LINUX 进程间通信的模型检测*)

姜玉蓉[1] 缪 力[1] 张大方[1,2] 刘潇潇[1]
(湖南大学软件学院 长沙 410082)[1] (湖南大学计算机与通信学院 长沙 410082)[2]

摘 要 模型检测是一种强大的自动分析验证技术。分析了 LINUX 进程间通信的部分源代码并进行手工形式化建模,使用有限状态自动机描述模型,继而转换成 SPIN 的输入语言 PROMELA,对其进行模型检测,验证了系统的有界性和可终止性,并就进程间通信中容易发生的问题提出了改进方案。

关键词 模型检测,LINUX,进程间通信,SPIN,系统验证

Model Checking for LINUX Interprocess Communication

JIANG Yu-rong[1] MIAO Li[1] ZHANG Da-fang[1,2] LIU Xiao-xiao[1]
(Department of Software Engineering, Hunan University, Changsha 410082, China)[1]
(Department of Computer and Communication, Hunan University, Changsha 410082, China)[2]

Abstract *Model checking is a powerful technique to analyse and verify system automatically. Part of the source code of Linux interprocess communication was chosen to be analysed and formalized modeled, then these models were translated into PROMELA, the input language to SPIN, to be model checked. Boundedness and terminability were verified and an improved method to the problem in interprocess communication was worked out.*

Keywords Model checking, Linux, Interprocess communication, Spin, System verification

1 引言

UNIX 操作系统的出现是计算机技术发展史上的一个重要里程碑,它诞生于 1969 年的贝尔实验室[1]。1991 年,Linus Torvalds 等人基本上按照 UNIX 的设计,开发出一个真正可以使用的 UNIX 内核,并称之为 LINUX[2]。可以说 LINUX 内核是目前覆盖面最广的一体化内核,它的安全性和稳定性也众所周知。作为多用户、多任务的操作系统,进程间通信的重要性不可小觑,因此对其进行模型检测成为一个意义重大的研究课题。

模型检测是一种强大的自动分析验证有限状态系统的技术,它将所需验证的系统及属性作为模型检测工具的输入,以判断属性的正确性,是有限状态验证(Finite State Verification)的方法之一,另外还有两种分别是流方程(Flow Equations)和数据流分析(Data Flow Analysis)[3]。迄今为止已开发出多种分析验证工具,如模型检测的 SMV(Symbolic Model Checking)[4],SPIN(Simple Promela INterpreter)[4,5],流方程的 INCA(Integer Necessary Conditions)以及数据流分析的 FLAVERS(FLow Analysis for VERification of Systems)[6,7]。美国的 NASA 实验室还专门开发了一种针对 JAVA 的工具——JPF(Java Path Finder)[8,9],对 JAVA 程序进行模型检测。

本文采用的是 SPIN 工具。它是一种由美国贝尔实验室开发的模型检测工具,应用了偏序规约、on the fly 等技术,大大提高了模型检测的效率,减小了状态空间,可以用断言(assertion)或线性时态逻辑 (LTL)来验证系统属性,已经成功地应用于协议验证、分布式系统验证和软件验证中。本文分析了 LINUX2.4.0 版本的基于管道的进程间通信的内核源码,从中提取形式化模型,再用 SPIN 进行模型检测。

文章结构如下:第 2 节简单介绍模型检测分析工具 SPIN;第 3 节是 LINUX 进程间通信的代码分析和形式化建模;第 4 节对它进行模型检测,并得出实验结果;最后总结全文。

2 模型检测器 SPIN

2.1 SPIN/PROMELA 概述

SPIN(Simple Promela INterpreter)是适合于并发系统的一种模型检测分析验证工具。它以 PROMELA 作为输入语言,对于给定的一个使用 PROMELA 描述的系统模型,SPIN 可以对其执行进行随意的模拟,也可以生成一个 C 代码程序,然后对系统的正确性进行有效的检验。

PROMELA(PROcess MEta LAnguage)是用来对有限状态系统进行建模的形式描述语言。它类似于 C 程序语言,允许动态创建并行的进程,并可以在进程之间通过消息通道进行同步或异步通信。一个 PROMELA 模型由进程、消息通道、变量和全局对象组成,相当于一个有限转换系统。其大体结构为:类型说明、通道说明、变量说明、进程说明、初始进程。

如图 1 所示,SPIN 首先从一个 PROMELA 描述的抽象模型开始,经过分析,没有语法错误后,对系统的交互进行模拟,直到确认系统设计拥有预期的行为;然后,SPIN 将产生一个用 C 语言描述的验证程序,经验证机编译后被执行。执行中如果发现了违背正确性说明的任何反例,则可反馈给交互模拟机,通过重现细节找出引发错误的原因。

*)国家自然科学基金项目(60673155,60703097),国防基础科研"十一五"项目资助(A1420060162)。**姜玉蓉** 硕士研究生,主要研究领域为软件测试、模型检测、网络测试;**缪 力** 副教授,主要研究领域为软件测试、软件工程;**张大方** 教授,博士生导师,主要研究领域为可信系统与网络、容错计算、网络测试、软件测试、网络安全;**刘潇潇** 硕士研究生,主要研究领域为网络测试、模型检测。

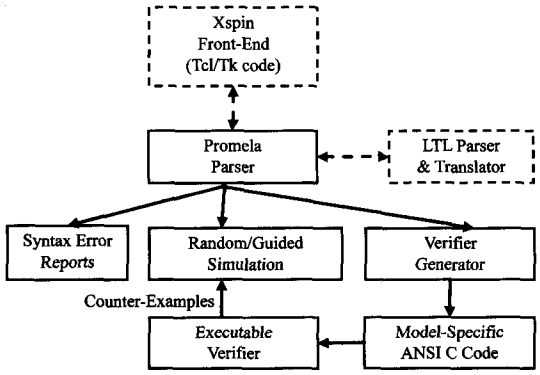


图 1　SPIN 模拟和检验的基本过程

2.2　SPIN 工作原理及相关定义

SPIN 的工作原理基于这样一个事实[5]，即分布式系统中异步进程的行为能够用有限状态自动机(FSA)来模拟。

定义 1(有限状态自动机，FSA)　有限状态自动机 A 是一个五元组(S,s_0,L,T,F)，其中：

$S=\{s_0,s_1,s_2,\cdots,s_n\}$是有限状态集合；

s_0 是 FSA 的初始状态，$s_0\in S$；

L 是触发状态迁移的事件的有限集合；

T 是状态之间的迁移关系，$T\subseteq(S\times L\times S)$；

F 是终止状态，$F\subseteq S$。

定义 2(同步积)　FSA P 和 B 的同步积是一个 FSA $A=(S,s_0,L,T,F)$，其中：

$A.S$ 是 $P'.S\times B.S$ 的笛卡尔积，P' 是 P 的 stutter 闭包；

$A.S_0$ 是二元组$(P.s_0,B.s_0)$；

$A.L$ 是 $P'.L\times B.L$ 的笛卡尔积；

$A.T$ 是(t_1,t_2)对，$t_1\in P'.T$ 且 $t_2\in B.T$；

$A.F$ 是$(s_1,s_2)\in A.S$ 对的集合，$s_1\in P.F\vee s_2\in B.F$。

定义 3(异步积)　一个 FSA 有限集合 $A_1,\cdots,A_n$ 的异步积是一个 FSA $A=(S,s_0,L,T,F)$，其中：

$A.S$ 是 $A_1.S\times\cdots\times A_n.S$ 笛卡尔积；

$A.s_0$ 是 n 元组$(A_1.s_0,\cdots,A_n.s_0)$；

$A.L$ 是 $A_1.L\cup\cdots\cup A_n.L$；

$A.T$ 是元组集合$((x_1,\cdots,x_n),I,(y_1,\cdots,y_n))$，其中 $\exists i,1\leqslant i\leqslant n,(x_i,I,y_i)\in A_i.T$，且 $\forall j,1\leqslant j\leqslant n,j\neq i\rightarrow(x_j\equiv y_j)$；

$A.F$ 是 $A.S$ 的子集，满足 $\forall(A_1.s,\cdots,A_n.s)\in A.F$，其中 $\exists i,A_i.s\in A_i.F$。

定义 4(线性时态逻辑[11,12]，LTL)　线性时态逻辑常用于描述需验证的属性，它由原子命题、逻辑连接符和模态算子组成。逻辑连接符包括$\neg$(非)、$\vee$(或)、$\wedge$(与)。模态算子包括$\langle\rangle$(Eventually)、$[]$(Always)。设 p 为原子命题，LTL 公式的产生规则如下：$\varphi::=p\mid(\neg\varphi)\mid(\varphi\wedge\varphi)\mid(\varphi\wedge\varphi)\mid([]\varphi)\mid(\langle\rangle\varphi)$

其中$\langle\rangle$表示现在或以后某一状态，$[]$表示现在和以后所有状态。例如，$[]\langle\rangle p$ 表示 p 无限多次为真，$\langle\rangle[]p$ 表示从某个时刻起，p 一直为真。

定义 5(全局系统自动机 S)

$$S=B\otimes\prod_{i=1}^{n}Ai$$

其中，B 表示 Büchi 自动机，由给定的 LTL 表达式转换而成；A_i 表示由进程形式化得到的自动机；$\otimes$表示两个自动机的同步积；Π表示若干自动机的异步积。

假设系统 A 有 n 个进程，形式化成 n 个自动机：$A_1,A_2,\cdots,A_n$，考虑一个 LTL 表达式 f 以及相应的$\neg f$形成的 Büchi 自动机 B，应用自动机验证理论，便能通过计算全局系统自动机 S 来检测系统 A 是否满足属性 f。

3　LINUX 进程间通信的形式化建模

3.1　基于管道的进程间通信

对于 LINUX 这种多任务、多用户的操作系统来说，进程间通信是一种非常重要甚至必不可少的基本手段和设施。本文针对的是通过管道(Pipe)进行的进程间通信。父进程与子进程之间，或者两个兄弟进程之间，可以通过系统调用建立起一个单向的通信管道，这种管道只能由父进程建立。管道两端的进程将该管道看成文件。一个进程往管道中写入数据，另一个进程从管道中读取，即管道的写端和读端，它们遵循先入先出的原则，且通信是单向的。图 2 展示了父子进程通过管道进行通信。

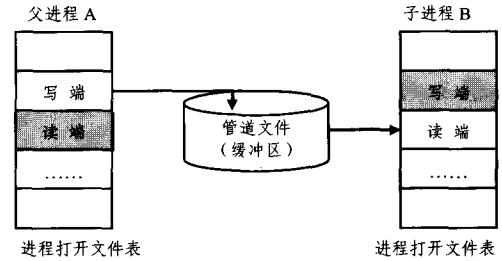


图 2　父子进程通过管道单向通信

如图 2 所示，进程 A 创建一个管道，创建完成时管道的读端和写端都在进程 A 中。进程 A 通过 fork()创建出进程 B，在 fork()的过程中进程 A 的打开文件表原封不动复制到进程 B 中。进程 A 关闭管道读端，而进程 B 关闭管道写端。于是，进程 B 就能够从管道中读取进程 A 写入的数据。

对于这样的进程间管道通信，有可能出现以下几个问题：第一，进程 A 有大量的数据需要写入管道，导致缓冲区溢出，这是非常危险的。据统计，通过缓冲区溢出进行的攻击占所有系统攻击总数的 80%以上。第二，进程 A、进程 B 以及管道，三者最终是否能够到达一个稳定的正确状态，这是管道通信正常工作的基本保证。第三，这种情况相对复杂，当进程 A 只写入了部分数据而管道已满时，要待 B 读走一些数据后才能继续写入，而 B 读完之后退出，A 还有未写完的数据，这时会发生什么状况呢？在下面的章节中，将一一详细解答。

3.2　形式化建模

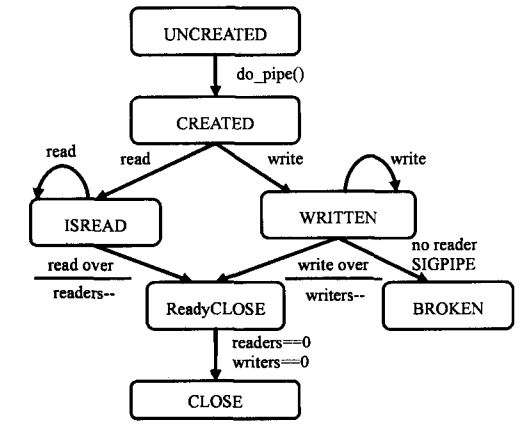
基于 LINUX 管道通信的基本思想，本文针对相关源代码(主体代码在 linux/fs/pipe.c 中)进行形式化建模。使用有限状态自动机(FSA)描述模型。将管道、管道读端以及写端看作三个实体，建立三个相应的 FSA。

三个有限状态自动机显然会涉及三个实体的互相交互，加上并发机制本身的不确定性因素，无疑给建模工作带来了困难。由于篇幅的限制，本文不对 LINUX 源码做具体讲解。

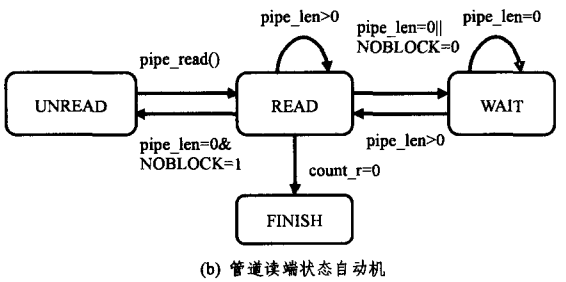
3.2.1　管道模型

通过分析 linux/fs/pipe.c 源代码，了解到管道机制的主体是系统调用 pipe()，系统调用的主体是 do_pipe()，以此建立管道。管道创建后，可通过它进行读写操作，实现进程间通信。当读者或写者结束时，管道进入预关闭状态，直到没有读者和写者，管道才正式关闭。若准备写入数据时检测到没有

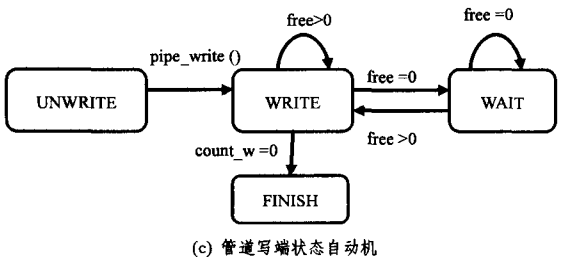
读者，则宣布管道断裂。根据管道的特性对其形式化建模，见图 3(a)。圆角矩形表示管道所处的状态，箭头表示状态的迁移，箭头上的标识说明迁移的触发条件或动作。



(a) 管道状态自动机



(b) 管道读端状态自动机



(c) 管道写端状态自动机

图 3 状态自动机

3.2.2 管道读端

通过对源码的分析了解到，管道的读端总是处于一个 for 循环中，通过系统调用 read()从管道中读取数据进行处理，处理完后接着读取。对管道的读操作，在内核中经过 sys_read()和数据结构 read_pipe_fops 中的函数指针而到达主体函数 pipe_read()。调用该函数，进行管道的读取操作。如果此时管道缓冲区中已有数据，则开始读取；然而，如果管道中没有数据，一般就需要进入等待状态。其中，又有两种情况例外：

- 此时写者计数器已经为 0，即没有写者，说明管道中不会再被写入数据。在这种情况下等待没有意义。
- 管道创建时，设置了标志位 NOBLOCK，表示当读者读取不到管道中的数据时，不被阻塞。此时也无需等待。

读者在等待的过程中，如果写端向管道中写入了数据，则它重新进入读取状态。当读者完成了读取操作，就自然进入到最终的完成状态。同样地，对于管道读端建立的模型如图 3(b)。

3.2.3 管道写端

管道写端的操作与读端类似，只是当写端请求写入的长度小于管道缓冲区容量时，内核需要保证写入操作的原子性。也就是说，在管道空闲区域足够大，能容纳写入的全部数据时才开始进行一次性写入操作。然而，当请求写入的长度大于管道缓冲区时，就不保证写入的原子性，一旦有空闲区域就写入，未写完的数据待读端读取部分数据后继续写入。最终同样进入完成状态。建立的模型如图 3(c)所示。

3.3 形式化模型到 PROMELA 的转换

形式化建模后，将它转化成 PROMELA。以管道模型部分代码为例：

```
proctype Pipe()
{ do
  ::Pipe_State == WRITTEN → atomic
    {if
    ::write == 1 → skip;
      if
      ::(writers > 0 && readers == 0
        → Pipe_State = BROKEN;
            read = 0;
            write = 0
      ::else → skip
      if
    ::(write == 0 && Writer_State ! = WAIT)
          → Pipe_State = ReadyCLOSE
    if
    }
  ::.....
  od
}
```

进程通过 do 和 od 循环判断管道所处的状态，atomic{}中的内容表示不被打断的原子操作。如果管道当前的状态为 WRITTEN，表示管道的写端正在向管道中写入数据。此时，如果检测到管道另一端没有读者，则写操作无意义。于是，内核发出 SIGPIPE 信号，随即管道断裂。管道写端完成写操作后，管道状态则迁移到 ReadyCLOSE（预关闭）。

4 模型检测实验

实验环境：CPU3.06GHz，内存 512M，硬盘 80G，操作系统 WinXP，模型检测器 SPIN 4.3.0。通过 SPIN 对系统进行模拟，结果显示，在设置了不同参数的条件下，它能很好地模拟出系统各条不同的执行路径，并且最终到达正确的结束状态。

4.1 属性表达

正如 3.1 节中所述，进程间管道通信存在几个重要的问题，换言之，它具有几个关键属性。因此，本文通过模型检测的方法对其属性正确性进行验证是非常有意义的。

属性常用线性时态逻辑(LTL)表达式描述，并作为 SPIN 的输入。

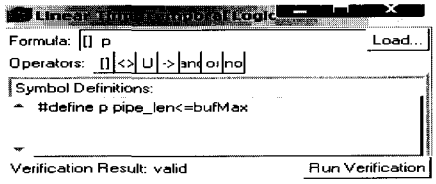
首先，验证有界性，即变量不能超过它的边界。例如，写端向管道中写入数据后，管道长度为 pipe_len，该长度不允许超过管道缓冲区的总长 bufMax，否则会导致缓冲区溢出，从而发生不可预测的危险。用 LTL 公式来表示：[](pipe_len <= bufMax)。如果出现越界的情况，则不满足该属性。作为对比，这里将检验的性质改为[](pipe_len < bufMax)，以断言的形式插入程序中，重新检验一次。

其次，验证可终止性，也就是检验系统最终是否都达到有效的结束状态，以保证进程间通信的正常工作。具体的终止

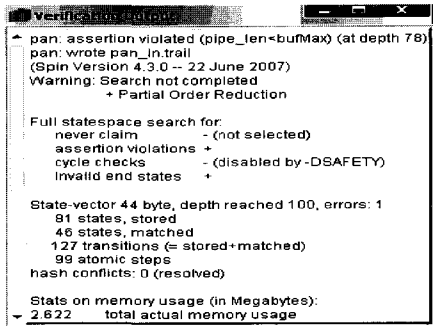
状态在系统验证的结果中列出(见表 1),此处不一一赘述。

4.2 系统验证

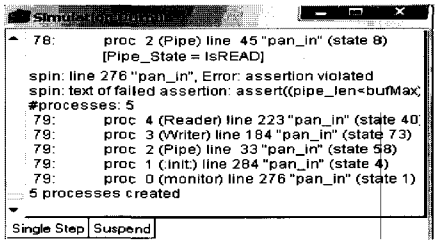
本文使用 Xspin 对系统进行模型检测,这是一个独立运行于 SPIN 的图形界面软件,图 4 显示其工作界面。对于 LTL 的第一个公式,在 Xspin 中表示如图 4(a),验证结果显示该性质有效。



(a) LTL 公式的表达及验证结果



(b) 检测输出结果



(c) 模拟输出结果

图 4 Xspin 工作界面

插入断言后,再次检验的结果说明系统不满足性质,详细输出见图 4(b)。

可以看到,详细输出结果说明系统在第 78 层违反了属性,并且生成了 trail 文件(即 SPIN 生成的反例),以便查找错误原因。随后列出 SPIN 的版本信息、对整个状态空间进行搜索之后获得的信息(包括声称、断言、循环检查和无效最终状态)、状态矢量、可达深度和错误数的信息(包括存储状态、匹配状态、迁移和原子操作数)、哈希冲突以及实验所使用的内存信息。从图 1 中得知,生成的反例可以反馈回模拟器再次进行系统模拟,此时发现 SPIN 结束在断言之后:spin:line 276 "pan_in",Error:assertion violated(见图 4(c)),其中 pan_in 是模型经编译后生成的 C 程序。以此模拟输出为基础,能够迅速找到错误并加以解决。

通过设置不同的参数,Xspin 仿真输出的结果显示管道、管道写端以及管道读端的最终停止状态(见表 1)。

表 1 各实体终止状态

ENTITY	FINAL STATE		
Pipe	CLOSE	BROKEN	BROKEN
Writer	FINISH	WRITE	WAIT
Reader	FINISH	FINISH	FINISH

第一列是各实体名称。

第二列说明管道到达关闭状态,写端和读端都到达完成状态。这是管道完成通信的最佳状态。

第三列显示此时写端还能够进行写操作,但是读端已经读取完毕,并且没有读者再读管道,因此管道断裂。这也是正确结束的状态。

第四列说明此时写端未写完数据,由于管道已满而进入等待,读端读取完后结束,于是管道断裂。

从实验结果可以看出,系统满足可终止性。然而,第三种情况(第四列)常常导致不完整的通信,即 3.1 节中提到的第三个问题。在实际应用中,两个进程通过管道进行通信,常常会出现 BROKEN PIPE,即"管道断裂"的提示,导致信息往往无法完整地传递给另一进程。为什么会出现这样的提示信息呢?原因之一是写端在写入了部分数据后,管道缓冲区满,则进入等待,读端开始读取,完成操作后,读者计数器减到 0(假设只有这一个读者)。此时,写端希望继续向管道中写入剩下的数据以供读取,然而此时已经没有读者,因此会收到 SIGPIPE 信号,随之管道断裂。这样的问题层出不穷,严重影响了进程间的正常通信。本文提出的改进措施是,在读端读取完之后,应该及时检验是否写端还有未写完的数据。如果有,则等待,待写端完成操作,读端再次读取管道中的数据。图 5 给出改进后的管道读端模型,虚线部分为新增的状态和迁移条件。

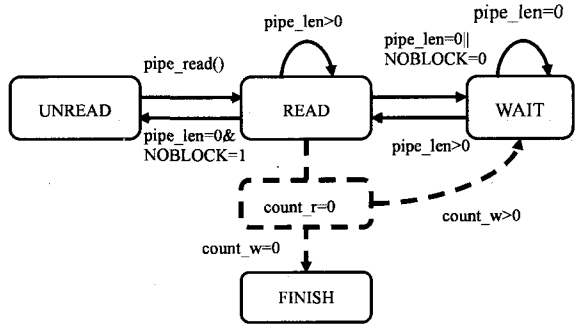


图 5 改进后的管道读端自动状态机

经过改进,方能避免不必要的管道断裂信息,大大减少了由于管道断裂而导致的进程间不完整通信。

结束语 本文从 LINUX 2.4.0 版本开始着手分析内核源码,研究了与进程间管道通信相关的部分代码,并从中提取出管道模型、管道读端模型以及管道写端模型。三个实体之间的交互,以及并发行为的不确定性都大大增加了建模工作的难度,经过努力得以解决。随后将模型转换成 SPIN 的输入语言 PROMELA,对系统进行模拟和检测工作。实验结果表明,系统满足有界性和可终止性。最后针对实际进程间通信工作中容易出现的一个问题提出了改进方案,以避免过多地提示管道断裂,导致不完整的通信。下一步工作是细化和改进现有模型,并对性质进行更加深入的分析与验证。

参 考 文 献

[1] AL-Saqabi S. LINUX&UNIX 程序开发基础教程[M]. 清华大学出版社,2004

[2] 毛德操,胡希明. Linux 内核源代码情景分析[M]. 浙江大学出版社,2001

[3] Dwyer M B,Clarke L A. Data flow analysis for verifying properties of concurrent programs[J]. Software Engineering Notes,

1994,19(5):62-75
[4] 古天龙,蔡国勇. 网络协议的形式化分析与设计. 电子工业出版社[M],2003
[5] Holzmann G. The SPIN model checker. Pearson Publishing House[M/CD],2003
[6] Cobleigh J M, Clarke L A, Osterweil L J. FLAVERS: A finite state verification technique for software systems[J]. IBM Systems Journal,2002,41(1):140-165
[7] Dwyer M B, Clarke L A, Naumovich G. Flow Analysis for Verifying Properties of Concurrent Software Systems [J]. ACM TOSEM,2004,13(4):359-430
[8] Havelund K, Pressburger T. Model Checking Java Programs Using Java PathFinder[J/OL]. Journal on STTT, December 1998[2007-8-3]. http://citeseer.ist.psu.edu/havelund98model.html
[9] Visser W, Havelund K, Brat G, et al. Java PathFinder - Second Generation of a Java Model Checker[C/OL]// Proc. of Post-CAV Workshop on Advances in Verification. Chicago, July 2000 [2007-8-3]. http://citeseer.ist.psu.edu/visser00java.html
[10] Esparza J, Heljanko K. Implementing LTL model checking with net unfoldings[C]. New York: Springer-Verlag, Inc, 2001:37-56
[11] Bodden E. A lightweight LTL runtime verification tool for java [C]. New York: ACM Press, 2004:306-307
[12] 林惠民,张文辉. 模型检测:理论、方法与应用[J]. 电子学报,2002,30(12A):1907-1912

(上接第 151 页)

则

$$F(\lambda x+(1-\lambda)y)\leqslant F(\lambda x)+F((1-\lambda)y)=\lambda F(x)+(1-\lambda)F(y)$$

从而 F 是凸的。

定理 11 设 $F: H\to\Omega$ 是一个直觉模糊映射,则 F 是凸的当且仅当对于任意的 $x,y\in H$

$$\varphi(\lambda)=F(\lambda x+(1-\lambda)y)$$

是[0,1]上的凸直觉模糊映射。

证明:若 $\varphi(\lambda)$ 是[0,1]上的凸直觉模糊映射,则对于任意的 $x,y\in H,\lambda\in[0,1]$

$$\begin{aligned}F(\lambda x+(1-\lambda)y)&=\varphi(\lambda)\\&=\varphi(\lambda 1+(1-\lambda)0)\\&\leqslant\lambda\varphi(1)+(1-\lambda)\varphi(0)\\&=\lambda F(x)+(1-\lambda)F(y)\end{aligned}$$

因此 F 是凸的。

反之,若 F 是凸的,则对于任意的 $x,y\in H,\lambda_1,\lambda_2\in[0,1],k\in[0,1]$

$$\begin{aligned}&\varphi(k\lambda_1+(1-k)\lambda_2)=F((k\lambda_1+(1-k)\lambda_2)x+(1-k\lambda_1-(1-k)\lambda_2)y)=F(k(\lambda_1x+(1-\lambda_1)y_2)+(1-k)(\lambda_2x+(1-\lambda_2)y))\\&\leqslant kF(\lambda_1x+(1-\lambda_1)y_2)+(1-k)F(\lambda_2x+(1-\lambda_2)y)=k\varphi(\lambda_1)+(1-k)\varphi(\lambda_2)\end{aligned}$$

因此 $\varphi(\lambda)$ 是[0,1]上的直觉模糊凸映射。

结束语 在本文中,我们得到了刻画凸直觉模糊映射的充要条件,给出了具有上半连续性(下半连续性)直觉模糊映射是凸直觉模糊映射的充要条件,获得了直觉模糊算子与凸性之间的关系相关性质。这些结论很容易推广到直觉模糊映射的广义凸性,如直觉模糊映的 b-vexity, Preinvexity, φ_1-convexity 等。由于直觉模糊数空间不构成任何线性空间,因此我们不能用通常的方法给出直觉模糊映射的可微性的相关理论,进而把本文的相关结果应用到直觉模糊非线性规划的研究中,这也是我们下一步所要研究的课题。

参 考 文 献

[1] Zaden L A. Fuzzy sets. Infromation and Control, 1965, 8: 338-353
[2] Atanassov K. Intuitionistic fuzzy sets. Fuzzy sets and systems, 1986,20:87-96
[3] Gau W L, Buehrer D J. Vague sets. IEEE Trans. On Systems Man and Cybernet,1993,23(2):610-614
[4] Bustince H, Burillo P. Vague sets are intuitionistic fuzzy sets. Fuzzy sets and system,1966,79:403-405
[5] Park J H. Intuitionistic fuzzy metric spaces. Chaos, Solitions and Fractals,2004,22:1039-1048
[6] Gregori V, Romiguera S, veeramari P. A note on intuiionistic fuzzy metric spaces. Chaos, Solitions and Fractals,2006,28:902-905
[7] Reza, Saadati. Notes to the paper "Fized points in intuitionistic fuzzy metric spaces" and its generalization to-fuzzy metric space. Chaos, Solitions and Fractals,2006,28
[8] Coker D. An introduction to intistionistic fuzzy topological space. Fuzzy sets and systems,1997,88:81-89
[9] Saadati R, Park J H. On the intuitionistic fuzzy topological spaces. Chaos, Solitions and Fractals,2006,27:331-334
[10] Lupianez F G. Nets and filters in intuitionistic fuzzy topological spaces. Information Science,2006,176:2396-2404
[11] Razani A. Existence of fixed point for the nonexpansive mapping of intuitionistic fuzzy metric spaces. Chaos, Solitions and Fractals,2006,30:367-373
[12] Mohamed A. Fixed point theorems in intuitionistic fuzzy metric spaces. Chaos, Solitions and Fractals,2006,30
[13] Hong D H, Choi C H. Multicriteria fuzzy decision making problems based on vague set theory. Fuzzy sets and systems, 2000, 114:103-113
[14] Chen S M, Tan J M. Handing multicriteria fuzzy decision-making based on vague set theory. Fuzzy sets and systems,1994,67:163-172
[15] Lin L, Yuan Y H, Xia Z Q. Multicriteria fuzzy decision-making based on intuitionistic fuzzy sets[J]. Computer and system Science,2007,73:84-88
[16] Chen S M. Similarity measures between vague sets and between elements. IEEE Trans. System Man and Cybernet,1997,27(1):153-158
[17] Dengfeng L, Chuntian C. New similarity measure of intuitionistic fuzzy sets nad application to pattern recognitions. Pattern Recognition Letters,2002,23:221-225
[18] Li Y, Olson D L, Qin Z. Similarity measures between intuitionistic fuzzy (vague) sets: A comparative analysis. Pattern Recognition Letters,2006
[19] Ammar E, Metz J. On fuzzy convexity and parametric fuzzy optimization. Fuzzy sets and systems,1992,49:135-141
[20] Furukawa N. Convexity and local Lipschitz continuity of fuzzy-value mappings. Fuzzy sets and systems,1998,93:113-119
[21] Li D. Properties of B-Vex fuzzy mappings and applications to fuzzy optimization. Fuzzy sets and systems,1998,94:253-260
[22] Noor M A. Fuzzy preinvex functions. Fuzzy sets and systems, 1994,79:95-104
[23] Nanda S, Kar K. Convex fuzzy mappings. Fuzzy sets and systems,1992,48:129-132
[24] Yang X M. A characterization of convex function. Applied Mathematics letters,2000,13(1):27-30
[25] Yan H, Xu J. A class of convex fuzzy mappings. Fuzzy sets and systems,2002,129:47-56
[26] Syau Y. Invex and generalized convex fuzzy mappings. Fuzzy sets and systems,2000,115:455-461