はエントリ呼出に対応し，到達側はaccept文に対応する，とするのが自然である．イベントc?v は

```
accept C(V:in V_TYPE) do X:=V; end C;
```

に対応する．

しかし Adaの out引数を用いた異なる対応のさせ方も可能で，c!v は，エントリ呼出しでなく，

```
accept C(TV:out V_TYPE) do TV:=V; end C;
```

に対応するとする．

しかし CSPの全機能が Adaの機能に直接的に対応するわけではない．とくに，プロセス間の非決定性のもっとも一般的な形(演算子Π)は，Ada には存在しない．Ada における非決定性は上述のselect文におけるものだけである．また，CSP の同期は三つ以上のプロセス間で可能であるが，Ada のランデブーは二つのタスク間のみである．つまりCSP における同期を Adaでつねに直接的には実現できない．

以上のように，非決定性と，同期の方式という基本的な点で，CSP と Adaとの間に根本的な差がある．しかし本例題での CSPから Adaへの書き直しにおいては，これらの点は問題になることはなかった．つまりこれらの CSPの機能は，実際的な並列処理システムの記述のためには一般的に過ぎると考えられる．

6.2 Ada プログラムの構造

(1) 汎用構造

汎用パッケージとして実現する．課題の要請により，紙幣と硬貨の種類，商品の種類とその値段，および扱う数値の範囲を汎用パラメータとする．

(2) タスク構造

Ada プログラムにおけるタスクは，CSP による記述のプロセスそのままの，次の10個とする．

B_HDLR, C_HDLR, G_BTN, G_DSP, B_COUNT, C_COUNT, B_RTRN, C_RTRN, V_CTRL, CHANGE_COUNT

(3) 汎用パッケージの具体化（インスタンス）を作成して利用するプログラム

貨幣制度の変更や販売品とその値段の変更の実例として，円を通貨とする個別的実体を作り出して使用するプログラムをMAINVNDYと名づける．

6.3 Ada プログラムとCSP 仕様の大きさ

得られたAda プログラムの大きさを次に示す．
汎用パッケージ　PCKGVEND 498行(CSPは 97 行)
主プログラム　　MAINVNDY 66行

汎用パッケージPCKGVENDの中にある10個のタスクの行数と，対応するCSP の仕様の大きさを次に示す．

	Ada行数	CSP行数		Ada行数	CSP行数
B_HDLR	59	18	C_HDLR	59	18
G_BTN	29	8	G_DSP	53	14
B_COUNT	27	5	C_COUNT	27	5
B_RTRN	36	3	C_RTRN	36	3
V_CTRL	46	15	CHANGE_COUNT	67	8

CSP による記述の量は，最終の Adaプログラムと比べて，約 1/5であった．

7．ＣＳＰ使用の評価

(1) CSP は，並列処理の仕様を記述するための抽象度のレベルとしては，ちょうど適当であると考えられる．例．プロセス B_COUNTのCSPによる記述は，4.6.4節のとおりである．これに対して，Ada によるタスク仕様部とタスク本体部の記述は次の通りである．

```
task B_COUNT is
  entry BM(BV : in PRICE_T);
  entry BA(A : out PRICE_T);
  entry BARS;
end B_COUNT;
task body B_COUNT is
  TA : PRICE_T := 0; B : PRICE_T;
begin
  loop
    select
      accept BM(BV : in PRICE_T) do
         B := BV;
      end BM;
      TA := TA+B;
    or
      accept BA(A : out PRICE_T) do
        A := TA;
      end BA;
    or
      accept BARS;
      TA:=0;
    or
      terminate;
    end select;
  end loop;
end B_COUNT;
```

上の例では，CSPの5行が，Adaの26行に対応している．CSP は適切な抽象化のレベルで並列動作プロセスを表現することができる．

(2) CSP ではプロセスの代数的な変形操作が可能であり，これを用いて設計段階で十分な検討が行え，その結果，設計の洗練化が可能である． Adaでプログラムを書いた後にこのような検討を行うのはかなり大変である．

(3) デッドロック不在の証明は，かなりの計算量であったが， CSP以外では不可能と思われる．LOTOS では，記法上から，このような代数計算は非常にやりにくい．

(4) プロセスを合成すると，互いに関係をもたないイベント間の時間的順序の不定性(非決定性)のため，合成されたプロセスの，イベントの順序の組み合わせの数が膨大になる．このことは並列処理の逐次化においてつねに起ることであるが，これが合成計算を複雑にする最大の原因である．実際のプログラム作成においては，意味をもたない非決定性は，そのうちの一つを適当に選んでしまうことによって，逐次的なもので済ませる．このことを合成計算において，あるいはプログラム開発において，うまく定式化する必要がある．

(5) 逐次的だが論理的に複雑な処理操作の途中でイベントを発生させるようなプロセスは，CSP では記述しにくく，Ada のような手続き型言語の方が書きやすい．

(6) CSP には，LOTOS と違って，抽象データ型に対する代数的仕様記述の機能がない．しかしこれはプロセスの並列処理の記述からは独立した概念であり，CSP にこのような機能を追加することは容易である．

8　おわりに

本論文で得られた結論は以下のとおりである．

(1)CSPは，仕様の論理的ないし代数的な処理を可能とするだけの抽象度の高さと，現実的な並列システムを記述するための木目の細かさという，相反する条件をちょうど満たす，仕様記述言語であると言える．

(2) 並列処理システムのイベントによる仕様記述は，記述能力とプロセス計算による仕様検討の可能性だけを取りあげても，明解で優れたものである．

(3)CSPの逐次処理機能の細部において，仕様記述のためにやや不十分な面があり，既存の手続き型言語で補うことが必要な場合がある．

(4) プロセス合成において非決定性による仕様の膨張をおさえて，実用的な仕様を導き出す手法を開発する必要がある．

(5)CSPによる仕様記述から Adaプログラムへの人手による導出は，両言語をよく知る者にとっては容易である．

参考文献

1) Ada Programming Language, ANSI/MIL-STD-1815A, 1983.
2) R.J.A.Buhr: System Design with Ada, p.256, Prentice-Hall, 1984.
3) C.A.R.Hoare: Communicating Sequential Proccesses, Comm.A.C.M., vol.28(1979), pp.666-677.
4) C.A.R.Hoare: Communicating Sequential Proccesses, p.256, Prentice-Hall, 1985.
5) J.Hooman: Specification and Compositional Verification of Real-Time Systems, Lecture Notes in Computer Science, vol.558, p.235, Springer-Verlag, 1991.
6)情報処理振興事業協会：プログラムの演繹的導出法の調査研究，中間報告書，第一分冊，p.396, 1992.
7) Information processing systems - Open Systems Interconnection -LOTOS- A formal description technique based on the temporal ordering of observational behaviour, p.142, ISO 8807, 1989.
8) R.Milner: A Calculus of Communicating Systems, Lecture Notes in Computer Science, vol.92, Springer-Verlag, 1980.
9) J.M.Wing:A specifier's introduction to formal methods, IEEE Computer, Vol.23, No.9, pp.8-24, 1990.
10) 山本，丸山，喜家村:CSP入門(1)-(6), bit, vol.23, 1991.