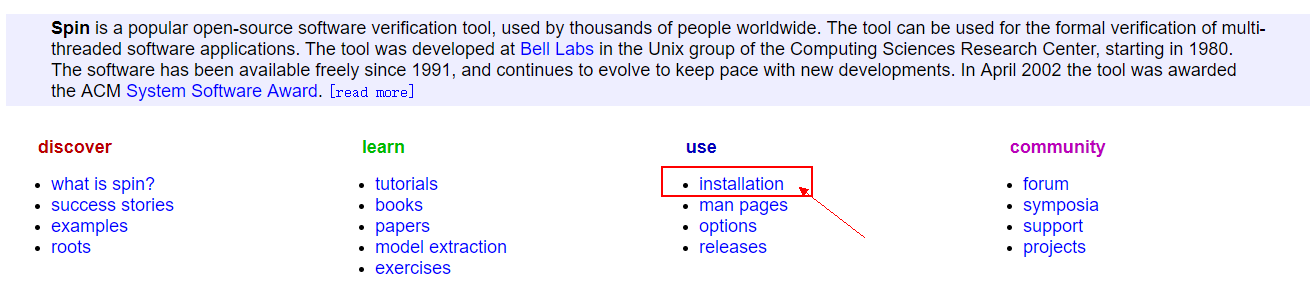
SPIN和SMV的对比学习

# 一.安装SPIN

## 1.1下载SPIN可执行文件

进入spin验证工具的官网：<http://spinroot.com/spin/whatispin.html>



进入安装向导页面，按照指示来进行安装。

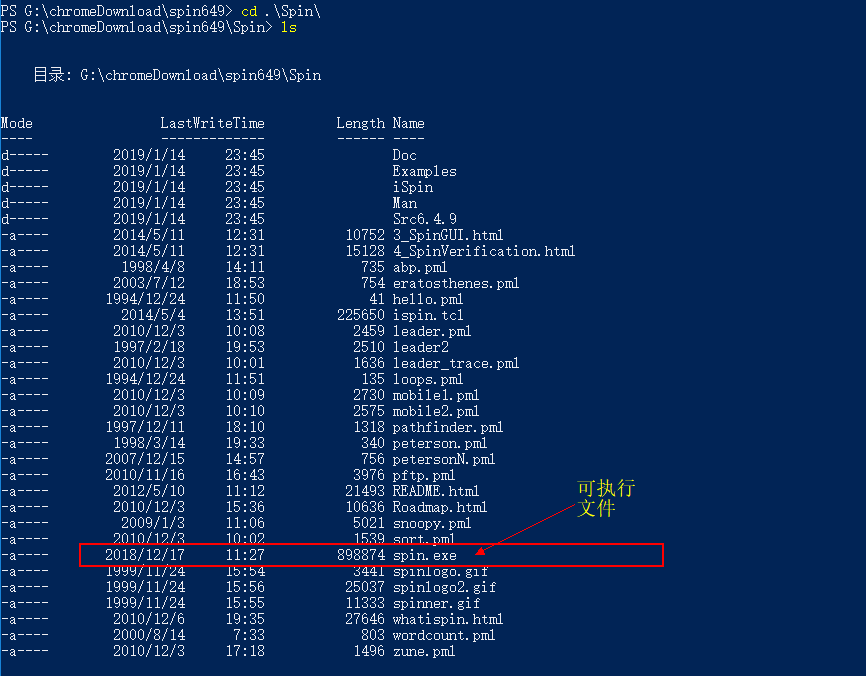
Windows 10操作系统可以直接下载二进制文件。

可执行文件链接：<http://spinroot.com/spin/Src/pc_spin649.zip>

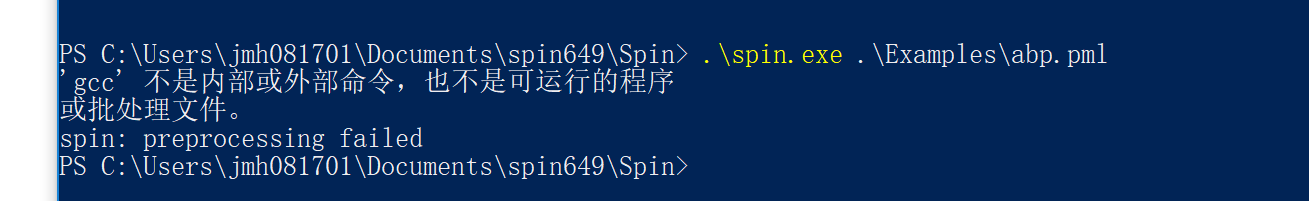
完整文件链接：<http://spinroot.com/spin/Src/spin649.tar.gz>

或者直接下载：<https://pan.baidu.com/s/1mXOpl4kmXTqE46mKqbtz5w>提取码：ji9h

下载完毕后解压,切换到spin.exe所在的目录下。



但是，直接跑是不得行的，它会报如下错误：



这是因为SPIN需要将promela语言先转义为C语言，然后再用C语言的编译器例如GCC生成最终的模型检验程序。

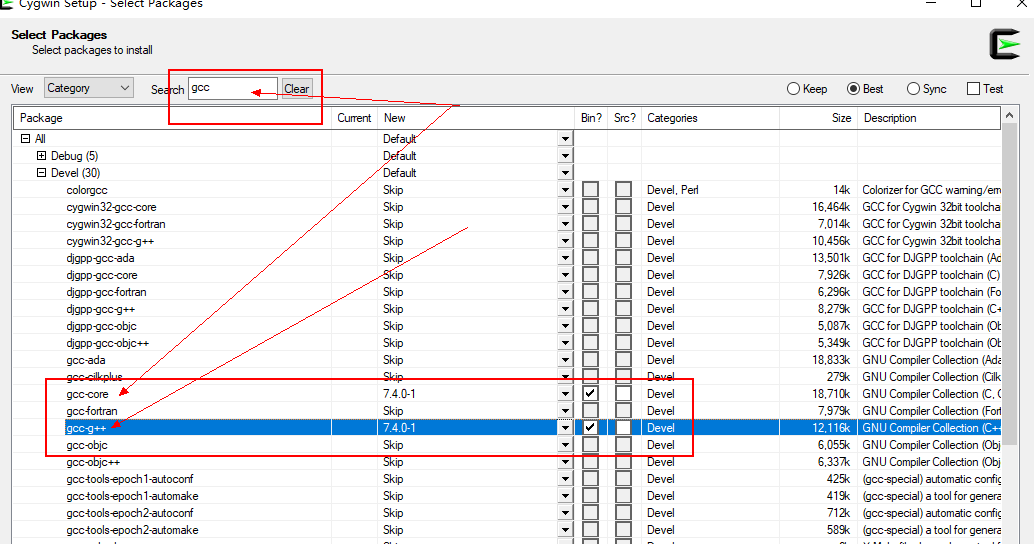
因此，还需给Windows 10安装gcc编译器。

## 1.2安装gcc编译器

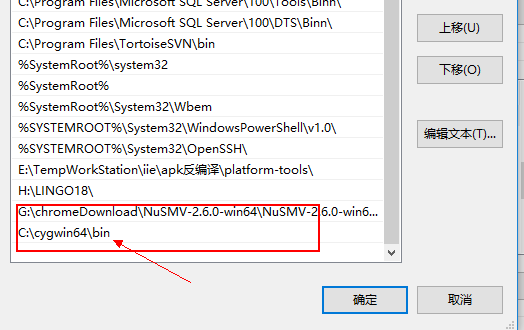
下载cygwin安装文件。

文件链接：<http://www.cygwin.com/setup-x86_64.exe>

安装时注意选择安装gcc这个packages。



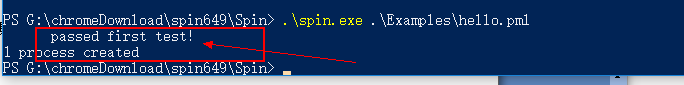
把路径C:\cygwin64\bin添加到系统的环境变量中。



使环境变量生效：



重新打开powershell,测试一下工具是否可以使用，如果出现如下界面就说明安装成功了。



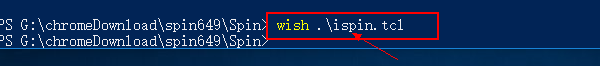
## 1.3安装SPIN的GUI界面iSpin

自从6.4.9开始,spin的gui界面就改由基于tcltk来实现了。这是因为使用tcltk实现的GUI程序可以具有更好的跨平台移植性。

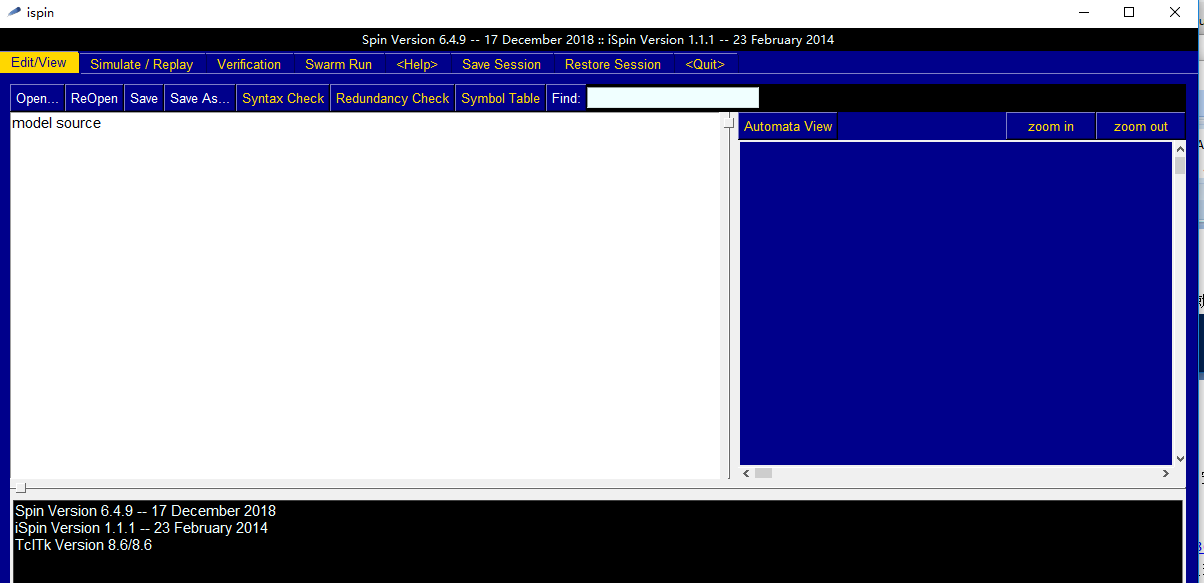
首先需要安装tclck.下载tcltk安装包,下载链接：<https://bitbucket.org/tombert/tcltk/downloads/tcltk86-8.6.8-10.tcl86.Win7.x86_64.tgz>

把文件解压后，将文件的bin路径添加到环境变量，并使环境变量生效。

然后切换到刚刚spin649所在的目标，输入：wish ispin.tcl，打开图形界面。



图形化界面视图：



运行一个Example测试一下。

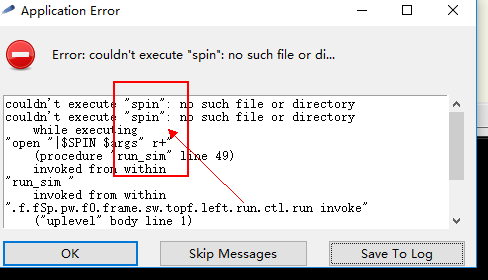
输入：

1 init {

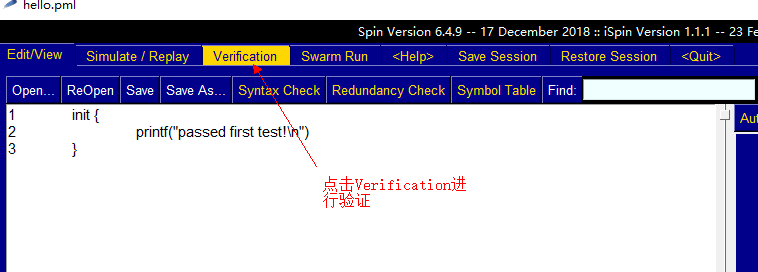
2 printf("passed first test!\n")

3 }

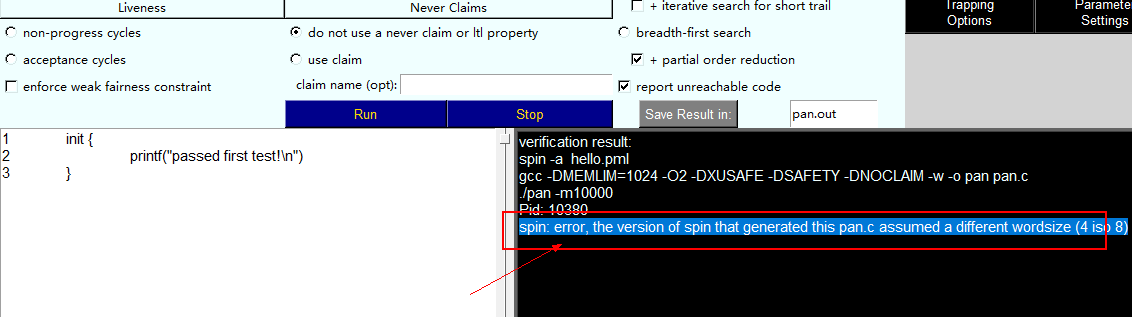
发现会报错：



这是因为刚刚没有把spin.exe所在的目录添加到环境变量中。现在把spin.exe所在目录添加至环境变量，并使环境变量生效，从新打开ispin图形化界面。



运行结果：

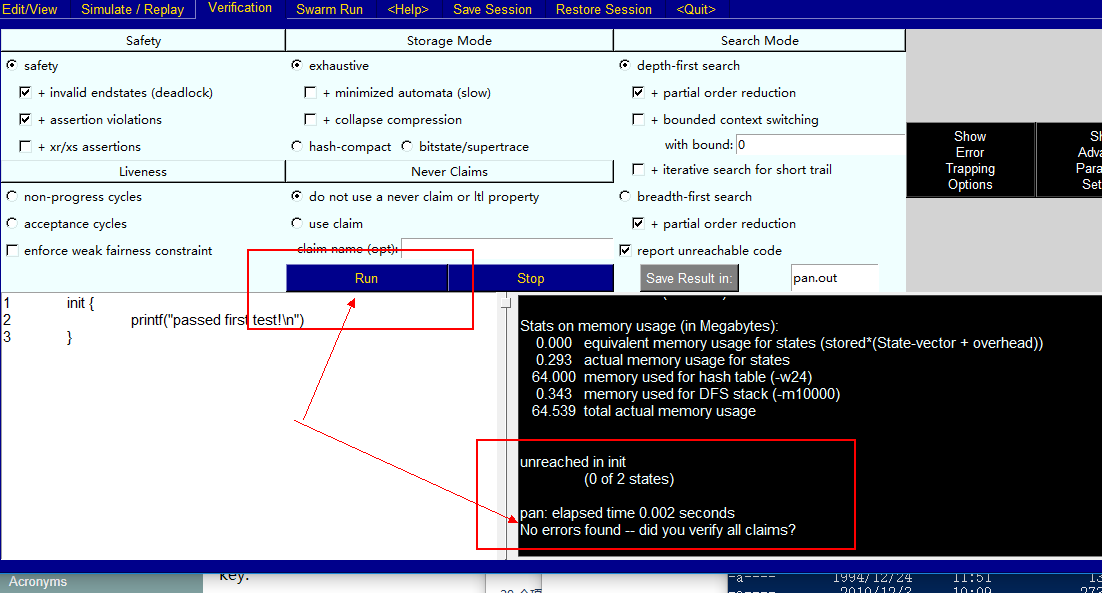


还是要报错，查询资料后发现需要做如下修改。

把刚刚安装的cygwin for 64bit,改成cygwin for 32 bit.这是因为spin只会生成32位的pan.c代码。

Cygwin for 32bit下载地址：https://cygwin.com/setup-x86.exe,安装过程同1.2所示。安装完毕后。

再运行Verification:



发现就是成功的了。

# 二.SPIN的简易教程

spin是数据通信并发系统进行形式化验证的工具。该工具使用Promela语言来描述其中的进程、通信过程。Promela支持动态的创建并发进程。通过消息信道来进行同步或异步通信。

给定一个用Promela描述的通信系统，Spin既可以直接模拟模型的运行，也可以先生成C程序，然后通过执行C程序来完成对系统的正确性验证。

Modeling Language

Promela程序由process,message channel,variable组成。process是全局的对象，而消息信道Message channel和变量既可以是全局的也可以是局部的。Process描述模型的行为，信道和全局变量则描述进程Process运行的环境。

Executability

在Promela里面的每一条语句的执行都是有条件的。任何一条语句，要么被执行，要么被阻塞。如果一个进程的某条语句不能执行了，那么这个进程也会被阻塞，只要出现这条语句可执行的事件发生，这个进程才能继续执行。

例如，我们可能会写如下语句：

while (a != b)

skip /\* wait for a==b \*/

表示当a!=b时就一直死循环。直到a等于b的时候，才会执行while之后的语句。

在promela只需要使用（a==b）就可以达到这个目的了。因为只有这个a==b成立的时候，它才会接着跑下一条语句。

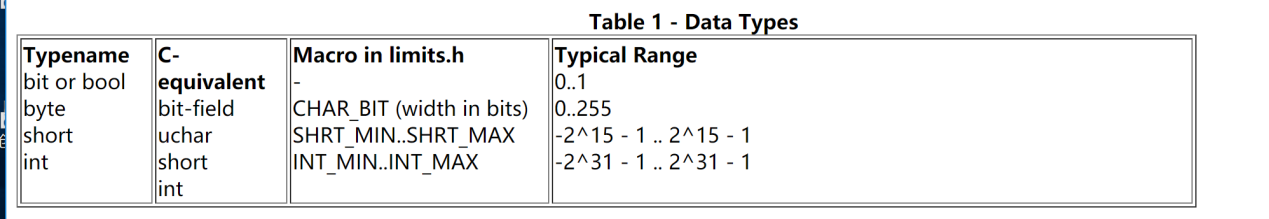
Variables

在Promela中，变量分为全局和局部变量之分，其含义与C语言的含义保持一致。

变量命名语法：变量类型变量名；

例如： bool flag;

Promela里面的基本数据类型如下：



数组类型：

数组的定义同C语言。例如：

byte state[N]

定义一个可以存储N个比特的数组state。数组元素的访问同C语言，下标从0开始。

赋值：同C语言里面的赋值语言用法。

赋值语句和声明语句在Promela都是无条件执行的。

进程类型：process Types

变量的值或信道里面的内容只能在某个进程的作用域内访问，包括读和写。

进程的定义：

proctype 进程名（）

{

进程语句；

}

分号是语句的分割符号，而不是终止符号。因此最后一条语句后面没有分号（与C语言有区别）。

Promela接受两种分割符：“->”和“；”。这两种符号是等价的。

例如：

byte state = 2;

proctype A()

{ (state == 1) -> state = 3

}

proctype B()

{ state = state - 1

}

进程实例:Process Instantiation

protype只是声明的一个进程及其该有的行为。这个进程其实是还没有跑起来的。

Promela语言规定，初始状态下只有一个名叫init的进程能够执行。init进程的作用就相当于C语言里面的main函数。init进程必须被显式的定义而且init形式比较特殊，否则整个模型就跑不起来。

例如：

init

{ run A(); run B()

}

在init里面可以进行全局变量的初始化，已经其它子进程的初始化。在上面的例子中，run是一个一元运算符。当模型的进程数没达到上界前，run运算符就会启动后面跟着的进程名对应的进程。

在run一个进程实例的时候，也可以往进程函数传基本参数。例如：

proctype A(byte state; short foo)

{

(state == 1) -> state = foo

}

init

{

run A(1, 3)

}

注意，只能传基本类型以及通道类型参数，数组和进程类型数据是传不得的。

run 语句可以在任何进程体内写。一个进程只会在由它创建的进程都跑完以后才会结束。

但是一旦有多个进程的时候，就要注意对全局变量的读写中存在的竞争了。

消息类型：

mtype={ack,nak,err,next,accept}这个是自己定义的，后面可以使用。

例如：chan q = [4] of { mtype, mtype, bit, short };

原子序列

atomic{

语句;

}

atomic花括号后面的语句都是原子执行的，要么全部执行，要么一条不执行，不可以中断。

消息传递：Message passing

消息通道用于进程间的信息交互。他们可以定义成全局的或者局部的。例如：

chan qname = [16] of { short }

定义一个可以存储16个short的信道qname.信道可以作为参数再进程间传递。

如果信道有多个参数的话，可以这么定义：

chan qname = [16] of { byte, int, chan, byte }

表示信道里面的每个元素都是{byte,int,chan,byte}的结构体。

信道发送语句：qname! 表达式expr。表示把表达式expr的值发送到qname里面去。默认信道都是队列。先进先出。

信道接收语句：qname?msg。表示从信道的头部接收一个消息，注意是FIFO的方式。然后把内容保存到msg里面去。

如果要一次性发多个值：qname!expr1,expr2,expr3.表示一次发三个值。

qname?var1,var2,var3表示一次性收三个值。注意一次性发几个值就要一次性收几个值。

信道发送的执行条件是信道未满，当信道慢了以后就阻塞。类似的，当信道非空时信道才接收。

qname?cons1,var2,cons2表示当接收到的信息的一头一尾都是对应的常量时才接收。

例子：

proctype A(chan q1)

{ chan q2;

q1?q2;

q2!123

}

proctype B(chan qforb)

{ int x;

qforb?x;

printf("x = %d\n", x)

}

init {

chan qname = [1] of { chan };

chan qforb = [1] of { int };

run A(qname);

run B(qforb);

qname!qforb

}

输出的值应该是123.

len(qname)得到信道qname包含的消息个数。

通信：

有缓存的：chan qname=[N] of {byte}

无缓存：chan port=[0] of {byte}🡪表示同步信道。表示有发送必须立马有人接收。

#define msgtype 33

chan name = [0] of { byte, byte };

proctype A()

{ name!msgtype(124);

name!msgtype(121)

}

proctype B()

{ byte state;

name?msgtype(state)

}

init

{ atomic { run A(); run B() }

}

第一步握手是可以的。第二次会gg.

控制语句

if语句：

if

:: (a != b) -> option1

:: (a == b) -> option2

fi

proctype C()

{ if

:: ch?a

:: ch?b

fi

}

if

:: a > b -> ...

:: else -> ...

fi

如果没有写条件，那么满足条件的随机选一条执行。

proctype counter()

{

if

:: count = count + 1

:: count = count - 1

fi

}

随机的加或减1.

循环

byte count;

proctype counter()

{

do

:: count = count + 1

:: count = count - 1

:: (count == 0) -> break

od

}

先随机的增或减，当等于0的时候就break.

改进：

proctype counter()

{

do

:: (count != 0) ->

if

:: count = count + 1

:: count = count - 1

fi

:: (count == 0) -> break

od

}

跳转：

proctype Euclid(int x, y)

{

do

:: (x > y) -> x = x - y

:: (x < y) -> y = y - x

:: (x == y) -> goto done

od;

done:

skip

}

timeout超时：其他系统不跑的时候

当运行结束后，可以查看\*.trial文件得到错误的路径。

错误路径的查看：

Spin –t –p –g –c 模型文件

# 三.使用SPIN验证电子支付协议

课堂介绍了电子支付协议的基本状态和基本过程。但是课堂上介绍的状态较为粗略，而且只展示了其中客户进程SPIN。

我们来使用SPIN做完整的电子支付协议的分析。

## 3.1 主体的状态及其行为描述

电子支付协议中共涉及了3类主体：客户（C），商家（M），以及银行（B）.

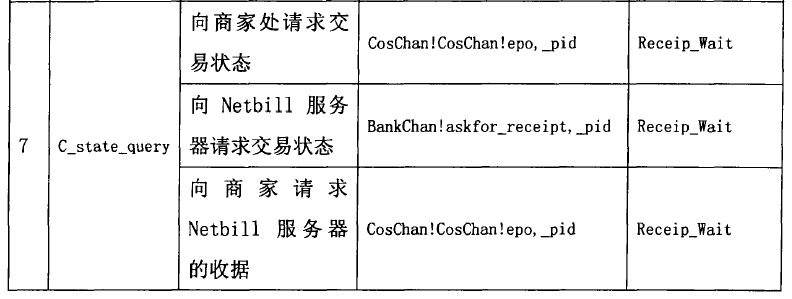
### 3.1.1 客户状态及其行为

客户的基本状态有7个，分别是：

|  |  |
| --- | --- |
| 状态名 | 描述 |
| C\_Idle | 客户闲置状态 |
| Resp\_Wait | 等待商家回应 |
| Goods\_Wait | 等待货物 |
| Receip\_Wait | 等待收据和商品密钥 |
| C\_Suc | 交易成功 |
| C\_Abort | 客户终止交易 |
| C\_state\_query | 向商家或银行查询状态 |

客户的状态转移关系表：





其状态机如图：

CosChan?abort,I 店家放弃

用户主动放弃

收到加密商品

向服务器请求收据，交易记录

等待超时

收到收据

收到报价,接受价格;

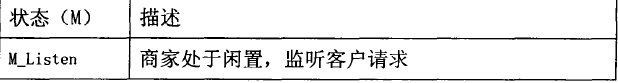
用户主动放弃

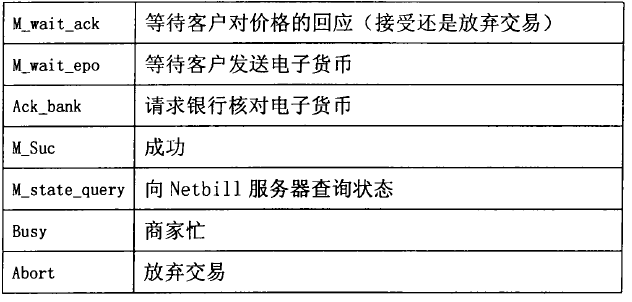
CosChan?abort,I 店家放弃

TransChan!what\_is\_price,\_pid,CosChan

交易失败

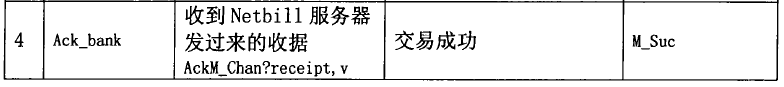
### 3.1.2商家状态及其行为





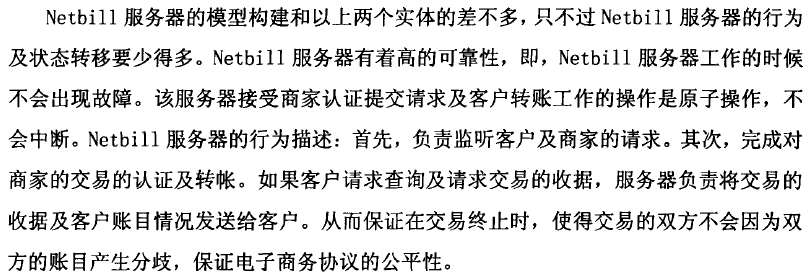
商家状态转移关系：







### 3.1.3银行状态及其行为



## 3.2 使用SPIN实现电子支付协议

## 3.3 使用LTL写出电子支付协议里面重要属性

### 3.3.1 钱的原子性

### 3.3.2 商品原子性

## 3.4 协议验证与分析

# 总结

## SPIN与SMV的使用区别

本人在做课后作业“农夫过河问题”使用的SMV工具来完成的。SPIN和SMV都是基于有限状态机来做形式化验证的。但是在使用过程中，个人感觉SPIN会比SMV具有更强的模型表达能力。因为SPIN是使用Promela语言来描述模型的，它可以很方便、自然的表达系统里面的各个并发进程。而SMV则是着重把状态机的状态转移表达出来，它在表达多个系统的同步转移的时候是需要进行一定的构造和转化的，它不想SPIN那样自然。

在输出结果上，SMV提供的错误反馈是一系列在执行路径上的各个状态的变量值的额序列，这一点非常类似于文本性的输出。这种过于简单的输出，使得寻找错误的过程变得非常困难和低效。而SPIN提供的错误反馈是整个协议执行的轨迹图，并且轨迹图直接编辑了发生错误（Assert失败）的具体行数。这使的分析变得更加便利和直观。

而且SPIN是具有图形化界面iSPIN的，因此其对用户的友好程度更佳。

## 课程的感受

很有幸在研究生第一学期选了这门课程。本人本科的专业是网络安全，而且是信工所二室信息对抗组的，之前的对常见的安全协议有比较多的接触，也接触了很多常见的协议攻击方式。这些协议的攻击方式里面，有一些是比较好构造的，比如中间人攻击，重放攻击。但是其他一些诸如类型错误攻击、反射攻击、预言机攻击（协议的参与方被当做解密机使用）则不是那么好构造，记得一开始接触反射攻击和预言机攻击的时候，觉着此类攻击特别巧妙，能够想到这样攻击的人一定是需要特别好的灵感。但是通过这门课程的学习，个人感觉这门介绍的使用工具进行形式化做安全协议验证的方式是一种特别普世通用的方法，借助于这一系列的理论工具和形式化工具，人们在设计安全协议的时候，就可以使用这些工具先验证一遍，这样在协议上线之前可以预先规避很多攻击。当然，这些工具只是验证工具，它会给我们举出一些破坏协议安全特性的攻击路径，但是它并不会指导我们怎么去设计一个协议。但是有了这个工具，最起码我们有了低成本的试错的机会，可以通过不断的“设计——>验证——>修改——>验证”来得到一个安全性优良的协议。这种工具就像是机器学习里面监督学习的“Teacher”，而我们的设计的安全协议就像是待优化训练的模型，虽然工具不会直接帮我们生成协议，但是它会给我们反馈当前的设计有没有满足我们需要的安全特性。

另外一点感受就是，之前没有看过形式化验证方面的研究论文，在老师的指导下，有幸拜读了几篇这个领域的最新论文，感觉这里面的套路基本就是拿这新的工具去对应用广泛的安全协议进行验证，然后对输出结果进行分析，提出其中一些新的漏洞。通过这种方式去发论文或许也是不错的方式，因为毕竟还有很多协议没有被验证。