# 基于 Monad 的纯函数式语言通道系统设计

袁华强[1] 孙永强[2]
(东莞理工学院计算机科学与技术系 软件研究所 东莞523106)[1]
(上海交通大学计算机科学与工程系 上海200030)[2]

**摘 要** 本文通过状态转换器来定义 I/O 的文件系统,并用非确定性 Monad 描述了操作系统的进程网,从而给出了通道系统的语义。

**关键词** 纯函数式语言,Monad

## The Design of File System in the Pure Functional I/O Based on Monads

YUAN Hua-Qiang SUN Yong-Qiang
(Department of Computer Science and Technology, Dongguan University of Technology, Dongguan 523106)
(Institute of Software, Dongguan University of Technology, Dongguan 523106)
(Department of Comuter Science and Engineering, Shanghai Jiaotong University, Shanghai 200030)

**Abstract** This paper discusses the definition of a complete functional I/O file system based on the state transformer. Using a nondeterministic Monad, we construct the process networks of operating systems. And the semantics of channel system have been given for using.

**Keywords** Pure function language, Monad

Monad 是范畴论中一个重要的概念,E. Moggi 在文[6,7]提出用 Monad 来构造计算的语义,其主要思想是把一个类型的值的对象与计算的对象区分开来,并把一个类型的程序的指称语义看成是该类型计算的对象的元素。

P. Wadler 在文[8]采用了 E. Moggi 的思想,把 Monad 作为构造纯函数式语言的工具,构造出诸如错误处理、状态、IO 等非函数式语言的特征,引起了人们的广泛关注。

本文采用非确定性 Monad 来构造纯函数式语言的通道系统,既保持了纯函数式语言的特征,又融入了非纯函数式语言的特征。

## 1 状态的定义

为了更好地设计纯函数语言的 I/O 系统,我们将 I/O 的类型定义为[1]:type IO a=ST RealWorld a。状态 RealWorld 如何定义要看具体的系统而定。在我们设计的 I/O 系统中,会将状态看成是整个操作系统的状态,并把操作系统状态分为两部分:文件系统和通道系统,本文只讨论通道系统。

```
Type RealWorld=(FileSystem,ChannelSystem)
Type FileSystem=String->Response
Type ChannelSystem=(Ics,Ocs)
Type Ics=String->(Agent,Open)
Type Ocs=string->Response
Type Agent=(FileSystem,Ocs)->Response
Type Open=Pid->bool
Type PId=Int
Type PList=[(Pid,[Request->Response])]
```

操作系统与进程的关系描述如图1所示。

对通道系统而言,我们把它看成这样的函数,对给定的文件名(类型为 String),判断它是否存在,若存在则返回成功信息,若不存在,则返回错误信息。

在非 Monad 的方式下,OS 的类型为:

OS::TagReqList->TagRespList

在 Monad 方式下,我们将 OS 的类型定义为:

OS::TagReaList->IO TagRespList

其中 type TagRespList=[(Pid,Response)]

type TagReqList=[(Pid,Request)]

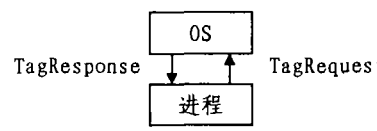


图1 操作系统与进程的关系图

操作系统是由非确定性函数 OS 来构造的,给定程序表 Plist,OS 为:

```
OS TagReqList=\state->(TagRespList,state')
TagReqList=merge[zip[Pid,Pid,…,](proc(untag Pid
                tagRespList))|(Pid,proc<-pList)]
```

## 2 非确定性 Monad

在上面,我们提到操作系统是由非确定性函数 OS 来构造的,接下来,我们将对此进行详细的研究。用纯函数式语言描述操作系统,其中一个重要问题就是如何处理非确定性问题。当我们把两个流归并成一个流时,由于系统对流的输入速度是一无所知的,究竟哪一个流的元素先到系统是不能确定的,这就有一个非确定性问题,人们一般对这个问题的解决是引入了一个非确定性算子 almost_merge:

```
almost_merge(a:rest) b=a:(almost_merge rest b)
```

袁华强 博士,副教授,主要研究方向为新型语言及其支撑环境.孙永强 教授,博士导师,主要研究方向为软件工程、并行处理.

```
almost_merge a (b:rest)=b:(almost_merge a rest)
almost_merge [] rest=rest
almost_merge rest []=rest
```

下面,我们将定义一个非确定性 Monad,然后再根据这个非确定性 Monad 来定义非确定性进程网和非确定性归并操作 merge。

**定义1** 非确定性 Monad

```
type M a=(a)
map::(a->b)->M a->M b
map f S=(f x|x∈S)
join::(M(M a))->M a
join S=U(Si|Si∈S)
unit::a->M a
unit a=(a)
```

其中 M 是类型构造子,(a)表示类型 a 的集合。

**定理1** (map,unit,join)构成了一个 Monad。

证明:(1)先证 map 是一个函子。

Map id S=(id x|x∈S)=(x|x∈S)=id S

所以 map id=id

map(f.g)S=((f.g)x|x∈S)=(f(g(x))|x∈S)

((map f).(map g))S=map f(map g S)=map f(g x|x∈S)=(f(g(x))|x∈S)

所以 map(f.g)=(map f).(map g),因而 map 是一个函子。

(2)再证 unit,join 是自然变换。

((map f).unit)a=map f(unit a)=map f(a)=(f a)

(unit.f)a=unit(f a)=(f a)

所以(map f).unit=unit.f

((map f).join)S=map f(join S)=map f U(Si|Si∈S)=U(map f Si|Si∈S)

=U((f x|x∈Si)|Si∈S)

(join.map(map f))S=join(map(map f)S)

=join(map f Si|Si∈S)=join((f x|x∈Si|Si∈S)

=U((f x|x∈Si|Si∈S)

所以(map f).join=join.map(map f)

所以 unit,join 是自然变换。

(3)最后证明 map,unit,join 可变换。

(join.map join)S=join(map join S)=join((join Si|Si∈S)

=U(join Si|Si∈S)

(join.join)S=join(join S)=join(U(Si|Si∈S))=U(join Si|Si∈S)

所以 join.map join=join.join

(join.unit)S=join(unit S)=join(S)=S

所以 join.unit=id

(join.map unit)S=join(map unit S)=join((unit Si|Si∈S))

=join((Si)|Si∈S)=S

所以 join.map unit=id

因而 map,unit,join 可变换。

综上所述,(map,unit,join)构成了一个 Monad。证毕

下面用非确定性 Monad 来定义非确定性算子 merge:

```
bias,merge::Stream a->Stream a->M(Stream a)
bias[]ys=unit ys
bias(x:xs)ys=map(x:)(merge xs ys)
merge xs ys=(bias xs ys)∪(bias ys xs)
```

例如,merge[1,2][3,4]=([1,2,3,4],[1,3,2,4],[3,1,2,4],[3,1,4,2],[3,4,1,2])

给定下列进程网:

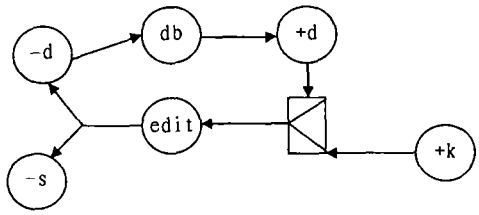


图2 操作系统进程网

其中+表示给流的每个元素加上一个标志,它是由函数 tag 来完成的,-表示给流的每一个元素去掉一个标志,它是由 untag 来完成的,k 表示键盘,s 表示屏幕,edit 表示编辑器,db 表示数据库系统,下面我们给出 tag,untag 的定义:

```
type tagged a=(Char,a)
tag::Char->Stream a->Stream (tagged a)
tag c[]=[]
tag c(m:ms)=(c,m):tag c ms
untag::Char->Stream(tagged a)->Stream a
untag c[]=[]
untag c((c',m):ms)=m:untag c ms if c=c'
                  =untag c ms therwise
```

例如:tag k[1,2,3]=[(k,1),(k,2),(k,3)]

untag D[(D,1),(k,2),(D,3)]=[1,3]

下面就是用非确定性 Monad 描述的该进程网:

```
screen=map(untag 's')edits
dbs=map db(map(untag 'd')edits)
edits=map edit joint(map(merge(tag 'k' keyboard))(map(tag 'd')dbs)
```

下面,我们将用非确定性 Monad 来描述一个操作系统内核,它支持进程调度和通讯:

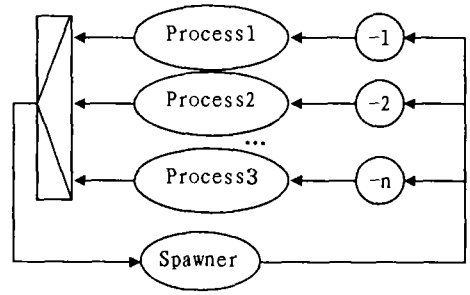


图3 非确定性 Monad 内核图

在其输入流中,进程可以有两种信号:一种是带有进程标志的数据,另一种是要求 Spawner 建立新进程的信号。其类型为:

data message=Tag Int value|Spawn(Int->[message]->[meaaage])

建立一个新进程后,其进程网变成:

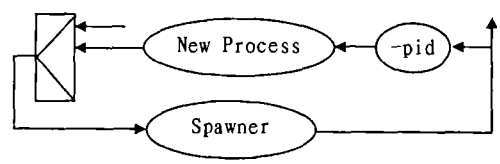


图4 新进程的网络变化

下面就是用非确定性 Monad 描述的非确定性进程调度:

```
Spawner nextid(tag pid v:ms)
=map(Tag pid v:)Spawner ms
Spawner nextid(Spawn f:ms)
=future
```

```
where
   nesproc=map(f nextpid)(map(untag nextid)future)
   future=join(man(Spawner(nextid t1))
          (join(map(merge ms)newproc)))
system=join(map(Spawner1)
        (join(map(merge boot)system)))
```

## 3 通道系统要求

### 3.1 读通道

ReadChan name

功能：打开通道 name 以便输入。若成功，则返回通道的内容，若通道不存在，则返回 Failure(SearchError String)，所有其它类型的错误信息为 Failure(ReadError String)。

一旦 ReadChan 已打开了一个特定的通道，同样的通道就不再为其他请求打开，所以在要求某个通道时，必须检测该通道是否存在，以及该通道是否已被打开，这个工作由 TestChan 来完成：

```
TestChan1::TagReqList->IO TagRespList
TestChan1[(n,ReadChan name)]
=\(fs,(ics,ocs))->
   let ack=(n,if snd(ics name)
               then Failure(OtherError"Channel already open\n")
               else(fst(ics name))(fs,ocs)
   in([ack],(fs,(ics,ocs)))
```

输入通道的类型为 Ics=String->(Agent,Open)，那么 ics name 的类型为(Agent,Open)，snd(ics name)得到的类型为 Open，它判断输入通道是否打开，fst(ics name)的类型为 Agent，它判断该通道是否已为某个进程打开，(fst(ics name))(fs,ocs)的类型为 Response。

测试完通道以后，若输入通道没有被打开，则打开此通道，修改系统的状态，若已经打开，则不修改系统状态，这个工作由 newics 完成：

```
newics::String->TagRespList->IO TagRespList
newics name resp
=\(fs,(ics,ocs))->(resp,(fs,(update ics name(fcs(ics name),
                    update open n true),ocs))
```

那么，ReadChan name 的语义为：

```
os[]=unitSt[]
os(n,ReadChan name);es
=TestChan1[(n,ReadChan name)]          'bindST' \a->
  newics name a                        'bindST' \b->
  os es                                'bindST' \c->
unitST(a++c)
```

### 3.2 写通道

AppendChan name String

功能：把 String 写入通道 name 中，若此通道系统不存在，返回 Failue(SearchError String)。与 ReadChan 类似，首先要检查通道是否存在，但是不要检测通道是否已被打开，这个功能由 TestChan2来完成：

```
TestChan2::TagRespList->IO TagRespList
TestChan2[(n,AppendChan name String)]
=\(fs,(ics,ocs))->
  let ack=
        (n,case (ocs name)of
              Failue msg -> Failue (SearchError "Nonexist
                    channel")
              Str ochan->Success)
        In([ack],(fs,(ics,ocs))
```

然后修改通道系统：

```
newocs::String->TagRespList->IO TagRespList
newocs name resp
=\(fs,(ics,ocs))->
  (resp,(fs,(ics,case(ocs name)of
              Failure msg->ocs
              Str ochan -> update ocs name(str(ochan++
                  contents))))
```

那么，AppendChan 的语义为：

```
os[]=unitST[]
os(n,AppendChan name contents);es
=TestChan2[(n,AppendChan name contents)]  'bindST' \a->
  newocs name a                           'bindST' \b->
  os es                                   'bindST' \c->
unitST(b++c)
```

**结论** 非确定性 Monad 在操作系统、交互式系统以及并行算法中有广泛的应用，但是纯函数式语言缺乏处理非确定性的能力。本文通过使用非确定性 Monad 来描述操作系统的进程网，很好地解决纯函数式语言通道系统的问题。

## 参考文献


1 袁华强，肖倩，孙永强. 基于非确定 Monad 的纯函数式树搜索算法. 软件学报，1997(8. 增刊)：189～193
2 袁华强，孙永强. 纯函数式 I/O 的操作语义. 计算机学报，1998，21(11)：1009～1014
3 石跃祥，袁华强，孙永强，陈静. 函数式语言中的赋值语句. 软件学报，1999，10(3)
4 石跃祥，袁华强. 纯函数式语言中的状态转换器与调用. 湘潭大学学报，2000，22(3)：25～29
5 袁华强. 函数式 I/O 系统的实现与研究：一种 Monad 方法：[博士论文]. 上海交通大学，1996
6 Moggi E. Computational Lamda-calculus and monads. In：Proc Symposium on Logic in Computer Science，CA，1989. 14～24
7 Moggi E. Notions of computation and monads. Information and Computation，1991，93：55～97
8 Wadler P. The essence of functional programming. In：Proc ACM Symposium on Principles of Programming Languages，NM，1992. 1～14


(上接第162页)

MPI 实现间的组通信操作，具有较强的可扩展性。消息传递性能优于 PVMPI，IMPI 的体系结构和功能正朝着融入 GRID 系统的方向发展。

FT-MPI 通过检查点和回卷技术以及通信体重构技术实现了静态 MPI 的动态容错应用，其高可靠性以及其使用本机优化组通信机制的高效通信功能为下一代 Peta-flop 系统、大规模广域网格系统提供了技术基础。

在 MPPS 系统、NOW 系统和 GRID 系统分布式计算应用中，可根据系统和应用的不同特点和要求选用不同的消息传递界面。

## 参考文献


1 Geist A，Beguelin C，et al，PVM：Parallel Virtual Machine. Boston：MIT Press，1994
2 PVM Documentation. Available at：http://www.epm.ornl.gov/pvm
3 Gropp W，Lusk E，et al. Using MPI-2：Advanced Features of the Message Passing Interface，firsted.，MIT Press，Cambrdge，MA，2000
4 Message Passing Interface Forum. MPI：A message -passing interface standard. The International Journal of Supercomputer Applications and High Performance Computing，1994，8(3/4)
5 Faag G，Dongarra J，Geist A. PVMPI provides interoperability between MPI implementations. In：Proc. 8th SIAM Conf. on Parallel Processing，SIAM 1997
6 Graham E F，Dongarra J. PVMPI：An Integration of the PVM and MPI Systems. Calculactures Paralleles，1996，8(2)：151～166
7 William L，George J，et al. IMPI：Making MPI Interoperable. Journal of Research of the National Institute of Standards and Technology，2000，105(3)：343～348
8 IMPI Steering Committees. IMPI：Interoperable Message -Passing Interface. Journal of Research of the National Institute of Standards and Technology，2000，105(3)：349～428
9 Graham E F，Antonin B，et al. HARNESS and fault tolerant MPI. Parallel Computing，2001，6：1～17
10 Beck D，Fagg G，et al. HARNESS：a next generation distributed virtual machine. Journal of Future Generation Computer Systems，Elsevier Science B. V.，1999，15
11 Stellner G. CoCheck：Checkpointing and Process Migration for MPI. In：Proceedings of the International Parallel Processing Symposium，Honolulu，April 1996. 526～531
12 Agbaria A，Roy F. Starfish：Fault-Tolerant Dynamic MPI Programs on Clusters of Work -stations. In：the 8th IEEE Intl. Symposium on High Performance Distributed Computing，1999