路由协议的符号化测试生成

邢 熠[1),2),3)] 叶新铭[2)] 谢高岗[1)]

[1)](中国科学院计算技术研究所 北京 100190)

[2)](内蒙古大学计算机学院 呼和浩特 010021)

[3)](中国科学院研究生院 北京 100049)

摘 要 协议一致性测试用于验证协议实现的正确性.文中根据路由协议的消息复杂特点,提出基于 on-the-fly 策略符号测试生成与动态执行的算法:建立了一种新的统一符号语义模型,该模型把数据操作和控制都抽象为动作行为;以该语义模型为基础,扩展了行为之间的关系以及一致性测试关系;给出了基于 on-the-fly 策略符号测试生成与符号动态执行的算法,在符号动态执行中,使用了统计的聚类算法来进行符号的数据选择.论文最后用具体的测试例说明该算法在 OSPFv3 协议一致性测试上的应用.

关键词 协议一致性测试;统一语义模型;符号测试生成;符号执行;数据选择

中图法分类号 TP393 **DOI 号**: 10.3724/SP.J.1016.2010.00589

Symbolic Test Generation and Execution of Route Protocol

XING Yi[1),2),3)] YE Xin-Ming[2)] XIE Gao-Gang[1)]

[1)](*Institute of Computing Technology, Chinese Academy of Sciences, Beijing* 100190)

[2)](*School of Computer, Inner Mongolia University, Hohhot* 010021)

[3)](*Graduate University of Chinese Academy of Sciences, Beijing* 100049)

Abstract Protocol conformance test can verify the correctness of implementation under test. Regarding the characteristic of IP network routing protocol, a symbolic test generation and execution algorithm is provided based on the on-the-fly strategy. A unified operation semantic model is constructed which can unify data operation and abstract behavior. Symbolic conformance test relationship is given from the above model. A test generation and execution algorithm based on on-the-fly strategy is proposed. The symbolic execution data selection can be undertaken with the clustering algorithms. The method is illuminated with the example of the generating of OSPFv3 protocol conformance test cases.

Keywords protocol conformance test; unified operation semantic; symbolic test generating; symbolic execution; data selection

1 引 言

一致性测试可以有效地衡量网络协议的实现与协议规范说明的行为保持一致的程度,主要包括测试生成、执行.手工进行测试生成与执行具有效率低和覆盖不全面的缺点,自动化测试是改进这些缺陷的一个有效途径.协议规范的形式化模型是自动化测试基础,可以充分描述协议的行为与属性;通过形式化模型就可以定义被测实现与规范说明一致性的

收稿日期:2007-01-01;最终修改稿收到日期:2010-02-25.本课题得到国家自然科学基金(60403031,90604015,60903208)、国家"九七三"重点基础研究发展规划项目基金(2007CB310702)、中国科学院重大科研装备研制项目(YZ200824)资助.**邢 熠**,男,1971 年生,博士研究生,主要研究方向为网络协议的测试.E-mail: xingyi@ict.ac.cn.**叶新铭**,男,1943 年生,教授,博士生导师,主要研究领域为网络协议形式验证与测试.**谢高岗**,男,1974 年生,博士,副研究员,主要研究方向为对等计算、网络测量与服务质量.

标准，也就是一致性关系；根据一致性关系就可以产生抽象测试套，然后转化为可执行测试套，在测试执行系统上执行来验证被测实现是否满足一致性关系. 在协议测试实践中，由于协议自身的复杂性，对形式化模型、一致性关系、测试生成、测试执行提出了更多的挑战.

路由协议的突出特点是消息数据与行为的极大相关性. 根据消息数据的不同属性字段，会产生完全不同的行为. 如 OSPFv3 协议在建立邻居关系时，只有当收到期望的标识符，同时回应相应的响应才可能建立关系. 所以在路由协议的一致性测试时就要充分考虑这种数据对控制行为的影响，行为和数据是不可分的. 这些新的特点要求建模时就要统一考虑控制行为和消息数据.

现有的路由协议测试研究主要集中在两个方面，一个是基于控制行为的测试生成. 状态机[1-4]或标记变迁系统[5]模型进行测试生成主要关注协议说明的控制行为，而忽略了消息的处理，需要人工根据生成的抽象测试套加入消息数据形成最终的测试套，由于人工的介入，造成了测试套覆盖的主观性和不全面性. 另一个是符号技术的测试产生. 文献[6-9]在测试生成时，同时关注了控制行为和消息数据，而且消息数据采取了符号化的技术，但是对数据的处理和行为处理仍然采取独立的策略，同时忽略了符号化数据的选择问题.

由于已有测试模型不能有效地解决路由协议测试，本文的工作围绕以下 3 个方面展开：第 1 个方面是基础，提出了一种新的路由协议的需求模型，集中在操作语义层面，给出了扩展的符号化输入输出变迁系统 EIOSTS(Extended Input-Output Symbolic Transition System)，主要拓展了行为的范围，不仅包括控制行为，而且涵盖消息数据上的操作，使得控制行为和数据操作采取统一的形式进行描述；从扩展语义模型出发，提出新的一致性关系，即扩展的符号一致性关系. 第 2 个方面关于测试生成，本文采取 on-the-fly 和符号推迟策略来进行产生符号测试套，从而极大地减少了状态空间的容量. 最后是关于符号测试套的执行，符号的实例化通过统计的聚类方法实现.

根据以上的思路，文章按如下方式进行组织：首先提出协议操作语义模型，主要包括 EIOSTS 模型和相关的符号一致性关系；然后是符号化模型 EIOSTS 下测试框架，给出了该框架下的符号测试生成和符号测试执行算法，同时提出了基于聚类的数据选择算法；最后通过 OSPFv3 协议实例给出该生成算法的说明.

2 EIOSTS 模型

协议规范和被测实现的操作语义模型是产生测试的基础，本文使用扩展输入输出符号变迁系统 EIOSTS 描述协议规范和被测实现的行为活动和符号一致性关系. 下面是该模型的语法和标记.

2.1 EIOSTS 的语法和语义

为了方便研究，本文使用的消息数据采取变量的定义. X 表示变量集合，元素记为 $x,y,\cdots$. F 表示数值计算函数的集合，是从变量 x 到数值 v 的函数，记为 ρ，$\rho[v/x]$ 表示把变量 x 换成数值 v 后得到的结果. 对于路由协议，计算函数 ρ 包括解包 *Unpack*、打包 *Pack*、提取字段变量 *Get*、变量赋值 *Entail*、变量比较 *Choice*、变量计算 *Com*. Exp 表示表达式的集合，记为 e，它可以包含变量 x 和数值 v，$\rho(e)$ 代表在表达式 e 上进行的计算.

定义 1. EIOSTS(Input Output Symbolic Transition Systems)是一个 6 元组 $EIOSTS=\langle S,L,X,\Sigma,T,s_0\rangle$. 其中

(1) S 表示系统的可数状态集.

(2) s_0 表示初始状态.

(3) X 表示数据变量集，变量具有有限的值域.

(4) L 表示接口集. 为了讨论方便，假定接口 L 分成输入接口 L_I 和输出接口 L_O.

(5) 系统活动 $\Sigma=\{L_I?x,L_O!e,\rho(e)\}\bigcup\tau$，$\Sigma_O=\{L_O!e|e\in Exp\}$，$\Sigma_I=\{L_I?x|x\in X\}$，$\tau$ 表示内部动作，$\Sigma_m=\rho(e)$ 表示消息数据的处理.

(6) $T\subseteq S\times\Sigma\times S'$ 是变迁关系. 简记为 $S\xrightarrow{\Sigma}S'$ 表示系统从状态 S 经历活动 Σ 后到达新的状态 S'.

对于该模型，本文假定是强收敛的，即不会执行无限的内部活动. 同时弱输入是可能的，即在任何情况下都不能拒绝输入活动.

输入输出符号变迁系统可以图示为有向标记图，节点对应状态，标记边对应符号变迁.

2.2 EIOSTS 的行为描述

具备了以上的语义模型，就可以给出基于 EIOSTS 的行为描述. 为了讨论上的方便，采用如下标记：

$P=\langle S,L,X,\Sigma,T,s_0\rangle$；$s,s'\in S$；$a,a_1,a_2,\cdots,a_n\in\Sigma$；

ε 表示空活动；$\sigma\in(\Sigma)^*$ 表示活动序列，$O\subseteq\Sigma$ 表示活动集的子集. 定义行为如下：

(1) $s\xrightarrow{\rho(e)}s'$ 表示系统在状态 s 进行数据处理 $\rho(e)$ 后进入状态 s'.

(2) $s\Rightarrow^{a}s'\overset{\text{def}}{\equiv}\exists s_1,s_2\in S, s\overset{\varepsilon}{\Rightarrow}s_1\xrightarrow{a}s_2\overset{\varepsilon}{\Rightarrow}s'$ 系统从状态经过可观察活动 a 到达状态 s'.

(3) s after $\sigma=_{\text{def}}\{s'\in S: s\overset{\sigma}{\Rightarrow}s'\}$ 表示状态 s 经历某个可观察活动序列 σ 到达的某些状态 s' 集.

(4) s after σ refuse $O=_{\text{def}}\exists s': s\overset{\sigma}{\Rightarrow}s'$ and $init(s')\cap O=\varnothing$ 状态 s 经历可观察活动序列 σ 到达状态 s'，在状态 s'，系统不可能进行任何包含在活动集 O 中的活动.

(5) s after σ deadlocks $=_{\text{def}}$ after σ refuse Σ 状态 s 经历可观察活动序列 σ 到达状态 s'；在状态 s'，系统不可能进行任何活动，进入死锁.

2.3　EIOSTS 上的一致性关系

为了便于讨论，被测实现、协议规范、测试器都建模为 EIOSTS 同时采取如下标记：$u, i, s\in EIOSTS$. 文献[10]中定义了符号化一致性关系 eiosconf，下面是该定义的形式化描述.

定理 1.　$i,s\in EIOSTS(\Sigma_I,\Sigma_O), i\leq_{\text{eiosconf}} s$ iff

$$\forall\sigma\in(\Sigma_O\cup\Sigma_I\cup\Sigma_m)^*: out(i\text{ after }\sigma)\subseteq out(s\text{ after }\sigma) \quad (1)$$

该定理表明，对于任意由输出、输入、数据计算形成的行为迹 σ，如果在实现 i 执行所产生的输出包含在相应的规范上执行所产生的输出，就认为实现 i 与规范 s 满足一致性关系.

3　测试生成与执行

3.1　测试框架

在协议规范和被测实现的模型以及符号化一致性关系 eiosconf 基础上就可以进行测试例的生成. 我们仍然把测试例建模为符号变迁系统，根据文献[9]中测试例特性，采用如下的测试例定义.

定义 2.　测试例是一个 7 元组，$t=\langle S,L,X,\Sigma,T,s_0,\upsilon\rangle$，其中

(1) $\langle S,L,X,\Sigma,F,T,s_0\rangle$ 是确定的输入输出符号变迁系统，而且只有有限的活动.

(2) 测试例在任何状态 t' 只能在输入活动、输出活动、数据计算、等待或者静止中选择一种.

(3) $\upsilon: S\rightarrow\{\text{pass},\text{fail}\}$ 是断言函数.

测试套 T 是测试例 t 的集合.

3.2　测试生成与执行

基于符号变迁模型和变迁图的同构关系，本文采用变迁图的强连通图来产生测试例. 由于数据处理行为直接影响了测试的选择和执行，本算法采取了动态生成和执行的两步策略. 算法 1 是符号测试套的动态生成算法. 输入是协议规范和测试目的，测试目的的作用就是在测试生成的过程中裁剪掉与之无关的分支，从而大大降低了状态空间，输出结果是符号化的测试套，符号化的数据还没有进行数据选择. 算法 2 是符号测试的动态执行，输入是算法 1 的输出，即符号化测试套，在执行过程中动态地进行数据选择，从而实例化符号化数据，这样就可以裁剪掉与具体数据无关的分支，降低了时间和空间规模.

算法 1 首先生成协议说明和测试目的的组合图，然后根据测试目的选择性地生成测试套. 根据文献[11-12]给出图 1 的符号测试生成算法，输入是协议说明和测试目的的笛卡尔积，表示成符号变迁模型，即符号变迁图，测试目的增加了 Accept 和Reject 状态便于测试选择. 使用的数据结构 Dfs_Stack：保存搜索过程中当前遍历序列的堆栈，Scc_Stack 保存当下的联通分量 Scc 中所有已经访问的状态. 节点 s 是个结构化的变量 $State$. 包含 act：到达节点(状态) s 的图边；$number$：访问编号；$lowlink$：根节点编号；$L2A$：Leading to Accept state. 过程 Adj_Set(p)：$=\{(a,q)\mid(p,a,q)\in T\}$ 计算与状态 p 相连的状态和边的集合. SCC 的构建过程中，遍历的边可以分为以下 4 类：tree-arc 到达新状态的边，如果在同一个 SCC 中，就是 tree-arc$_{\text{in}}$，不同的 SCC 中，表示为 tree-arc$_{\text{out}}$；fronds 从子孙到祖先的边；short-cut 从祖先到子孙的边，同样分为 short-cut$_{\text{in}}$，short-cut$_{\text{out}}$；cross-link 从一个子树到另一个子树的边，细化为 cross-link$_{\text{in}}$，cross-link$_{\text{out}}$.

算法 2 根据符号数据的实例化对算法 1 得到的符号化测试套进行裁剪，形成最终可执行测试套. 输入是符号化测试套 T_s，输出是经过数据选择的实例化测试套 T_D. 数据变量的实例化主要通过函数来实现：$DataSelect()$. 根据文献[13-18]数据说明 SP 的穷尽测试集 $exhaustive$(SP)是数据项所有可能的实例化集合. 通常 $exhaustive$(SP)太大，并不实用，所以可以对被测实现的活动增加更强的假设来减少必要测试的数量，这些假设通称为选择假设，最常用的包括一致性假设和规则性假设. 数据实例化的过程

可以采取两种策略,一种是用数据代数的所有可能值首先实例化,然后再进行数据值的选择;另一种策略是在实例化的过程中就采取一定的策略进行实例化,从而避免了后面的测试选择过程.本文中采取后一种策略.

```
算法 1. 测试套生成:TG(State: p_start∈S×TP)
/** depth first search **/
State: p_source, p_target, p_pred;
Adj_set: Adj_source, Adj_target, Adj_pred
 begin
   Init(Dfs_Stack); Init(Scc_Stack);
   p_start.number := p_start.lowlink := i := i+1; p_start.act=ε;
   p_start.L2A := p_start∈Accept;
   if (p_start∈Reject) then
     remove all (p_start,a,p') from transitionsT
   Push((p_start, Adj_Set(p_start)), Dfs_Stack);
   Push((p_start), Scc_Stack);
   while not empty Dfs_Stack do begin
    (p_source, Adj_source):=top(Dfs_Stack);
    if not empty Adj_source then begin
     remove(m, p_target) from Adj_source;
     if p_target isn't numbered then begin /** tree-arc **/
       p_target.number := p_target
       lowlink := i := i+1;
       p_target.act := m;
       Push((p_target, Adj_set(p_target)),Dfs_Satck);
       p_target.L2A := p_target∈Accept;
       if (p_target∈Reject) then
         remove all (p_target,a,p') from transitionsT;
     else begin /** not tree-arc,is frond or cross-link_in **/
       if (p_target.number<p_source.number∧
           p_target in Scc_Stack) then
         p_source.lowlink := min(p_source.lowlink, p_target.number);
       if p_target∉Scc_Stack then begin
     if p_target.number<p_source.number then
       p_source.L2A := p_source.L2A∨p_target.L2A;
      if ¬p_target.L2A∧m∈T_I then
        remove all (p_source,a,p_target) from transitionsT;
     end //end of if p_target isn't…
    else begin /** Adj_source empty **/
     Pop(p_source,Dfs_Stack);
     if p_source.lowlink=p_source.number then begin
      /** p_source is root of SCC **/
      while (p := top(Scc_Stack)∧
              p.number≥p_source.number) do begin
       Pop(p,Scc_Stack);
       p.L2A := p_source.L2A;
       if ¬p.L2A then
         remove all (p,a,p') from transitionsT;
       end
     end
     if not empty(Dfs_Stack) then begin
     /** backtracking **/
        (p_pred,Adj_pred):=top(Dfs_Stack);
        p_pred.lowlink := min(p_pred.lowlink, p_source.lowlink);
        p_pred.L2A := p_pred.L2A∨p_source.L2A;
        if p_source.number=p_source.lowlink∧
          ¬p_source.L2A∧m':=p_source.act∈T_I
         then
          remove all (p_pred,m',p_source) from transitionsT;
       end
     end
   end
 end
```

图 1 符号测试生成算法

```
算法 2. 测试套裁减:TD(State: P_start∈T_s)
/** breadth first search **/
 State: p_source, p_target.
  begin
    Init(Bfs_Stack); Init(TC_Stack);
    push((p_start, Adj_Set(p_start)), Bfs_Stack);
    while not empty Bfs_Stack dobegin
      (p_source,a,p_target):=Pop(Bfs_Stack);
      if p_target==(True|Accept|Refuse) then begin
        switch(a) {
         case!:
           getmessage(m);
           if (CheckConstraint(m)==True) then begin
             if (p_target==Accept) then begin
               Push((p_source,a,Pass),TC_Stack);
               break;
             end
             if (p_target==Refuse) then begin
               Push((p_source,a,Fail),TC_Stack);
               break;
             end
           Push((p_source,a,p_target),TC_Stack);
         case?:
           DataSelect(a,m); /** a=ρ(e) **/
           Outmessage(m);
           if (p_target==Refuse) then begin
             Push((p_source,a,Inconclusive),TC_Stack);
             break ;
           end
        }
      end
      p_source=False;
      Clear(Bfs_Stack);
      Push((p_target, Adj_Set(p_target)), Bfs_Stack);
    end
  end
end
```

图 2 符号执行算法

因为在执行过程中动态生成和执行测试例,所以必须有相关的支持环境.过程 getmessage(m),CheckConstraint(m),DataSelect(a,m),Outmessage(m)都是执行系统提供的支持服务.下面只介绍数据选择算法的架构.

3.3 数据选择算法

消息的数据选择过程采用了基于统计的聚类选择方法,使用框架法来表示消息.

为了提高数据值的真实性,我们采取了基于网络监测的统计方法来生成数据框架的具体值.数据的来源是网络中实际运行节点所交换的各种消息样本.由于现实网络中消息中每个字段取值的多样性,采取聚类的方法将那些看上去会自然落在一起的样本集合在一起,形成相应的数据分类.最终的数据值会存入框架数据库中,以备后来的数据选择.

现有的聚类算法主要是以下几种,即 k 均值聚类算法、递增聚类算法和基于概率的算法.由于协议消息字段都是数值类型,同时聚类的目标是寻找数据的最有可能的聚类,所以采取基于统计的聚类算法.

统计聚类的基础是建立在一个称为有限混合的统计模型上.混合是指用 x 个概率分布代表 x 个聚

类，对每个具体数据实例，每个分布会给出它实属这个聚类的概率，每个聚类都有不同的分布. 各个聚类并不是同等可能的，概率分布可以反应这种差异. 这种算法的前提条件是已经知道了数据实例的概率分布和分布参数，通过贝叶斯公式很容易计算数据属于某个聚类的概率.

假设论域上有 m 个聚类 $\omega_1,\omega_2,\cdots,\omega_m$；$x$ 是数据实例，x 属于聚类 ω_i 的概率如下：

$$p(\omega_i/x)=\frac{p(x/\omega_i)\times p(\omega_i)}{\sum_{i=1}^{m}p(x/\omega_i)\times p(\omega_i)} \quad (1)$$

其中 $p(\omega_i)$ 表示聚类 ω_i 的先验概率. $P(x/\omega_i)$ 表示当输入 x 属于聚类 ω_i，x 出现的条件概率. $P(\omega_i/x)$ 表示给定输入 x，x 属于聚类 ω_i 的后验条件概率. 它表明 x 发生的相对频率，值越大，表示发生的相对频率越高.

贝叶斯聚类法则：若存在 $i\in\{1,2,\cdots,m\}$，使得对所有的 $j(j=1,2,\cdots,i-1,i+1,\cdots,m)$ 均有

$p(\omega_i/x)>p(\omega_j/x)$，则 $x\in\omega_i$.

因为本文中处理的数据只有一个正整数属性，所以根据大数定理可以假定每个聚类呈正态分布.

算法的输入取自网络监测中的协议消息样本，我们以 OSPFv3 消息的数据聚类为例在图 3 中说明

算法 3. 数据聚类算法：DataCluster(m)

```
输入：
  m={x1,x2,…,xn}是消息字段集，
  xi={di1, di2,…, din}是字段的样本数据集
DataCluster(m)
{
 Prepare(m)
 /*获得消息样本，并进行预处理，满足聚类的输入要求*/
 swith(m.type)
 {
   case hello: /**邻居发现的聚类处理**/
   {
     for (i=1; i<=n; i++){
     /**对每个字段数据样本分别进行聚类处理**/
     BayesCluster(xi);
     SaveInDataBase(xi);
     /*把聚类的数据存入数据库，用于测试数据选择*/
     }
     break;
   }
 case Database Description: /**数据库描述的聚类处理**/
 {…}
 …
}
BsyesCluster(x)
{
/**基于贝叶斯统计的聚类算法 x={d1,d2,…,dn}**/
Init_D={D1,D2,…,Dn}
While (n>1)
{
  p(ωi/x)=f(x;ui,σi)×p(ωi)
  //根据正态分布的贝叶斯聚类式(2)计算聚类概率
}
p=max{p(ωi/x)}
Di=Di∪{x}/**合并聚类集合**/
}
```

图 3　数据选择算法

了算法. 经过上述贝叶斯聚类算法，就可以形成每个字段的实际数据的聚类分析. 只要把每个聚类的平均值存入框架数据库，在进行数据选择时，就可以根据聚类的种类来指导测试数据的生成. 如在 OSPFv3 协议 hello 消息的 helloInterval 和 RouterDeadInterval 字段就可以通过聚类算法来获得实际的参考值.

3.4　算法分析

在算法 1 的测试生成阶段，根据图论的结论，复杂度可以控制在 $O(n+|e|)$ 以内. 关于效率，首先由于测试目的的介入，大大减少了符号化测试例的数量，根据我们的实验研究，可以减掉 80% 以上的无用边，只留下了与测试目的相关的状态和变迁，形成了导向正确终点和错误终点的连通图. 在算法 2 的测试执行阶段根据数据的动态选择策略，随机地执行变迁序列，从而裁剪了不可达的变迁，最终形成了测试变迁序列.

与已有的符号化测试方法比较而言，TGV 是提供了符号化的测试生成算法，但是在数据建模上，仍然采取了与控制行为不同的方式，如一阶逻辑等方法，这样就割裂了数据与行为之间的天然联系，所以在生成阶段数据只是符号的表示，没有充分体现数据的重要作用，并且没有延伸到符号的执行，形成了执行的空白. 另一个工具是 Torx，它提供了测试生成与执行的同步，但是没有考虑生成过程中数据的建模和执行过程的数据选择问题，而且在执行时都是随机选择执行序列，增加了测试的不确定性.

本文在已经研究的基础上，统一了数据处理与控制行为的系统建模，在测试生成算法过程中采取了 on-the-fly 策略从而大大减少了测试例数量，最后在执行算法过程中引入了基于聚类的动态数据选择，又进一步减少了测试例的数量.

4　应用举例

下面通过 OSPFv3 协议作为实例来说明上面提出的测试生成算法. OSPFv3 协议规范说明 $S_{\text{OSPFv3}}=\langle S_s,L_s,X_s,\Sigma_s,T_s,s_{s0}\rangle$，测试目的 $TP=\langle S_{TP},L_{TP},X_{TP},\Sigma_{TP},T_{TP},s_{TP0}\rangle$. S_{OSPFv3} 用图 4 表示.

该协议图表示在收到 hello 消息后，经过 one-way 状态进入到 two-way 状态，hello 代表 hello 数据包，DD 表示数据库描述符包. 测试目的 S_{TP} 用图 5 来说明，该协议通过发送 hello 数据包建立双向关系. 协议规范与测试目的的笛卡尔积在图 6 中给出.

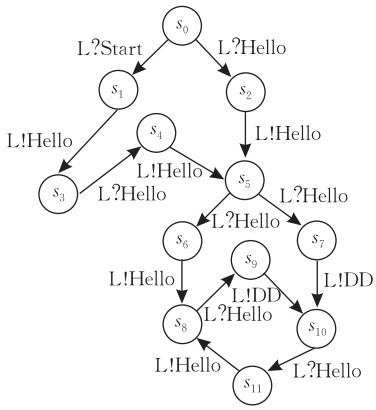

图 4 协议规范

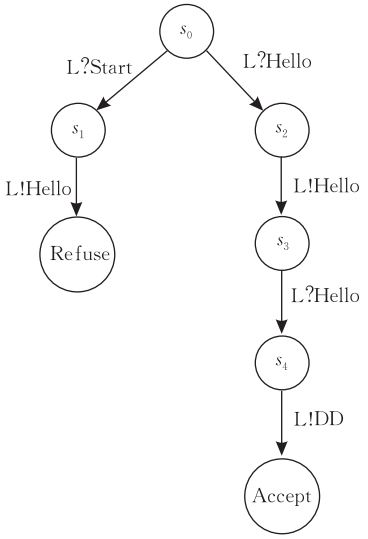

图 5 测试目的

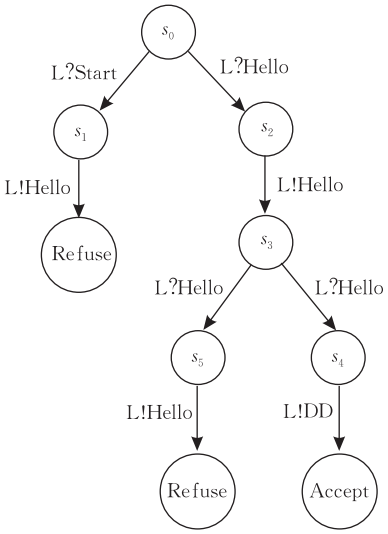

图 6 协议规范与测试目的的笛卡尔积

执行算法 1、2 后所得到的测试例如图 7 所示，测试例中引入了 Pass、Fail 状态. 图 7 中 Hello_T1 和 Hello_T2 消息都是测试器动态实例化后所得到的数据包，图 8 是以我们自己设计的测试例描述语言进行描述，该种描述语言已经移植到商业应用.

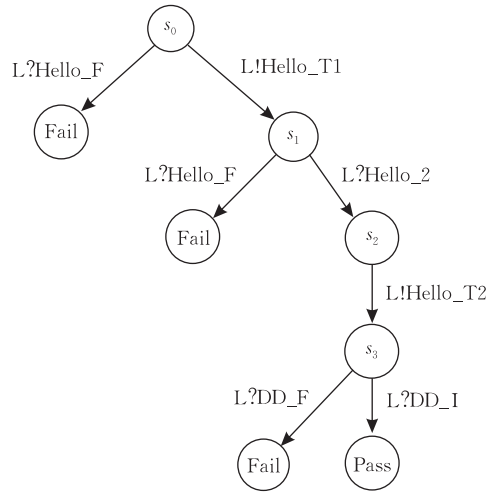

图 7 测试例

在上述描述中，有些字段是通过聚类算法获得的取值，有些字段没有取确定的数值，而是用变量来表示，主要是为了方便用户的测试. 用户可以在测试前和测试中根据具体值来实例化该字段，这种方法增加了执行时的灵活性.

```
数据包格式：
packet Hello_Init {
 header OSPF {
            Version=3//版本类型
            Type=1//包的类型
            Router_ID= $TN_RID//路由器 ID 号
            Area_ID= $NUT_AID//区域 ID 号
            Instance_ID=0//进程实例 ID 号
            Message_Body=[cat $TN_Interf1 01 00 00 13\
                          00 0a 00 28 00 00 00 00\
                          $TN_RID]//消息体内容
            }
            header IPv6 PCO1 {
          Destination_Address= $AllSPFRouters//目标地址
          Next_Header= $OSPF
 }
            OSPFChecksum
}
```

图 8 消息数据描述

5 结 论

现实协议的测试需求引入了符号化的测试模型，现有的基于符号化模型在进行测试生成时，截然分成两个阶段，第 1 个阶段是生成符号化的测试例，根据生成的策略，又可进一步分成静态生成和动态生成；第 2 个阶段是符号化测试套的执行，根据符号实例化的时机，也可以分成静态赋值和动态赋值. 本文根据协议的现实需求，把测试生成和测试执行集成在一起，而且都采取了动态策略，这样就充分体现

了符号化模型的优势，避免了由于数据引入所造成的空间爆炸，在很早的阶段就裁剪了无用的分支，从而节约了空间和时间，极大地提高了效率.

当然，在我们的实践中，对模型进行了一些假设，这不可避免地与现实有一些差距，主要是在数据的选择上不够智能化，希望在以后的工作引入学习机制，可以更好地选择与实际接近的数据.

参 考 文 献

[1] Petrenko Alexandre, Yevtushenko Nina. Testing from partial deterministic FSM specifications. IEEE Transactions on Computers, 2005, 54(9): 1154-1165

[2] Lee David, Yannakakis Mihalis. Principles and methods of testing finite state machines — A survey. Proceedings of the IEEE, 1996, 84(8): 1090-1123

[3] Chen J, Hierons R M, Ural H. Resolving observability problems in distributed test architectures//Proceedings of the FORT2005. Taipei, Taiwan, China, 2005: 219-232

[4] Miller R E, Chen D-L, Lee D, Hao R. Coping with nondeterminism in network protocol testing//Proceedings of the TestCom 2005. Montreal, Canada, 2005: 129-145

[5] Brinksma E, Tretmans J. Testing transition systems: An annotated bibliography//Proceedings of Summer School MOVEP'2k Modelling and Verification of Parallel Processes. Nantes, 2000: 44-50

[6] Ingolfsdottir A, Lin Huimin. A symbolic approach to value-passing processes//Bergstra J A, Ponse A, Smolka S A eds. Handbook of Process Algebra. Amesterdam: Elsevier, 2001: 427-477

[7] Rusu V, du Bousquet L, Jeron T. An approach to symbolic test generation//Integrated Formal Methods-INMF 2000. Lecture Notes in Computer Science 1945. Springer-Verlag, 2000: 338-357

[8] Frantzen Lars, Tretmans Jan, Willemse Tim A C. Test generation based on symbolic specifications//Proceedings of the FATES 2004. LNCS 3395. 2005: 1-15

[9] Faivre Alain, Gaston Christophe. Symboilc model based testing for component oriented system//Petrenko A et al eds. Proceedings of the TestCom/FATES 2007. LNCS 4581. Springer, 2007: 90-106

[10] Tretmans J. Conformance testing with labeled transition systems: Implementation relations and test generation. Computer Networks and ISDN Systems, 1996, 29(1): 49-79

[11] Jéron Thierry, Morel Pierre. Test generation derived from model-checking//Proceedings of the 11th International Conference on Computer Aided Verification, 1999: 108-121

[12] Tarjan R. Depth-first search and linear graph algorithms. SIAM Journal on Computing, 1972, 1(2): 146-160

[13] Lestiennes G, Gaudel M-C. Testing processes from formal specifications with inputs, outputs and data types//Proceedings of the 13th International Symposium on Software Reliability Engineering, 2002: 3-14

[14] ISO/IEC JTC1/SC21 WG7, ITU-T SG 10/Q. 8 Propsed ITU-T Z. 500 and Committee Draft on "Formal Methods in Conformance Testing" CD 132451. ISO-ITU-T, Geneve, 1996

[15] Tretmans J. A formal approach to conformance testing [Ph. D. dissertation]. University of Twente, Enschede, The Netherlands, 1992

[16] Bernot G, Gaudel M-C, Marre B. Software testing based on formal specification: A theory and a tool. IEE Software Engineering Journal, 1991, 6(6): 387-405

[17] Gaudel M-C. Testing can be formal, too//Proceedings of the 6th International Joint Conference CAAP/FASE on Theory and Practice of Software Development, 1995: 82-96

[18] Belinfante A, Frantzen L, Schallhart C. Tools for test generation//Broy M et al eds. Model based Testing of Reactive Systems — A seminar. LNCS 3472. Springer Verlag, 2005: 391-438

XING Yi, born in 1971, Ph. D. candidate. His current research interests focus on network protocol test.

YE Xin-Ming, born in 1943, professor, Ph. D. supervisor. His main research interests include network protocol verification and test.

XIE Gao-Gang, born in 1974, Ph. D., associate professor. His main research interests include peer-to-peer computing, network measurement and QoS.

Background

The research of next generation internet has been concentrating on the correctness of IPv6 routing protocols. Conformance test is an effective method to validate the correctness of the implementation of IPv6 routing protocol. Symbolic conformance test is used to meet IPv6 routing protocols' requirement that their behaviours and messages should depend on each other. Current researches are devoted to using symbolic techniques to express simple data. However they only focus on symbolic test generation without symbolic data selection, symbolic execution and evaluation. This doesn't really solve the problem of the data explosion but just defer it. This paper constructs a unified symbolic test method which includes symbolic model, symbolic conformance relation, symbolic test generation, symbolic test selection, symbolic test execution. It works perfectly for IPv6 protocols.