2R－1

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

荒木　啓二郎
九州大学工学部情報工学科

1．はじめに

並行動作システムの要求定義並びに仕様記述をストリームとストリーム処理関数[1]を用いて形式的枠組みのもとに行なうM．Broyの方法[2]を適用した例と，その方法に基づく並行動作システムの設計において関数型プログラミング言語Miranda[3,4]を実行可能な仕様記述言語として利用する方法について述べる．

2．ストリーム処理関数による仕様記述

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をストリーム処理関数の系として構成することができる．Broy[2]は，並行動作システムをこのようなストリームによって結合されたネットワークとしてモデル化し，その振舞いをインターリービングモデルに基づいて，代数的仕様記述の枠組みで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て記述する方法を提案した．並行動作システムの振舞いに対する要求は，ストリームに関する述語として表現され，並行動作システムはストリーム処理関数によって構成される．ストリーム処理関数の系として仕様記述された並行動作システムが，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．

並行動作システムの振舞いを表わすストリームとしては，並行動作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．これによって，システムのモデル化や記述の抽象度を自由に選定できる．

3．実行可能仕様としてのMirandaプログラム

関数型プログラミング言語Miranda[3]は，代数的データ型や無限データ構造等の機能を持つために，前述の仕様との親和性が高い．Mirandaが提供するリストによって無限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無限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並行動作システムを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．

並行動作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．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，テストによって動作確認を行なった後に，改めて仕様と見なして，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．

4．例題

哲学者の食事問題を本方法に基づいて仕様記述した例を掲げる．図1に，ストリームによって結合されたネットワークとして表わしたシステムの構成を示す．図2には，Mirandaプログラムとして記述した仕様の主要部分を掲げる．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を表現していると見なす．

この問題に対する要求としては，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
 ∀i. 0≦i≦4 ⊃
        #((Request i), a) = #((Eating i), a)
```

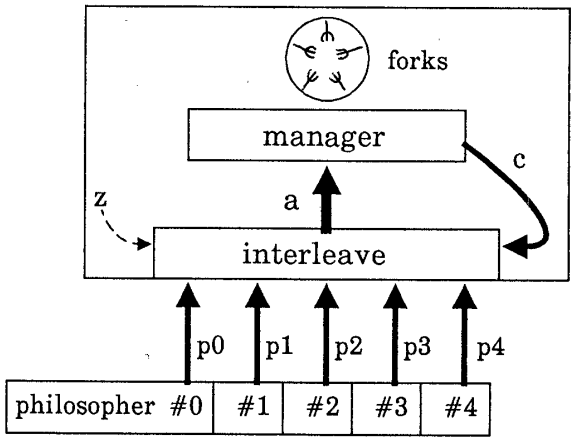

図1．哲学者の食事問題のシステム構成

Specification and Design of Concurrent Systems with Stream Functions
Keijiro ARAKI
Kyushu University

(2) safety property

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

$0 \leqq i \leqq 4 \wedge p \sqsubseteq a$

$\wedge\ p=q^\frown r^\frown\langle Release\ i\rangle \wedge\ q=s^\frown\langle Eating\ i\rangle$

$\wedge\ \#((Request\ i)\ r) = 0 \supset$

$\#((Eating\ (right\ i))\ r) = 0$

$\wedge\ \#((Eating\ (left\ i))\ r) = 0$

ここで，#(e,a) は，ストリームａ中に現われるｅの個数を表わす．〈e〉は，ｅのみからなる長さ１のストリームである．また，a⊑b は，ストリームａがストリームｂの前置部分ストリーム(prefix)であることを表わし，a^b は，ａとｂとの連接を表わす．

図１に掲げた仕様がこれらの性質を満足することの証明は，形式的枠組みに沿って行なう．（但し，スケジューリングが公平であることなどを仮定する必要がある．）

５．おわりに

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．[5] 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．[4] 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．

実際に，いくつかの問題に本方法を適用した結果，その有用性に対する見通しが得られた．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，その実用化を目指す．併せて，本方法に基づいて並行動作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．

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．

参考文献

[1] 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.

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

[3] 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.

[4] Miranda: Miranda System Manual, Research Software Limited, 1987.

[5] 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.

```
||  Dining Philosophers

|| actions of philosophers

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

|| philosopher

philosopher i
    = Thinking i : Request i : Eating i : Release i
      : philosopher i

|| control information from manager

control ::= SIG num | BLK [num]

isblocked i (BLK s) = isin i s

|| manager

manager :: fork -> [action] -> [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
    = SIG (who(hd e))
      : manager (GRASP f (who (hd e))) (tl e) q,
        isok f (who (hd e))
    = BLK (insert(who(hd e))q)
      : manager f (tl e) (insert (who (hd 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)  (remove (right(who(hd e))) q) (who(hd e)),
        isin (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)  (remove (left w) q),
        isin (left w) q & isok f (left w)
    = BLK q : manager f (tl e) q,
        otherwise
```

図２．Ｍｉｒａｎｄａによる記述

```
|| forks

fork ::= INITIAL | GRASP fork num
       | RELEASE fork num

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

|| 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
```