プログラミング言語　24－6
（1990. 2. 9）

分散型システムの仕様記述と設計支援への関数型プログラミング言語の応用

荒木　啓二郎

九州大学工学部情報工学科

あらまし　分散型システムの開発を形式的な枠組みのもとに行うことを目的として，ストリーム処理関数を分散型システムの仕様記述に利用する方法について検討する．分散型システムを，ストリームによって結合された並行プロセスのネットワークとしてモデル化し，システム開発過程における要求定義ならびに仕様記述において，抽象データ型として形式的に定義されたストリームとストリーム処理関数とを用いる方法を，簡単な通信システムの記述と正当性の証明，ならびに，哲学者の食事問題の二つの例題に適用した例を掲げる．この方法に基づいて分散型システムの開発を行う際に，関数型プログラミング言語Ｍｉｒａｎｄａを実行可能な仕様記述言語として利用する．

Application of Functional Programming Languages to Specification and Design of Distributed Systems

Keijiro ARAKI

Department of Computer Science and Communication Engineering
Kyushu University
Hakozaki, Higashi-ku, Fukuoka 812 JAPAN

Abstract　We discuss a development method for distributed systems based on a formal specification technique with stream processing functions. Distributed systems are described as networks of parallel processes communicating each other with streams. We show two examples. A simple communication system is specified formally and proved correct with respect to its requirement. The dining philosopher problem is also specified as a network of stream processing functions. We use the functional programming language Miranda as an executable specification language.

1．はじめに

分散型システムは，非決定的な振舞いや，意味の明確な定義が容易でないことなどのために，その開発には多くの困難を伴う．このために，分散型システムに対する有効な開発方法論の確立や開発支援環境の整備が望まれている．

本稿では，分散型システムの開発を形式的な枠組みのもとに行うことを目的として，ストリーム処理関数[田中-88]を分散型システムの仕様記述に応用することを検討する．ストリームとは，データの一次元系列であり，その先頭の要素から順にアクセスされる．ストリームは，有限の系列でもよいし，無限の系列でもよい．

分散型システムを，ストリームによって結合された並行プロセスのネットワークとしてモデル化し，システムに対する要求定義ならびに仕様記述において，抽象データ型として形式的に定義されたストリームとストリーム処理関数とを用いる方法 [Broy-87, Broy-88] を簡単に紹介して，それを適用した例題を掲げる．この方法に基づいて分散型システムの開発を行う際に，我々は，関数型プログラミング言語 Ｍｉｒａｎｄａ [Turner-85b, RSL] を実行可能な仕様記述言語として利用する．

以下では，まず２節で，ストリームとその演算とを提示した後，分散型システムをストリーム処理関数のネットワークとして仕様記述する方法を述べる．その記述には，関数型プログラミング言語Ｍｉｒａｎｄａを用いる．３節では，簡単な通信システムと哲学者の食事問題の二つの例題を，ストリーム処理関数のネットワークとして記述する．いずれの例題についても，Ｍｉｒａｎｄａによる記述とその実行例を掲げる．４節では，Ｍｉｒａｎｄａを実行可能な仕様記述言語として用いる分散型システム開発法について考察する．５節では，今後の課題等を述べる．

2．ストリーム処理関数と並行動作システムの記述

本節では，ストリームとその演算を定義し，ストリーム処理関数から構成されるネットワークとして分散型システムを記述する方法について述べる．

2．1　ストリーム

集合Ａの要素からなるストリームを STREAM(A) とし，次のように定義する．

$$STREAM(A) = A^* \cup A^\infty$$

ここで，A^∞ は無限長の系列の集合である．

また，$A^\perp = A \cup \{\perp\}$ とする．$\perp$は，未定義であることを表わす．STREAM(A) 上の半順序$\sqsubseteq$を，s1, s2$\in$STREAM(A)に対して次のように定義する．

$$s1 \sqsubseteq s2 \equiv \exists s3 \in STREAM(A).\ s2 = s1 \circ s3$$

ストリームに関する演算としては次のようなものを用いる．

```
fst : stream(A) → A⊥
rest : STREAM(A) → STREAM(A)
& : A⊥ × STREAM(A) → STREAM(A)
^ : STREAM(A) × STREAM(A) → STREAM(A)
[ ] : STREAM(A) × positive → A⊥
[ .. ] : STREAM(A) × positive × positive
          → STREAM(A)
| | : STREAM(A) → natural
# : A × STREAM(A) → natural
< > : A → STREAM(A)
isempty : STREAM(A) → boolean
```

ここで，natural, positive, boolean は，それぞれ，自然数，非負整数，論理値の集合を表わす．以下，これらの演算について説明する．

演算 fst(s) は，ストリーム s の先頭の要素を値とし，

$$fst(\varepsilon) = \perp$$
$$fst(a \,\&\, s) = a$$

と，定義する．rest(s) は，ストリームの先頭の要素を除いた残りのストリームを値とし，

$$rest(\varepsilon) = \varepsilon$$
$$rest(a \,\&\, s) = s$$

と，定義する．a & s は，ストリーム s の先頭に要素 a を一つ加えてできるストリームを値とし，

$$\perp \,\&\, s = \varepsilon$$
$$a \,\&\, s = \langle a \rangle \,\hat{}\, s$$

と，定義する．s1 ^ s2 は，二つのストリーム s1 と s2 を繋ぎ合わせて得られるストリームを値とする．ただし，s1$\in A^\infty$ に対しては，s1 ^ s2 = s1 とする．s[k] は，ストリーム s 中の k 番目の要素を表わし，

$$s[k] = fst(rest^k\ s)$$

と，定義する．但し，$rest^0\ s = s$, $k>0$ に対して，

$rest^k$ = rest($rest^{k-1}$ s) とする．s[k..l] は，ストリーム s の k 番目から l 番目までの要素からなる部分ストリームを値とする．但し，k>l の場合には s[k..l]=ε とする．|s| は，ストリーム s の長さを表わし，#(a,s) は，ストリーム s の中に現われる a の個数を表わす．<a> は，Aの要素から長さ1のストリームを生成する．isempty は，空ストリームか否かを示す．

2．2　ストリーム処理関数のネットワーク

ストリーム処理関数を，ストリームを生成・変換・消滅させる主体（プロセス）と見なすことによって，ストリームにより結合される並行プロセスのネットワークとしての分散型システムをストリーム処理関数から構成される系として記述することができる．Broy は，分散型システムをこのようなストリームによって結合された並行プロセスのネットワークとしてモデル化し，ストリームとストリーム処理関数の形式的定義に基づいて，分散型システムの開発を行なう方法を提案した．[Broy-87,Broy-88] 分散型システムの仕様を，ストリーム処理関数のネットワークとして記述し，その形式的意味を与えた．分散型システムの振舞いに対する要求は，システム中の並行プロセスの動作をインターリービングモデルによって逐次化することによって得られるストリームに関する条件として表現される．ストリーム処理関数の系として仕様記述された分散型システムが，与えられた要求を満足することを，ストリームとストリーム処理関数についての形式的枠組みの中で証明する方法を示した．

分散型システムの振舞いを表わすストリームとしては，分散型システム中で実際に入出力されるデータからなるストリームである必要はなく，システム中で発生する事象やシステムの状態などの仮想的なデータからなるストリームを用いることもできる．従って，システムのモデル化や仕様記述記述の抽象度を適切に設定できる．

2．3　Ｍｉｒａｎｄａによる記述

関数型プログラミング言語Ｍｉｒａｎｄａ [Turner-85b, RSL] は，代数的データ型や抽象データ型や無限データ構造等の機能を持つために，前述のようなストリーム処理関数のネットワークとしての分散型システムの記述に利用することができる．Ｍｉｒａｎｄａが提供するリストによってストリームを表現してもよいし，プログラマが独自に定義したデータ構造によってストリームを表現してもよい．ストリームによって結合されるプロセスのネットワークとしてモデル化された分散型システムについてストリーム処理関数の系として与えられた仕様は，容易にＭｉｒａｎｄａプログラムとして記述できる．

分散型システムを記述したＭｉｒａｎｄａプログラムは実行可能な仕様と見なすことができる．[Turner-85a] これによって，仕様の静的な分析のみならず，実際に実行させることによる実現可能性の確認や動的な分析を行なうことができる．

また，Ｍｉｒａｎｄａプログラムをプロトタイプと見なすことにより，開発すべき並行動作システムに対する仕様の明確化や要求の形成にも利用することができる．プロトタイプとして記述されたＭｉｒａｎｄａプログラムを，まずテストによって動作確認を行なった後に，改めて形式的仕様と見なして，システムに対する要求に関する検証を前述の形式的な枠組みのもとで行なうこともできる．

我々は，並行動作システムのストリームに基づく設計において，形式性と実現性の両面を兼ね備えた実行可能な仕様記述言語としてＭｉｒａｎｄａを利用することにした．

3．例題

いくつかの例題について，分散型システムをストリーム処理関数のネットワークとして仕様記述した．いずれの例も，Ｍｉｒａｎｄａで記述を行い，実行させて，その動作を確認した．ここでは，二つの例を掲げる．第一の例では，簡単な通信システムを仕様記述して，それに対する要求をその仕様が満足することを証明する．第二の例では，哲学者の食事問題を仕様記述する．

3．1　通信システム

図1には，メッセージの送受信を行なうための単純なプロトコルを表わす状態遷移図を示す．[Sunshine-79] このプロトコルに従ってメッセージの転送を行なうシステムの構成を図2に示す．図1に示すプロトコルは，通信媒体に関して以下のような仮定をおいている．

送信側から受信側にメッセージを送信する際に利用する通信媒体では，送信側が送り出したメッセージが化けることがある．しかしながら，同一のメッセージを繰り返し送信すれば，必ずいつかは化けることなく正しく受信側に届くものとする．また，通信媒体の中で，メッセージが消失したり，送信側が送っていない

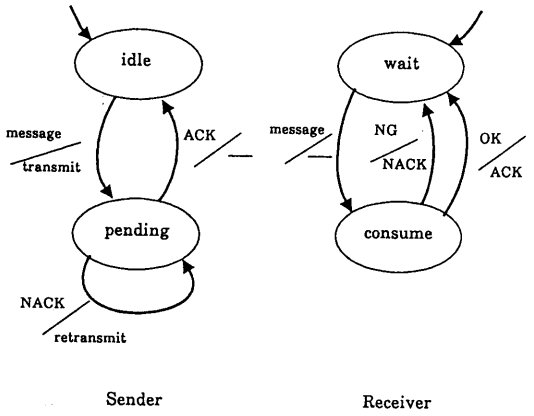

図1．簡単な通信プロトコルの状態遷移図
（文献[Sunshine-79]による）

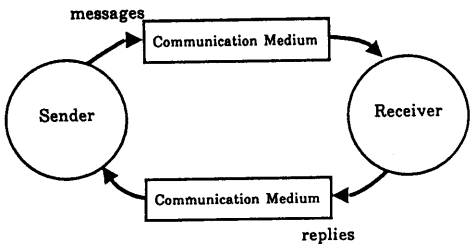

図2．通信システムの構成

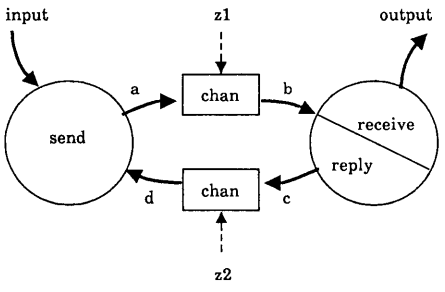

図3．ストリーム処理関数のネットワーク

メッセージが発生することはない．

一方，受信側から送信側へ応答を返す際に利用する通信媒体は，常に正しく応答を伝えるものとする．

図2に示したシステム構成を，ストリーム処理関数のネットワークとして構成したものを図3に掲げる．関数 send は，入力メッセージの系列 input を相手に送る．関数 receive は，正しく受け取ったメッセージの系列を出力する．関数 reply は，受け取ったメッセージに対して応答を返す．正しく受け取ったメッセージに対しては，ACK を送り返し，化けたメッセージに対しては，NACK を送り返す．送信側は，受信側からの応答に応じて，それが ACK ならば次のメッセージを送り出し，NACK ならば前回送り出したメッセージを再送する．

通信媒体は，関数 chan によって次のように表わす．

```
chan(z,x) = if fst(z)=0
              then fst(x) & chan(rest(z),rest(x))
              else NG & chan(rest(z),rest(x))
```

通信媒体が擾乱を起こしてメッセージを化けさせるかどうかを，chan にストリーム $z \in STREAM(\{0,1\})$ を与えることによって指定する．fst(z) = 0 の時は，擾乱なく正しいメッセージを受信側に渡し，fst(z) = 1 の時は，擾乱を起こして化けたメッセージ NG を受信側に渡す．

送信用の通信媒体に対する擾乱を指定するストリーム z1 は，前述の仮定に従って，次の条件 FAIR を満足するものとする．

$$FAIR(z) \equiv \forall n.\exists m.\ n \leq m \wedge z[m]=0$$

一方，応答返送用の通信媒体に対する擾乱を指定するストリーム z2 は，仮定により，0 のみからなるストリームである．即ち，

$$z2 = 0\ \&\ z2$$

である．

関数 send, receive, reply を，それぞれ以下のストリーム処理関数として与える．

```
send(ε,d) = ε
send(s, d) = fst(s) & send1(s,d)

send1(ε,d) = ε
send1(s, d)
  = if fst(d)=ACK
      then fst(rest(s)) & send1(rest(s),rest(d))
```

```
    else fst(s) & send1(s,rest(d))

receive(ε) = ε
receive(b)
  = if fst(b)=NG then receive(rest(b))
                 else fst(b) & receive(rest(b))

reply(ε) = ε
reply(b)
  = if fst(b)=NG then NACK & reply(rest(b))
                 else  ACK & reply(rest(b))
```

図3に示すストリーム処理関数のネットワーク構成は，次の等式の系によって表現される．

```
a = send(input,d)
b = chan(z1,a)
c = reply(b)                    (*)
d = chan(z2,c)
output = receive(b)
```

この通信システムに対して要求されることは，送信すべきメッセージの系列が，正しく受け取られることである．即ち，いかなる入力ストリーム input に対しても，どのような z1 を与えようとも FAIR(z1) を満足しておれば，

```
output = input
```

が成り立つことである．

これの証明の概要を以下に示す．

（Ⅰ）input = ε の場合

```
output
= receive(chan(z1,send(input,d)))     ((*) より)
= receive(chan(z1,ε))                 (send の定義より)
= receive(if fst(z1)=0
          then fst(ε) & chan(rest(z1),rest(ε))
          else NG & chan(rest(z1),rest(ε)))
                                      (chan の定義より)
= if fst(z1)=0
    then fst(ε)
         & receive(chan(rest(z1),rest(ε))
    else receive(chan(rest(z1),rest(ε)))
                                      (receive の定義より)
= if fst(z1)=0
    then ⊥ & receive(chan(rest(z1),ε)
    else receive(chan(rest(z1),rest(ε)))
                                      (fst, rest の定義より)
= if z1[1]=0 then ε
             else receive(chan(rest(z1),ε))
                                      (& の定義より)
= if z1[1]=0 then ε
  else if z1[2]=0 then ε
  else if z1[3]=0 then ε
    ・・・
  else if z1[k]=0 then ε
  else receive(chan(rest^k(z1),ε))
                                      (同様に繰り返して)
```

FAIR(z1) より，z1[k]=0 なる k≧1 が存在するので，

```
output = ε = input
```

となる．

（Ⅱ）input ∈ A* の場合

この時，|input| = m とする．この場合について

```
output = input
```

を示すために，次の補題を用いる．

［補題］ input ∈ A*，|input| = m とする．この時，1≦k≦m なる k について

```
output
= input[1..k]
  ^ receive(chan(rest^n(z1),
      input[k+1] & send1(rest^k(input),rest^n(d))))
```

となる n≧k が存在する．

この補題より，k=m とおくと，

```
output
= input[1..m]
  ^ receive(chan(rest^n(z1),
      input[m+1] & send1(rest^m(input),rest^n(d))))
= input ^ receive(chan(rest^n(z1),
              ⊥ & send1(rest^m(input),rest^n(d))))
= input ^ receive(chan(rest^n(z1),ε))
= input ^ ε
= input
```

（Ⅲ）input ∈ A^∞ の場合

（Ⅰ），（Ⅱ）より，input∈ A* の場合について

```
output = input
```

```
elm * ::= UNDEF | ELM * | NG
mkelm :: * -> elm *
mkelm a = ELM a
shelm :: elm * -> *
shelm (ELM a) = a

abstype stream *
with empty :: stream *
     cons :: (elm *) -> stream * -> stream *
     fst :: stream * -> (elm *)
     rest :: stream * -> stream *

stream * == [elm *]
empty = []
cons a x = [], a=UNDEF
         = a : x, otherwise
fst [] = UNDEF
fst x = hd x
rest [] = []
rest x = tl x

|| sender

send x y
    = empty, x=empty
    = cons (fst x) (send1 x y), otherwise

send1 x y
    = empty, x=empty
    = cons (fst (rest x)) (send1 (rest x) (rest y)), fst y = ack
    = cons (fst x) (send1 x (rest y)), fst y = nack

|| receiver

reply x
    = empty, x = empty
    = cons nack (reply (rest x)), fst x = NG
    = cons ack (reply (rest x)), otherwise

receive x
    = empty, x = empty
    = receive (rest x), fst x = NG
    = cons (fst x) (receive (rest x)), otherwise

|| communication media

chan z a
    = cons (fst a) (chan (rest z) (rest a)), fst z = mkelm 0
    = cons NG (chan (rest z) (rest a)), otherwise

|| system configuration

a = send input d
b = chan z1 a
c = reply b
d = chan z2 c
output = receive b
```

図４．通信システムのＭｉｒａｎｄａプログラム

```
input (Input Data) :

[ELM 'I',ELM ' ',ELM 'a',ELM 'm',ELM ' ',ELM 'h',ELM 'e',ELM ' ',ELM 'a',ELM 's'
,ELM ' ',ELM 'y',ELM 'o',ELM 'u',ELM ' ',ELM 'a',ELM 'r',ELM 'e',ELM ' ',ELM 'h'
,ELM 'e',ELM ' ',ELM '¥n',ELM 'a',ELM 's',ELM ' ',ELM 'y',ELM 'o',ELM 'u',ELM '
',ELM 'a',ELM 'r',ELM 'e',ELM ' ',ELM 'm',ELM 'e',ELM '¥n',ELM 'a',ELM 'n',ELM '
d',ELM ' ',ELM 'w',ELM 'e',ELM ' ',ELM 'a',ELM 'r',ELM 'e',ELM ' ',ELM 'a',ELM '
l',ELM 'l',ELM ' ',ELM 't',ELM 'o',ELM 'g',ELM 'e',ELM 't',ELM 'h',ELM 'e',ELM '
r',ELM '.',ELM '¥n']

a (Data sent) :

[ELM 'I',ELM ' ',ELM 'a',ELM 'a',ELM 'm',ELM ' ',ELM ' ',ELM ' ',ELM ' ',ELM 'h'
,ELM 'h',ELM 'h',ELM 'e',ELM 'e',ELM 'e',ELM ' ',ELM 'a',ELM 'a',ELM 's',ELM ' '
,ELM ' ',ELM 'y',ELM 'y',ELM 'y',ELM 'o',ELM 'o',ELM 'u',ELM 'u',ELM 'u',ELM 'u'
,ELM ' ',ELM 'a',ELM 'r',ELM 'e',ELM ' ',ELM 'h',ELM 'e',ELM 'e',ELM 'e',ELM 'e'
,ELM ' ',ELM '¥n',ELM 'a',ELM 'a',ELM 's',ELM 's',ELM 's',ELM ' ',ELM 'y',ELM 'o
',ELM 'o',ELM 'o',ELM 'o',ELM 'o',ELM 'u',ELM ' ',ELM ' ',ELM 'a',ELM 'a',ELM 'a
',ELM 'a',ELM 'r',ELM 'e',ELM ' ',ELM 'm',ELM 'm',ELM 'e',ELM 'e',ELM 'e',ELM '¥
n',ELM 'a',ELM 'n',ELM 'n',ELM 'd',ELM 'd',ELM 'd',ELM 'd',ELM 'd',ELM ' ',ELM '
 ',ELM 'w',ELM 'w',ELM 'w',ELM 'e',ELM ' ',ELM 'a',ELM 'a',ELM 'r',ELM 'e',ELM '
 ',ELM ' ',ELM 'a',ELM 'l',ELM 'l',ELM ' ',ELM 't',ELM 'o',ELM 'g',ELM 'g',ELM '
e',ELM 'e',ELM 'e',ELM 'e',ELM 't',ELM 'h',ELM 'h',ELM 'h',ELM 'h',ELM 'h',ELM '
h',ELM 'e',ELM 'e',ELM 'r',ELM 'r',ELM 'r',ELM 'r',ELM '.',ELM '.',ELM '¥n',ELM
'¥n',ELM '¥n']

z1 (Noise of sending channel) :

[ELM 0,ELM 0,ELM 1,ELM 0,ELM 0,ELM 1,ELM 1,ELM 1,ELM 0,ELM 1,ELM 1,ELM 0,ELM 1,E
LM 1,ELM 0,ELM 0,ELM 1,ELM 0,ELM 0,ELM 1,ELM 0,ELM 1,ELM 1,ELM 0,ELM 1,ELM 0,ELM
 1,ELM 1,ELM 1,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 1,ELM 1,ELM 1,ELM 0
,ELM 0,ELM 0,ELM 1,ELM 0,ELM 1,ELM 1,ELM 0,ELM 0,ELM 0,ELM 1,ELM 1,ELM 1,ELM 1,E
LM 0,ELM 0,ELM 1,ELM 0,ELM 1,ELM 1,ELM 1,ELM 0,ELM 0,ELM 0,ELM 0,ELM 1,ELM 0,ELM
 1,ELM 1,ELM 0,ELM 0,ELM 0,ELM 1,ELM 0,ELM 1,ELM 1,ELM 1,ELM 1,ELM 0,ELM 1,ELM 0
,ELM 1,ELM 1,ELM 0,ELM 0,ELM 0,ELM 1,ELM 0,ELM 0,ELM 0,ELM 1,ELM 0,ELM 0,ELM 0,E
LM 0,ELM 0,ELM 0,ELM 0,ELM 1,ELM 0,ELM 1,ELM 1,ELM 1,ELM 0,ELM 0,ELM 1,ELM 1,ELM
 1,ELM 1,ELM 1,ELM 0,ELM 1,ELM 0,ELM 1,ELM 1,ELM 1,ELM 0,ELM 1,ELM 0,ELM 1,ELM 1
,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 1,ELM 0,ELM 0,ELM 1,ELM 0,ELM 0,ELM 1,E
LM 0,ELM 1,ELM 1,ELM 1,ELM 0,ELM 1,ELM 0,ELM 1,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM
 1,ELM 1,ELM 1,ELM 1,ELM 0,ELM 0,ELM 1,ELM 1,ELM 0,ELM 0,ELM 0,ELM 1,ELM 1,ELM 0
,ELM 1,ELM 1,ELM 1,ELM 1,ELM 0,ELM 1,ELM 1,ELM 0,ELM 1,ELM 1,ELM 1,ELM 0,ELM 0,E
LM 0,ELM 0,ELM 0,ELM 1,ELM 0,ELM 0,ELM 1,ELM 0,ELM 1,ELM 0,ELM 1,ELM 1,ELM 1,ELM
 0,ELM 1,ELM 0,ELM 1,ELM 0,ELM 0,ELM 0,ELM 1,ELM 0,ELM

b (Data received) :

[ELM 'I',ELM ' ',NG,ELM 'a',ELM 'm',NG,NG,NG,ELM ' ',NG,NG,ELM 'h',NG,NG,ELM 'e'
,ELM ' ',NG,ELM 'a',ELM 's',NG,ELM ' ',NG,NG,ELM 'y',NG,ELM 'o',NG,NG,NG,ELM 'u'
,ELM ' ',ELM 'a',ELM 'r',ELM 'e',ELM ' ',ELM 'h',NG,NG,NG,ELM 'e',ELM ' ',ELM '¥
n',NG,ELM 'a',NG,NG,ELM 's',ELM ' ',ELM 'y',NG,NG,NG,NG,ELM 'o',ELM 'u',NG,ELM '
 ',NG,NG,NG,ELM 'a',ELM 'r',ELM 'e',ELM ' ',NG,ELM 'm',NG,NG,ELM 'e',ELM '¥n',EL
M 'a',NG,ELM 'n',NG,NG,NG,NG,ELM 'd',NG,ELM ' ',NG,NG,ELM 'w',ELM 'e',ELM ' ',NG
,ELM 'a',ELM 'r',ELM 'e',NG,ELM ' ',ELM 'a',ELM 'l',ELM 'l',ELM ' ',ELM 't',ELM
'o',NG,ELM 'g',NG,NG,NG,ELM 'e',ELM 't',NG,NG,NG,NG,NG,ELM 'h',NG,ELM 'e',NG,NG,
NG,ELM 'r',NG,ELM '.',NG,NG,ELM '¥n']

c (Reply sent) :

[ELM 'A',ELM 'A',ELM 'N',ELM 'A',ELM 'A',ELM 'N',ELM 'N',ELM 'N',ELM 'A',ELM 'N'
,ELM 'N',ELM 'A',ELM 'N',ELM 'N',ELM 'A',ELM 'A',ELM 'N',ELM 'A',ELM 'A',ELM 'N'
,ELM 'A',ELM 'N',ELM 'N',ELM 'A',ELM 'N',ELM 'A',ELM 'N',ELM 'N',ELM 'N',ELM 'A'
,ELM 'A',ELM 'A',ELM 'A',ELM 'A',ELM 'A',ELM 'A',ELM 'N',ELM 'N',ELM 'N',ELM 'A'
,ELM 'A',ELM 'A',ELM 'N',ELM 'A',ELM 'N',ELM 'N',ELM 'A',ELM 'A',ELM 'A',ELM 'N'
,ELM 'N',ELM 'N',ELM 'N',ELM 'A',ELM 'A',ELM 'N',ELM 'A',ELM 'N',ELM 'N',ELM 'N'
,ELM 'A',ELM 'A',ELM 'A',ELM 'A',ELM 'N',ELM 'A',ELM 'N',ELM 'N',ELM 'A',ELM 'A'
,ELM 'A',ELM 'N',ELM 'A',ELM 'N',ELM 'N',ELM 'N',ELM 'N',ELM 'A',ELM 'N',ELM 'A'
,ELM 'N',ELM 'N',ELM 'A',ELM 'A',ELM 'A',ELM 'N',ELM 'A',ELM 'A',ELM 'A',ELM 'N'
,ELM 'A',ELM 'A',ELM 'A',ELM 'A',ELM 'A',ELM 'A',ELM 'A',ELM 'N',ELM 'A',ELM 'N'
,ELM 'N',ELM 'N',ELM 'A',ELM 'A',ELM 'N',ELM 'N',ELM 'N',ELM 'N',ELM 'N',ELM 'A'
,ELM 'N',ELM 'A',ELM 'N',ELM 'N',ELM 'N',ELM 'A',ELM 'N',ELM 'A',ELM 'N',ELM 'N'
,ELM 'A']

z2 (Noise of replying channel) :

[ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,E
LM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM
 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0
,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,E
LM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM
 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0
,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,E
LM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM
 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0
,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,E
LM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM
 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0
,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,E
LM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM
 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0,ELM 0
,ELM 0,ELM 0,ELM 0,ELM 0,ELM

d (Reply received) :

[ELM 'A',ELM 'A',ELM 'N',ELM 'A',ELM 'A',ELM 'N',ELM 'N',ELM 'N',ELM 'A',ELM 'N'
,ELM 'N',ELM 'A',ELM 'N',ELM 'N',ELM 'A',ELM 'A',ELM 'N',ELM 'A',ELM 'A',ELM 'N'
,ELM 'A',ELM 'N',ELM 'N',ELM 'A',ELM 'N',ELM 'A',ELM 'N',ELM 'N',ELM 'N',ELM 'A'
,ELM 'A',ELM 'A',ELM 'A',ELM 'A',ELM 'A',ELM 'A',ELM 'N',ELM 'N',ELM 'N',ELM 'A'
,ELM 'A',ELM 'A',ELM 'N',ELM 'A',ELM 'N',ELM 'N',ELM 'A',ELM 'A',ELM 'A',ELM 'N'
,ELM 'N',ELM 'N',ELM 'N',ELM 'A',ELM 'A',ELM 'N',ELM 'A',ELM 'N',ELM 'N',ELM 'N'
,ELM 'A',ELM 'A',ELM 'A',ELM 'A',ELM 'N',ELM 'A',ELM 'N',ELM 'N',ELM 'A',ELM 'A'
,ELM 'A',ELM 'N',ELM 'A',ELM 'N',ELM 'N',ELM 'N',ELM 'N',ELM 'A',ELM 'N',ELM 'A'
,ELM 'N',ELM 'N',ELM 'A',ELM 'A',ELM 'A',ELM 'N',ELM 'A',ELM 'A',ELM 'A',ELM 'N'
,ELM 'A',ELM 'A',ELM 'A',ELM 'A',ELM 'A',ELM 'A',ELM 'A',ELM 'N',ELM 'A',ELM 'N'
,ELM 'N',ELM 'N',ELM 'A',ELM 'A',ELM 'N',ELM 'N',ELM 'N',ELM 'N',ELM 'N',ELM 'A'
,ELM 'N',ELM 'A',ELM 'N',ELM 'N',ELM 'N',ELM 'A',ELM 'N',ELM 'A',ELM 'N',ELM 'N'
,ELM 'A']

output (Output Data) :

[ELM 'I',ELM ' ',ELM 'a',ELM 'm',ELM ' ',ELM 'h',ELM 'e',ELM ' ',ELM 'a',ELM 's'
,ELM ' ',ELM 'y',ELM 'o',ELM 'u',ELM ' ',ELM 'a',ELM 'r',ELM 'e',ELM ' ',ELM 'h'
,ELM 'e',ELM ' ',ELM '¥n',ELM 'a',ELM 's',ELM ' ',ELM 'y',ELM 'o',ELM 'u',ELM '
',ELM 'a',ELM 'r',ELM 'e',ELM ' ',ELM 'm',ELM 'e',ELM '¥n',ELM 'a',ELM 'n',ELM '
d',ELM ' ',ELM 'w',ELM 'e',ELM ' ',ELM 'a',ELM 'r',ELM 'e',ELM ' ',ELM 'a',ELM '
l',ELM 'l',ELM ' ',ELM 't',ELM 'o',ELM 'g',ELM 'e',ELM 't',ELM 'h',ELM 'e',ELM '
r',ELM '.',ELM '¥n']
```

図５．通信システムの実行例

が示されたことと，上で定義された各関数が半順序⊑に関して連続であることによって，input ∈ A^{∞} の場合も

output = input

が成り立つ．

以上，（Ⅰ），（Ⅱ），（Ⅲ）より，任意の input ∈ STREAM(A) に対して

output = input

が満足される．

この通信システムの例題をＭｉｒａｎｄａで記述したものを図４に示す．ストリームは，抽象データ型として定義しており，上で用いた演算ないし定数 fst, rest, &, ε をそれぞれ，fst, rest, cons, empty と定義した．このＭｉｒａｎｄａプログラムの実行例を図５に掲げる．送信すべきメッセージの系列と正しく受信されたメッセージの系列とが全く等しいことが確認できる．また，通信媒体で発生した擾乱の様子や，メッセージの再送の様子を観察できる．

３．２　哲学者の食事問題

哲学者の食事問題をストリーム処理関数のネットワークとして仕様記述した例を掲げる．図６に，その構成を示す．図７には，Ｍｉｒａｎｄａプログラムとして記述した仕様を掲げる．この例では，前の通信システムの場合とは異なって，ストリーム型を新たに定義せずに，Ｍｉｒａｎｄａが提供するリスト型をそのままストリームとして用いている．

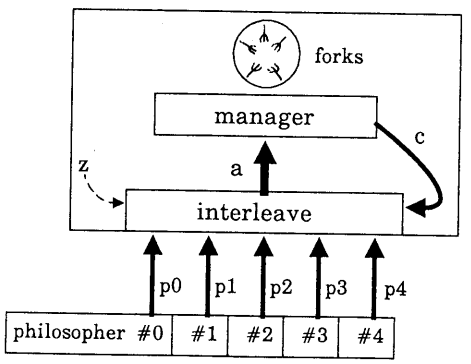

図６．哲学者の食事問題のシステム構成

各哲学者は，思索(Thinking)，食事の要求(Request)，食事(Eating)，食事の終了(Release) という事象を繰り返して発生させようとする．manager は，フォークの状態と食事をするのを待っている哲学者の状態とを見て，次に発生しうる事象を制御するための情報を interleave に送る．interleave は，発生可能な事象の中から一つを選んで，それを発生させる．どれを選ぶかは，前掲の通信システムの例における通信媒体の擾乱と同様に，interleave に対する入力ストリーム z によって指定する．interleave が発生させるストリーム a は，システム内で発生した事象の系列となっており，これがシステムの振舞いの履歴を表現していると見なす．

この例題に対する要求としては，共有資源であるフォークに対して相互排除を行なうこと，デッドロックに陥らないこと，スタベーションをおこさないことなどが考えられる．これらは，システムの振舞いを表わすストリーム a についての次のような記述として与えることができる．

(1) liveness property

食事をしたいという要求を出した哲学者は，いつかは必ずその要求が満足されて実際に食事をするということを，次のように表わす．システムの振舞いの履歴を表わすストリーム a 中に発生した哲学者 i からの食事の要求（Request i）の回数と，その哲学者 i の食事（Eating i）の回数とが等しい．このことは，ストリーム a に対する条件として

$$\forall i.\ 0 \leq i \leq 4 \supset \#((\text{Request } i), a) = \#((\text{Eating } i), a)$$

と記述される．あるいは，システムの全履歴を表わすストリーム a の部分ストリーム q (⊑a) に対する不変条件として，次のように記述することもできる．

$$\forall i, q.\ \exists p.\ 0 \leq i \leq 4 \wedge q \sqsubseteq a \supset q \sqsubseteq p \sqsubseteq a \wedge \#((\text{Request } i)\ q) = \#((\text{Eating } i)\ p)$$

(2) safety property

フォークの利用を相互排除するために，哲学者 i が食事をしている間，その両隣の哲学者は食事をすることができない．このことを，ストリーム a に対する条件として

$$\forall i, p, q.\ \exists r, s.$$

```
|| actions of philosophers

action ::= Thinking num | Request num
         | Eating num | Release num

isrequest (Thinking i) = False
isrequest (Request i) = True
isrequest (Eating i) = False
isrequest (Release i) = False

isrelease (Thinking i) = False
isrelease (Request i) = False
isrelease (Eating i) = False
isrelease (Release i) = True

who (Thinking i) = i
who (Request i) = i
who (Eating i) = i
who (Release i) = i

right i = (i-1) mod 5
left  i = (i+1) mod 5

|| philosopher

philosopher :: num -> [action]
philosopher i
    = Thinking i : Request i : Eating i : Release i
      : philosopher i

|| forks

fork ::= INITIAL | GRASP fork num

grasp f n = GRASP f n
release (GRASP f j) i = f, i=j
                      = grasp INITIAL j, f=GRASP INITIAL i

isok INITIAL i = True
isok (GRASP INITIAL j) i
    = False, i=(right j) \/ i=(left j)
    = True, otherwise
isok (GRASP (GRASP INITIAL j) k) i = False

|| queue

insert i q = i : q

remove i q = [], q=[]
           = tl q, hd q=i
           = hd q : remove i (tl q), otherwise

isinhighpq i (q1,q2) = isin i q1

isblocked i (BLK (q1,q2)) = isin i (q1++q2)

insertpq i (q1,q2)
    = (q1, (insert i q2)), isin (left i) q1 \/ isin (right i) q1
    = ((insert i q1), q2), otherwise

removepq i (q1,q2)
    = ((insert(left i)(insert(right i)(remove i q1))),
       (remove(left i)(remove(right i) q2))),
         isin (left i) q2 & isin (right i) q2
    = ((insert (left i) (remove i q1)), (remove (left i) q2)),
         isin (left i) q2
    = ((insert (right i) (remove i q1)), (remove (right i) q2)),
         isin (right i) q2
    = ((remove i q1), q2), otherwise

|| control information from manager

control ::= SIG num | BLK ([num],[num])

isin i s = False, s=[]
         = True, i=hd s
         = isin i (tl s), otherwise

issig (SIG i) = True
issig (BLK s) = False

isblk (SIG i) = False
isblk (BLK s) = True
```

```
|| manager

manager :: fork -> [action] -> ([num],[num]) -> [control]
manager f e q
    = [], e=[]
    = reqman f e q, isrequest (hd e)
    = relman f e q, isrelease (hd e)
    = BLK q : manager f (tl e) q, otherwise

reqman f e q
    = BLK (insertpq(who(hd e))q)
      : manager f (tl e) (insertpq (who (hd e)) q),
        ~isok f (who (hd e))
    = BLK (insertpq(who(hd e))q)
      : manager f (tl e) (insertpq (who (hd e)) q),
        isinhighpq (left(who(hd e))) q \/ isinhighpq (right(who(hd e))) q
    = SIG (who(hd e))
      : manager (grasp f (who (hd e))) (tl e) q,
        otherwise

relman f e q
    = SIG (right(who(hd e)))
      : relman2 (grasp(release f (who(hd e))) (right(who(hd e))))
                (tl e)   (removepq (right(who(hd e))) q)  (who(hd e)),
        isinhighpq (right(who(hd e))) q
        & isok (release f (who(hd e))) (right (who(hd e)))
    = relman2 (release f (who(hd e))) e q (who(hd e)),
        otherwise

relman2 f e q w
    = SIG (left w)
      : manager (grasp f (left w))
                (tl e)   (removepq (left w) q),
        isinhighpq (left w) q & isok f (left w)
    = BLK q : manager f (tl e) q,
        otherwise

|| interleave

interleave z c a0 a1 a2 a3 a4
    = [], a0=[] & a1=[] & a2=[] & a3=[] & a4=[]
    = interleavesig z c a0 a1 a2 a3 a4, issig (hd c)
    = interleaveblk z c a0 a1 a2 a3 a4, isblk (hd c)

interleavesig z c a0 a1 a2 a3 a4
    = hd a0 : interleave z (tl c) (tl a0) a1 a2 a3 a4,
        hd c=SIG 0
    = hd a1 : interleave z (tl c) a0 (tl a1) a2 a3 a4,
        hd c=SIG 1
    = hd a2 : interleave z (tl c) a0 a1 (tl a2) a3 a4,
        hd c=SIG 2
    = hd a3 : interleave z (tl c) a0 a1 a2 (tl a3) a4,
        hd c=SIG 3
    = hd a4 : interleave z (tl c) a0 a1 a2 a3 (tl a4),
        hd c=SIG 4

interleaveblk z c a0 a1 a2 a3 a4
    = hd a0 : interleave (tl z) (tl c) (tl a0) a1 a2 a3 a4,
        hd z=0 & a0~=[] & ~isblocked 0 (hd c)
    = hd a1 : interleave (tl z) (tl c) a0 (tl a1) a2 a3 a4,
        hd z=1 & a1~=[] & ~isblocked 1 (hd c)
    = hd a2 : interleave (tl z) (tl c) a0 a1 (tl a2) a3 a4,
        hd z=2 & a2~=[] & ~isblocked 2 (hd c)
    = hd a3 : interleave (tl z) (tl c) a0 a1 a2 (tl a3) a4,
        hd z=3 & a3~=[] & ~isblocked 3 (hd c)
    = hd a4 : interleave (tl z) (tl c) a0 a1 a2 a3 (tl a4),
        hd z=4 & a4~=[] &  ~isblocked 4 (hd c)
    = interleaveblk (tl z) c a0 a1 a2 a3 a4, otherwise

|| system configuration

a = interleave z c p0 p1 p2 p3 p4
c = BLK ([],[]) : manager INITIAL a ([],[])
p0 = philosopher 0
p1 = philosopher 1
p2 = philosopher 2
p3 = philosopher 3
p4 = philosopher 4
```

図７．哲学者の食事問題のＭｉｒａｎｄａプログラム

図８．哲学者の食事問題の実行例

```
0≦i≦4 ∧ p⊑a
 ∧ p=q^r^<Release i> ∧ q=s^<Eating i>
 ∧ #((Release i) r) = 0  ⊃
   ⊃ #((Eating (right i)) r) = 0
      ∧ #((Eating (left i)) r) = 0
```

と記述する．

この例では，前の例とは異なり，図7に掲げた仕様が上記の条件を満足することを証明するのではなくて，実行可能な仕様であるＭｉｒａｎｄａプログラムを実際に動作させるテストによって確認した．それによって，当初の仕様［荒木-90］に不備があったことを発見できた．図7のＭｉｒａｎｄａプログラムの実行結果の例を図8に示す．

前の通信システムの例では，システムの中を実際に流れる具体的データを構成要素とするストリームに関して記述と証明を行なった．一方，この哲学者の食事問題では，システム中で発生する事象という仮想的なデータを構成要素とするストリーム ａ を，システムの振舞いの履歴を見なして，それについての条件としてシステムに対する要求を記述した．システムの仕様記述を行なう際に，このようにストリームの構成要素の抽象度を，対象とする問題や記述のレベルに応じて適宜選定することができる．

4．考察

我々は，分散型ソフトウェアの開発過程として，プロトタイピングや形式的仕様に基づく方法［松本-87］や，図9に示す方法［Yeh-89］などを想定している．その場合に，開発の対象となる分散型システムを，ストリームによって結合された並行プロセスのネットワークとしてモデル化し，実行可能仕様記述言語としてＭｉｒａｎｄａを用いてそれを仕様記述し，評価分析を行なって，最終的な分散型ソフトウェアを作成する．仕様の評価分析は，形式的な基礎付けに基づいて静的に行なうこともあろうし，仕様を実行させて動的に行なうこともあろう．

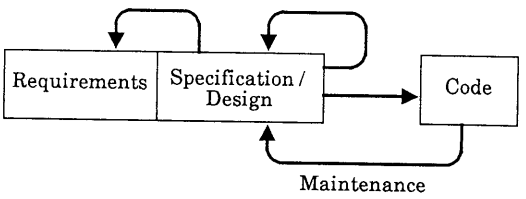

図9．ソフトウェア進化の過程
（文献[Yeh-89]による）

実行可能な仕様は，システム開発におけるプロトタイプとも見なすことができる．実際に実行させてみてその動作の様子を観察することによって，対象とするシステムの理解も深まり，要求の形成や仕様の明確化に役立つ．実行可能仕様記述言語に対する要求として，実現性，形式性，構築性，モデル性等が挙げられている．［二木-87］Ｍｉｒａｎｄａは，実現性と形式性とを兼ね備えており，プログラミング言語としての構築性をも持っている．また，本稿で述べたストリームに基づく並行動作システムのモデル化にも良く適合する．

実行可能な仕様を実際に実行させることによって，その仕様についての動的な解析を行なうことができる．よって，従来のプログラムのテストと同様な手法によって，仕様の妥当性を示すことができる．仕様をテスト実行させた時の結果が正しいものであるかどうかは，形式的に記述された要求に照らして判定することができる．

本稿で述べた記述の仕方では，並行プロセスをインターリーブさせて逐次化する際のスケジューリングを陽に与える．どのような（公平な）スケジューリングを行なっても，システムに対する要求を満足することを示すことがここでの目的の一つである．そうすることによって，分散型システムの非決定的な振舞いを考慮に入れた仕様記述となっている．このため，テスト実行で仕様の妥当性を示す場合には，スケジューリングのためのテストケースをどのように選択すればよいかということが問題となる．その反面，スケジューリングを陽に行なうために動作の再現性を保証することができることは，分散型システムのテストにとって有益である．［Tai-87］

5．おわりに

実際に，いくつかの例題について，ストリーム処理関数のネットワークとして分散型システムを記述してみた結果，分散型システムの開発における要求形成や仕様記述や設計支援に対して有効であると期待できる．厳密に定義された関数の合成によってシステムを構成するために，モジュール化やトップダウン設計も自然に行なうことができる．今後は，実用規模の問題への適用を試みながら，本方法の有用性を実証するとともに，分散型システム開発方法論の確立とその実用化を目指す．併せて，本方法に基づいて分散型システムの要求定義，仕様記述，設計を支援するシステムの構築も計画中である．また，抽象的な仕様から具体的なプログラムを得る方法についても検討しなければならな

い．

謝辞　抽象データ型として代数的に定義されたストリームによる並行動作システムの仕様記述法を御教示頂いた Passau 大学 Manfred Broy 教授に謝意を表します．また，日頃ご討論頂く九州大学の皆様に感謝致します．

参考文献

[Broy-87] Broy, M.: "Specification and Top-Down Design of Distributed Systems," Journal of Computer and System Sciences, Vol.34, pp.236-265, 1987.

[Broy-88] Broy, M.: "Requirement and Design Specification for Distributed Systems," Proc. CONCURRENCY 88, Lecture Notes in Computer Science, Vol.335, pp.33-62, 1988.

[RSL] Research Software Limited: Miranda System Manual, 1987 and 1989.

[Sunshine-79] Sunshine, C.: "Formal Techniques for Protocol Specification and Verification," IEEE Computer Magazine, Vol.12, No.9, pp.20-27, 1979.

[Tai-87] Tai, K.-C. and Ahuga, S.: "Reproducible Testing of Communication Software," Proc. COMPSAC 87, pp.331-337, 1987.

[Turner-85a] Turner, D.A.: "Functional Programs as Executable Specifications," Mathematical Logic and Programming Languages (C.A.R. Hoare and J.C. Shepherdson, eds.), Prentice/Hall, pp.29-54, 1985.

[Turner-85b] Turner, D.A.: "Miranda: A Non-strict Functional Language with Polymorphic Types," Lecture Notes in Computer Science, Vol.201, Springer-Verlag, pp.1-16, 1985.

[Yeh-89] Yeh, R.T.: "An Alternative Paradigm for Software Evolution," Modern Software Engineering (P.A. Ng and R.T. Yeh, eds.), Van Nostrand Reinhold, pp.7-22, 1989.

[荒木-90] 荒木啓二郎：ストリーム処理関数による並行動作システムの仕様記述と設計，情報処理学会第４０回全国大会，1990.

[田中-88] 田中二郎：関数型プログラムにおけるストリーム計算，情報処理，Vol.29, No.8, pp.836-844, 1988.

[二木-87] 二木厚吉：実行可能仕様に基づく変換プログラミング，情報処理，Vol.28, No.7, pp.906-912, 1987.

[松本-87] 松本吉弘：ソフトウェアに対する要求の形成，情報処理，Vol.28, No.7, pp.853-861, 1987.