计算机科学2004Vol.31№.12

基于SDL和MSC模型的一致性测试生成方法*)

叶新铭　吴铁楠
(内蒙古大学计算机学院　呼和浩特010021)

摘　要　本文提出了一种基于协议的SDL和MSC描述的一致性测试生成方法。这种方法从协议的形式化描述出发，用形式化的语言定义测试目的和测试组，通过本文提出的算法自动生成一致性测试套。

关键词　SDL,MSC,一致性测试,测试套

The Generation of Conformance Test Suite Based on SDL and MSC

YE Xin-Ming　WU Tie-Nan
(Department of Computer Science,Inner Mongolia University,Hohhot 010021)

Abstract　In this paper,we present a method of conformance testing generation based on the SDL and MSC. This method will automatic generate conformance test suite. It started form the formal description of protocol,and defined test purpose and test group using the formal language SDL and MSC.

Keywords　SDL,MSC,Conformance testing,Test suite

ISO 9646中提供了一种非形式化的一致性测试的方法[1]。在这个方法中，测试目的是非形式化定义的，测试目的与测试例之间没有一个形式化的关系。所以一个测试例可能与他们被假设覆盖的测试目的不符。因为非形式化方法的不足，所以有很多研究提出了形式化的方法进行一致性测试套的生成方法[2~4]。这些方法也都各有不足，有的方法是通过人工定义测试目的，然后进行测试套的自动生成[5]，有的方法是生成测试例的过程不能自动完成。

本文提出的方法是要在描述协议的SDL模型和MSC模型基础上自动生成一致性测试的测试套。这个方法将从协议的形式化描述出发，根据模型自动产生测试目的，通过形式化语法说明测试组和测试目的，分别产生测试例中的测试体和前导，最终得到测试套。

本文是基于协议的形式化描述模型，也就是基于协议的SDL和MSC模型。因为这两种建模语言各自具备不同的特点，所以协议的SDL模型将产生基于控制流的测试例，而协议的MSC模型将产生基于数据流的测试例。所以用本文提出的自动生成测试套的方法得到的一致性测试套是结合了控制流覆盖和数据流覆盖的测试套。

1　基本定义

定义1　测试套和测试例[1]

ISO 9646中提出了一种一致性测试方法，这种方法定义了测试例的组织结构。一个测试套被分成若干个测试组，每个测试组用于测试一个较大的目的。每个测试组又分成若干测试例，测试例测试较明了的目的。最后，每个测试例由原子的测试步组成。

一个测试例分为两部分：

前导(Preamble)：用于定义从测试例的起始点到测试体开始状态要经过的测试步。

测试体(Body of the test)：是一系列测试步，这些测试步用于检验测试目的，并为可能得到的结果做出判定。

定义2　形式化语法[5]

```
test:=[sub_objectives '>>'][IOs '>>' ]objectives;
sub_objectives := collection_of_labels { ';'collection_of_labels };
IOs:= collection_of_labels ;
objectives := collection_of_labels ;
```

其中sub_objectives和objectives都是状态集合，IOs是输入输出集合，它们都是用标记(label)表示的。'>>'是连接符，用[]括起来的内容表示是可选项。用这个形式化语法可以描述测试组和测试目的，描述测试目的的时候必须带有IOs，而描述测试组的时候不带有IOs。

2　算法思想

算法是从协议的形式化描述开始，通过划分测试组和确定测试目的，得到一致性测试套。

算法1　自动生成一致性测试套

```
AutoGenerConfTestSuit (SDLmodel, MSCmodel)
{ Identify the informal test purpose; //根据模型确定非形式化测试目的
  DivideTestCaseGroup(); //划分测试组
  Formal describe the Test Purpose; //测试目的的形式化描述
  GenerTestCase(); //测试例的生成
  Trace inversion; //从IUT的角度转换到测试者的角度
}
DivideTestCaseGroup()
{ Identify states; //定义系统模型中的状态
  Formal describe Pattern of Test Case; //对测试组进行形式化描述
}
GenerTestCase()
{ Trace extraction; //获得描述中允许的进行的期望的活动路径
  Trace completion; //补充发生在测试者控制之外的其他路径
  Body generation; //根据输入输出活动树确定每个测试例中的测试步
  Preamble generation; //确定完成前导所需的输入输出序列
}
```

3　从模型产生测试目的

因为有明确定义的测试目的是开始生成测试套的前提，所以首先要做的就是确定测试目的。根据协议的SDL和MSC模型来确定非形式化的测试目的，在后面的步骤中再用定义2中的形式化语法进行。

*)基金项目：国家自然基金项目(60263002)，内蒙古科技攻关项目(2002061002)。叶新铭　教授，博士生导师，主要研究方向：网络协议形式验证与测试。吴铁楠　硕士研究生，主要研究方向：网络协议形式验证与测试。

对协议的测试是黑盒测试，在测试过程中仅关心测试下实现(IUT)的输入和输出。认为每个输入和输出都对应一个测试目的。输入所对应的测试目的用于测试IUT能否正确接收协议规定的数据包，输出所对应的测试目的用于测试IUT是否在一定条件下，对输入事件做出正确的响应，这些测试目的都要根据IUT的响应来判断IUT是否与协议说明相一致。

·输入：包括IUT从控制观察点(PCO)收到的协议包和定时器超时所引起的信号输入。在SDL中，输入就是input；在MSC中，输入是found message。

·输出：包括测试下实现(IUT)从控制观察点(PCO)发送出去的协议包。在SDL中，输出就是代表信号输出的output；在MSC中，输出是代表进程实例发送一个消息的message。

4 划分测试组

在测试组的形式化过程中，为了能作为后面步骤的参考，必须在FSP中确定相关的状态。本文采用形式化描述技术SDL和MSC来描述OSPF协议[6]。在SDL模型中含有状态节点，用来表示特定的状态。而MSC模型所描述的是协议模型中的消息交换，这是在协议SDL模型中某一状态下进行的，终止于另一状态。对于MSC模型来说，两个状态：起始状态和终止状态。要给SDL和MSC模型中所有的状态做好标记(labels)，以便在后面的步骤中引用。

接下来，协议的形式化描述(FSP)中带标记的状态将像地界标一样用于测试组的确定。这种思想是在一系列带标记状态中建立一种逻辑关系，从而划定一个特定测试例中的前导和测试体之间的界限。这样得到的就是测试组。

测试组(GTC，Group of Test Case)，用于标识一组测试例。一个测试组，就是一系列子目标集合后面跟一个目标。在FSP中子目标和目标都是通过标记(labels)来标识的。标记集合为通过一系列事件到达[子]目标提供了可选项。

基本地，测试组(GTC)有如下的结构：

$test\_group := S_1^1 | S_2^1 K; S_1^2 | S_2^2 | K; K \gg o_1 | o_2 K$

在中间的S_i^j是子目标，O_i是最终目标。子目标和目标都是在FSP中定义的标记名。一组中的所有子目标(比如S_i^1)是一个可选集，其中有一个并且只有一个会被访问到。访问完S_i^1中的一个子目标后，将访问S_i^2中的一个，如此类推。最后，我们将访问到一个最终目标O_i。

一系列要被访问的子目标是一组测试例的前导。从最后一个子目标到一个最终目标，我们就得到了一个测试组。前面表示的测试组描述语言允许在所有目标和子目标中设一个最大可观察事件的数目的上界和整个测试例的最大长度。

这个阶段的整体结果是得到一棵FSP的状态树。这棵树上的每个节点都是一个FSP中的状态。树的根节点是协议的起始状态。因为FSP中可以有环路存在，所以树上的节点与FSP中的状态并不是一一对应，而是多对一的关系，也就是说可能多个节点都对应着同一个状态。这棵树上的路径就代表着协议可以进行的状态转换次序。树上的节点，即FSP中的状态，将整个测试套划分成不同的测试组。在SDL模型中，任意相邻的两个状态间的测试例属于同一个测试组。因为在MSC模型中，MSC图形中的所有活动都是在两个状态间进行的，所以期间所有的测试都属于同一个测试组。

5 形式化描述测试目的

在确定非形式化的测试目的的时候，已经确定了对应测试目的的输入和输出，现在要做的是给所有的输入输出做好形式化的标记。这些形式化的标记结合上节标记的状态将用于对测试目的进行形式化描述。

测试目的(PTC，Purpose of Test Case)，用于标识测试例。一个测试目的，就是一系列子目标集合后面跟一系列输入输出，最后再跟一个最终目标。在FSP中子目标和目标及输入输出都是通过标记(labels)来标识的。从它推导出的测试例可能受到给定长度的限制。

基本地，一组测试目的有如下结构：

$test\_purpose := S_0 S_1 K S_i \gg [i_1, i_2, K, i_n, o_1, o_2, K, o_m] \gg O$

其中$S_0 S_1 K S_i$是测试例前导中经历的一系列状态，是从测试组$[S_1^1 | S_2^1 K; S_1^2 | S_2^2 | K; K]$中取出的一条最优路径。这条最优路径应该具备有效、经济和最短的特点。S_i是这个测试例方案中最后一个子目标，也就是前导中的最后一个状态。到达S_i后，就开始了本组测试目的的测试体部分。O是S_i的下一个状态，是从测试例方案$[O_1 | O_2 K]$中根据FSP选定一个目标。$[i_1, i_2, K, i_n, o_1, o_2, K, o_m]$是在$S_i$和$O$两个状态之间的一系列有序的输入输出。这些输入输出序列分别对应着测试目的。一部分测试目的能使FSP从S_i状态到达O状态，但并不是所有的测试目的都可以做到。

经过这个步骤，得到状态树中任意两个相邻状态间的输入输出活动树如图1所示。这些树是由三部分组成，分别对应着测试目的的形式化描述中的三个部分。第一部分对应测试目的的前导，从树的根开始的没有分支的子树，路径上的节点都是子目标。第二部分是输入输出的活动子树，子树的节点都是输入或输出，对应测试例体中的测试步。第三部分是一个单独的节点，对应测试目的的目标，输入输出的活动子树中的叶子节点是这个单独节点的前驱，这个单独节点没有后继节点。

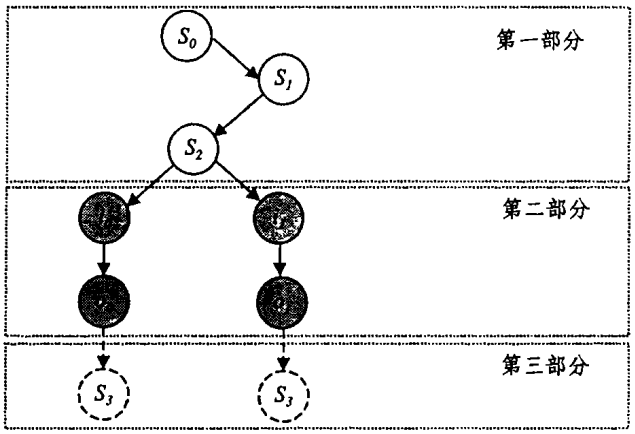

图1 输入输出活动树

6 测试例的生成

从前面的步骤得到一棵状态树和多棵各相邻状态间的输入输出活动树。下面将应用前面得到的结果，生成测试例。这个过程可以分成四个阶段：

1. 路径选取：获得协议描述中允许进行的活动路径。
2. 路径补充：补充发生在测试者控制之外的其他路径。
3. 生成测试步：根据输入输出活动树确定每个测试例中的测试步。
4. 生成前导：确定相邻状态间转换所需的输入输出，从而确定一系列从S_{init}状态到S_{body}间转换所需的输入输出，完成前导。

6.1 路径选取

进行路径选取也就是要生成测试例的测试体部分。测试体就是一系列测试步，这些测试步是检验测试目的和为可能的结果做出判定的关键。

在一组测试目的中，比如：

$S_0S_1KS_k \gg [i_1,i_2,K,i_n,o_1,o_2,K,o_m] \gg O$

$[i_1,i_2,K,i_n,o_1,o_2,K,o_m]$是在 S_k 和 O 两个状态之间的一系列有序的输入输出。这些输入输出序列就是所对应的测试目的的测试体。

从算法上看，可以用一些扩展的定理法则来获得路径。一些工具允许生成描述的变迁树。扩展部分通过长度限制被删去了，只有那些访问预期的目标的分枝被保留下来。

这个阶段的整体结果是一个 FSP 的完全描述的子树。这棵树上的路径就是对应其测试目的的测试组。路径上的部分节点被标记为 S，表示成功：到达标有 S 的节点意味着测试成功通过。

路径选取阶段可以得到从一个状态到下一个相邻状态的输入输出序列，这是后面步骤中要利用的中间结果。

6.2 路径补充

要获得一个考虑了所有 IUT 的可能相应的测试套，我们需要加入所有有效的会妨碍我们达到测试目的的 IUT 行为。对于从测试方案生成的树 T 的每一个中间状态，我们必须预测所有发生在控制之外的可能发生的事件和接下来相应的路径，直到确定再没有可能的路径。经过路径补充之后的输入输出活动树就是 $C(T)$。

6.3 生成测试步

现在我们有了一组从 FSP 中选取的行为。它对应着一个具体的测试目的，简化了 IUT 的行为，测试例就是从 IUT 中选取的。

如果 T 中只有一条路径，选择就很简单了：只有一个测试例，就是 T。如果 T 中有多条路径，并有若干个带有判定 (S) 的节点，我们就必须选取和上述节点一样多的测试例，换句话说，就是 T 中有多少带判定的节点就从 $C(T)$ 中选择多少测试例。从算法上讲，从 $C(T)$ 中每选择一条路径，就去掉 T 中通向该节点的其他路径。注意在补充阶段加上的路径不能被去掉。

在生成测试步的过程中，可以得到任意两个状态间转换所需的测试步，这个结果将在后面生成前导的步骤中得到充分利用。

6.4 生成前导

测试例由前导和测试体两部分组成，前面已经生成了测试体，接下来将要生成前导部分。生成前导分两个步骤：1. 生成前导状态序列。首先确定前导所包含的一系列有序状态，是从系统的初始状态开始到测试体开始的状态为止；2. 生成前导测试步。利用前面得到的结果，将状态序列展开成能够完成状态转换的测试步就得到了前导测试步。

1. 生成前导状态序列

任何一个测试例都是从某个状态 S 开始的，S 之前的所有输入输出测试步都是该测试例的前导。测试例的前导用于定义从测试例的起始点到测试体开始的初始状态之间的路径上必须的测试步。在前导的定义中，我们能看到两个定义清楚的状态：测试例开始的状态 (S_{init}) 和测试体开始的状态 (S_{body})。在这两个状态之间可能还有若干个状态，也可能没有其它状态。如果这个测试例的测试目的是关于协议初始状态的，那么测试例开始的状态 (S_{init}) 和测试体开始的状态 (S_{body}) 就是同一个状态。

从前面得到的状态树中，可以知道状态之间的序列关系。测试例开始的状态 (S_{init}) 是状态树的根节点。在状态树找到所有的测试体开始的状态 (S_{body})，然后找到每个 S_{body} 到 S_{init} 的路径，从中选出最优路径。如图2所示。选举最优路径的原则是选择有效、经济和最短的路径。

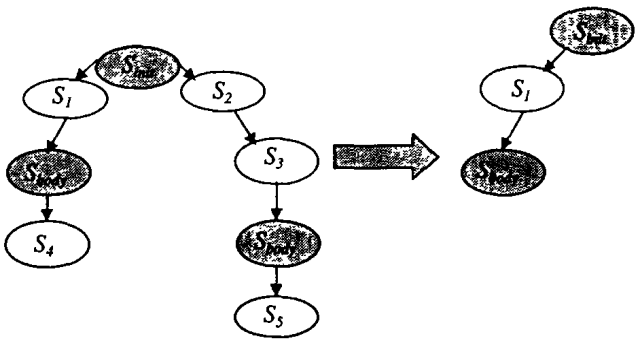

图2 状态最优路径

从 S_{init} 到 S_{body} 的路径有两条，分别经过 (S_1) 和 (S_2,S_3)，根据最优原则选择了经过 (S_1) 的路径。

这样想开始某个测试例的测试体先要经过一系列测试步使协议从 S_{init} 到达该测试例的 S_{body}。使 IUT 从一个状态到达另一个状态的测试步可以从上面的步骤中获得。

2. 生成前导测试步

先利用前面得到结果确定任意两个相邻状态 S_i 和 S_{i+1} 间转换所需的输入输出序列；从而确定一系列从 S_{init} 状态到 S_{body} 状态间转换所需的输入输出序列。

经过这两个步骤就可以得到测试例的前导，将它与前面小节中得到的测试体合并在一起就得到完整的测试例。

7 转换角度

最后，我们必须从 IUT 的角度转换到测试者的角度。也就是说，我们必须考虑 $C(T)$ 中的每条路径，从自观察者的角度来看待它。而不是从一个 IUT 的角度。因为本文做的是主动测试，所以由测试者来决定 IUT 可能的行为。

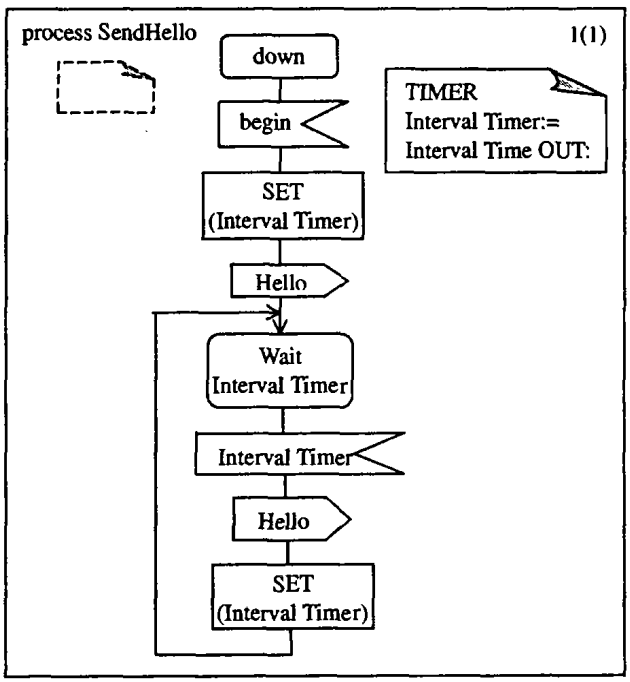

图3 SDL 模型的 SendHello 进程

8 算法举例

本小节将以发送 Hello 消息的部分为例说明上文提出的基于形式化的方法生成一致性测试套。SDL 模型的 SendHel-

lo 进程如图3所示[3]。

第一步：确定非形式化测试目的。根据 SDL 模型（见图3），系统将收到三种消息，?begin、?IntervalTimer 和 !hello。输入输出对应着测试目的，所以系统的这部分模型中包含三个测试目的，即能够接收到 begin、IntervalTimer 和发送 Hello 消息。

第二步：划分测试组。先标记系统模型中的状态：系统中有两个状态，分别是 down 和 Wait_IntervalTimer。

再对测试组进行形式化描述，根据模型中状态的转换关系，将分为两个测试组：

1) down>>Wait_IntervalTimer

2) Wait_IntervalTimer>>Wait_IntervalTimer

以从 Wait_IntervalTimer 状态到 Wait_IntervalTimer 状态之间的测试例为例说明对测试组的描述：down;Wait_IntervalTimer>>Wait_IntervalTimer

第三步：对测试目的进行形式化描述。

down;Wait_IntervalTimer>>[IntervaTimer,Hello]>> Wait_IntervalTimer

这个步骤得到一个输入输出活动树，如图4所示。

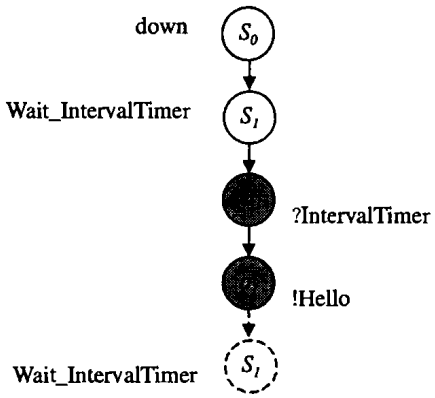

图4 发送 Hello 消息的输入输出活动树

第四步：生成测试例。

1. 获得描述中允许的进行的期望的活动路径：?IntervalTimer,!Hello

2. 这个例子中不需要补充，所以没有加入其它路径。

3. 根据输入输出活动树确定测试例中的测试步：?IntervalTimer,!Hello

4. 确定完成前导所需的输入输出序列

1)先确定前导状态序列：down，Wait_IntervalTimer

2)再确定相邻状态间状态的输入输出活动序列：?begin

从而推出前导输入输出活动序列为：?begin

将4中得到的前导输入输出序列分别和3中得到的两个测试例的测试步合并，就得到一个完整的测试例：?begin,?IntervalTimer,!Hello

第五步：转换角度。将测试例中的输入输出从 IUT 的角度转换到测试者的角度，即将接收?转换为发送!，发送!转换为接收?。

最终得到测试例：!begin,!IntervalTimer,?Hello

小结 本文提出的方法是通过一连串简单步骤来实现，简单的步骤在控制下保持自身的复杂性。保持测试套按照清晰的测试目的进行分组可以控制整体的复杂性。算法的设计并不局限于某个具体的协议。所以这个方法不仅能够完成 OSPF 协议的一致性测试套的生成，还能应用于其它的路由协议，比如 RIP 和 BGP 等路由协议。

本文提出了一种基于 SDL 和 MSC 模型的一致性测试的生成方法，它的优点如下：

1. 从协议的 SDL 模型和 MSC 模型出发产生的测试套能达到功能覆盖和数据覆盖；

2. 测试组和测试目的都是通过形式化方法进行描述，易于抽象；

3. 没有人为因素，利于保证测试例的正确性；

4. 测试例的选择容易执行；

5. 在整个生产生命周期中易于维护。

参 考 文 献

1 Information technology—Open Systems Interconnection -- Conformance testing methodology and framework -- Part 1: General concepts, ISO/IEC 9646-1,1994

2 Kerbrat A,Jeron T,Groz R. Automated test generation from SDL specifications. In: Proc. of SDL Forum '99, Montreal, Canada, 1999. 135～151

3 Deussen P H,Tobies S. Formal test Purposes and the Validity of Test Cases, FORTE2002,114～129

4 Ural H,Saleh K,Williams A. Test generation based on control and data dependencies within system specifications in SDL, Computer Communications, 2000,23:609～627

5 Robles T,Mañas J A,Huecas G. Specification and Derivation of OSI Conformance Test Suites, FORTE1993,177～187

6 Moy J. OSPF Version 2,STD 54, RFC 2328, April 1998

（上接第22页）

3) 高的可用性和可扩展性支持，如本文第3部分所介绍的。

4) 在消息定义中可以有8种优先级，提供了对 QoS 保证的支持。

5) 缺点是需要有一个安全可靠的运行环境来预防 DoS 的攻击。

总结 本文介绍了一种用在 ForCES[1]框架中的 master/slave 结构的控制协议 FACT[2]协议，它可以提供控制层对数据转发层的控制和配置，并且为网络设备提供服务扩展能力，可延展性以及高可用性的支持，对基于网络处理器构建可扩展服务的路由器，以及其他采用 ForCES 结构的网络设备都具有重要的意义。

参 考 文 献

1 http://www.ietf.org/html.charters/forces-charter.html

2 Audu A, et al. ForwArding and Control ElemenT protocol (FACT),draft-gopal-forces-fact-05. Sep. 2003

3 Khosravi H, et al. Requirements for Separation of IP Control and Forwarding. RFC 3654, Nov. 2003

4 Yang L, et al. ForCES Forwarding Element Model, draft-ietf-forces-model-01. Oct. 2003

5 Yang L, et al. Forwarding and Control Element Separation (ForCES) Framework, draft-ietf-forces-framework-13. work in progress, July 2003

6 Salim J H, et al. Netlink2 as ForCES protocol, draft-jhsrha-forces-netlink2-02. Nov. 2003

7 Wang Weiming, et al. General Router Management Protocol (GRMP) Version 1, draft-wang-forces-grmp-00. Nov. 2003