ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.21, No.7, July 2010, pp.1536−1549
doi: 10.3724/SP.J.1001.2010.03615

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

基于规格说明的若干逻辑覆盖测试准则*

钱忠胜[1,2+], 缪淮扣[1]

[1](上海大学 计算机工程与科学学院,上海 200072)

[2](江西财经大学 信息管理学院,江西 南昌 330013)

Specification-Based Logic Coverage Testing Criteria

QIAN Zhong-Sheng[1,2+], MIAO Huai-Kou[1]

[1](School of Computer Engineering and Science, Shanghai University, Shanghai 200072, China)

[2](School of Information Technology, Jiangxi University of Finance and Economics, Nanchang 330013, China)

+ Corresponding author: E-mail: changesme@163.com, http://www.shu.edu.cn, http://www.jxufe.edu.cn

Qian ZS, Miao HK. Specification-Based logic coverage testing criteria. *Journal of Software*, 2010,21(7): 1536−1549. http://www.jos.org.cn/1000-9825/3615.htm

Abstract: The specification-based testing can be used to test software functions without knowing program code. Decisions are the main form of the pre- and post-conditions in formal specifications. This work analyzes logic coverage testing criteria for specification-based testing. It proposes and analyzes in detail masking logic coverage testing criteria, to solve the problems that the existent determinant logic coverage testing criteria can not solve. A feasible test case generation algorithm based on the masking logic coverage testing criteria is presented. The test cases satisfying the masking logic coverage testing criteria can detect those errors caused by the masking property of conditions. It also analyzes the constraints among conditions, how to decompose and compose a complicated decision, and the relationship among decisions. These can respectively clarify the coupling problem among conditions, the multiple occurrences of a condition in a decision, and the position problem of decisions in a program. Additionally, test criteria including full true decision coverage, full false decision coverage, all sub-decisions coverage, unique condition true coverage and unique condition false coverage are proposed. The test sets satisfying these criteria can detect respectively different types of errors. Finally, the subsumption relation graph among these testing criteria is presented and different applicable scenarios for different testing criteria are suggested.

Key words: test case; testing criterion; decision; condition; specification; subsumption relation

* Supported by the National Natural Science Foundation of China under Grant No.60673115 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant No.2007AA01Z144 (国家高技术研究发展计划(863)); the National Basic Research Program of China under Grant Nos.2007CB310800, 2002CB312001 (国家重点基础研究发展计划(973)); the Research Program of Shanghai Education Committee of China under Grant No.07ZZ06 (上海市教委科研项目); the Shanghai Leading Academic Discipline Project of China under Grant No.J50103 (上海市重点学科建设项目); the Science and Technology Plan Project of the Education Department of Jiangxi Province of China under Grant No.GJJ10120 (江西省教育厅科技计划项目); the School Foundation of Jiangxi University of Finace and Economics of China under Grant No.04722015 (江西财经大学校级课题)

Received 2008-04-03; Revised 2008-11-27, 2009-01-14; Accepted 2009-03-31

摘 要: 基于规格说明的测试可以在不需要了解软件程序代码的情况下对软件进行功能测试.判定是形式规格说明中用于描述前、后置条件的主要形式.分析了基于规格说明的逻辑覆盖测试准则,针对已有的决定性逻辑覆盖测试准则的不足,提出了掩盖性逻辑覆盖测试准则,并对其进行了详细分析.提出了掩盖性逻辑覆盖测试准则的一个可行的测试生成算法.根据该准则生成的测试用例能够发现条件的掩盖性带来的错误.然后,从判定的结构入手,分析了条件之间的约束关系、复杂判定的分解与合成、判定之间的关系.这些分别能够阐明逻辑覆盖中条件间的耦合性问题、同一个条件在判定中的多次出现问题以及判定在程序中的位置问题.继而提出了全真判定覆盖、全假判定覆盖、完全子判定覆盖、唯一条件真覆盖以及唯一条件假覆盖等测试准则.满足这些测试准则的测试用例集能检测出不同类型的错误.最后,给出了这些测试准则之间的包含关系图,并建议了不同测试准则适用的应用场景.

关键词: 测试用例;测试准则;判定;条件;规格说明;包含关系

中图法分类号: TP311 **文献标识码**: A

测试覆盖是根据测试准则来评估的,而测试准则的提出又与测试需求相关联[1].若测试需求描述的是要覆盖“什么”,则测试规格说明阐述了它们“如何”被覆盖,而测试准则是一条或一组规则,这些规则把测试需求施加于测试用例集.也就是说,测试准则用一种完整而清晰的方式描述了测试需求.因此在软件测试中,测试准则的选择非常重要.

测试准则一般分为基于程序的(白盒的、结构化的)和基于规格说明的(黑盒的、功能的).基于程序的测试准则考虑程序的内部结构.基于规格说明的测试准则用于从系统的形式规格说明出发产生测试用例,不需要程序的代码.形式化规格说明精确地描述了“做什么”而不是“怎么做”.因此,测试者可以从中获得重要的信息而无需从繁琐以及不重要的细节中提取.基于规格说明的测试与传统的基于代码的测试相比,它提供了一种更为严格的方法,并且简化了回归测试.此外,基于规格说明的测试还可以从测试数据中得出预期值,以及测试的开发和程序的设计与运行可以同步,这更有利于打破软件工程中的“先编码后测试”的做法,并且有利于对软件的生命周期中所有阶段并行地进行测试活动.

为适应不同的需求,研究人员已经提出了各种各样的测试准则[1−11].其中,逻辑覆盖测试准则主要用于基于形式规格说明的测试中[12],通过分析规格说明中判定(decision)和条件(condition)的真值关系来产生覆盖某些判定和条件的真值取值的测试用例.文中涉及到的常见的逻辑覆盖测试准则的定义(定义 2~定义 4 以及定义 6~定义 9)来自于文献[1]和文献[12],但是为了适应需要,有些地方作了相应的改动.在此基础上,本文提出了相应的逻辑覆盖测试准则(定义 11~定义 18).

这里讨论的判定(谓词表达式)仅使用到常用的 3 种逻辑联结词:∧(与),∨(或),¬(非),当然还有许多其他逻辑联结词(如异或⊕、蕴含→等),但包含其他逻辑联结词的判定可以通过等价公式转换成仅含有这 3 种逻辑联结词的判定.而且,两个语法结构不一样的等价判定的主析取范式或主合取范式一定是一样的.所以,我们一般只考虑这 3 种逻辑联结词.

本文第 1 节界定判定的结构,给出测试用例的形式化定义,并讨论基本的逻辑覆盖测试准则.第 2 节给出决定性逻辑覆盖测试准则,这种准则测试条件的取值改变能够引起判定值的改变,举例说明这种准则不能适用的情况.第 3 节针对决定性逻辑覆盖测试准则的缺陷,提出掩盖性逻辑覆盖测试准则,并对其进行详细而深入的分析.提出掩盖性逻辑覆盖测试准则的一种可行的测试生成算法.还利用配对表分析条件的掩盖性,即条件取值的改变对判定的值没有影响.第 4 节从分析判定的结构入手,研究了条件之间的约束关系、复杂判定的分解与合成、判定之间的关系.并提出全真判定覆盖、全假判定覆盖、完全子判定覆盖、唯一条件真覆盖以及唯一条件假覆盖等测试准则.满足这些测试准则的测试用例集能够检测出不同类型的错误.第 5 节与相关工作进行比较,给出本文提出的测试准则与部分已有的测试准则之间的包含关系图,并建议不同测试准则适用的应用场景.第 6 节总结全文,并给出下一步的研究工作.

1 基本逻辑覆盖测试准则

判定是形式规格说明中用于描述前、后置条件的主要形式,也是程序中分支控制条件的主要表达形式,因而可作为基于规格说明的测试用例研究的主要对象和定义测试充分性的基础[12].判定的主要元素是谓词,谓词中可以有常量、变量、函数以及它们的组合等项.关系表达式也是一个谓词.谓词通过逻辑联结词相连,构成判定.如,$\neg X\wedge Y\vee x<y\vee a>b\wedge Z$(记为$\mathcal{K}$)就是一个判定.不包含任何逻辑联结词的原子谓词或其否定称为条件.判定$\mathcal{K}$中有 5 个条件:$\neg X,Y,x<y,a>b$ 和 Z.注意,x 和 y 都不是条件,而 $x<y$ 才是(因为考察的是逻辑联结词,而非关系算子).判定可用抽象语法树[2]表示.

在软件的形式需求规格说明中,操作的前、后置条件是一组判定的集合,用 P 表示.C 是 P 中所有判定中的条件所构成的集合.对 P 中的每个判定 $p\in P$,C_p 是判定 p 中所有条件构成的集合,即 $C_p=\{c|c\in C$ 且 c 出现在 p 中$\}$.所以,$C=\bigcup_{p\in P}C_p$ [12].此外,TS 表示测试用例集,用 $t,t_1,t_2,\ldots$表示测试用例.本文把测试用例简化为判定中每个条件的真值取值组合.下面给出测试用例的形式化定义.

定义 1(测试用例,test case). 一个测试用例是一个三元组 $t=(Pre,In,Out)$,它是可导致程序一次执行的输入集,其中,Pre 表示输入的前置条件,In 表示输入值,Out 表示期望输出.

一个测试用例集是否检测了某些判定和条件的真值决定了该测试用例集的充分性.利用逻辑覆盖测试准则推导具体测试数据过程的关键是找出满足测试准则的条件真值取值的组合.对于一个测试用例,若其输入的前置条件 Pre 永真,则它的输入值 In 总会被"执行".在下面的讨论中,若不作特别说明,默认前置条件 Pre 为永真并忽略前置条件.另外,若不致引起混淆,也不给出测试用例的期望输出 Out.这样一个测试用例只用其输入值 In 来表示.因此,我们将每个条件真值取值的组合看成一个测试用例.例如,在判定 $A\wedge B$ 中,A=true$\wedge B$=true 就是一个测试用例.对应于实际的规格说明,若用 A 表示"用户的 id 号是否正确",B 表示"用户的口令是否正确",则该测试用例表示用户的 id 号和口令都输入正确,即用户是合法时的情况.

定义 2(判定覆盖,decision coverage,简称 DC). 一组条件的真值组合构成的集合 TS 满足判定覆盖准则,当且仅当对每个判定 $p\in P$,TS 至少包含两个关于 p 中条件的真值组合,其中一个使得 p 为 true,另一个使得 p 为 false.

当被测判定是永真(或永假)时,条件的取值任意,因而实际上不存在测试用例的选择问题.如果在这种退化了的判定情况下仍然有效,则认为满足判定覆盖,虽然条件的真值组合不能使该判定的取值为 false(或 true).判定覆盖不能决定其中某个条件的取值.为了揭露判定中一个条件的错误,判定中的其他条件必须满足一定的约束.

定义 3(条件覆盖,condition coverage,简称 CC). 一组条件的真值组合构成的集合 TS 满足条件覆盖准则,当且仅当对每个判定 $p\in P$,C_p 中的条件 $c\in C_p$,TS 至少包含两个关于 p 中条件的真值组合,其中一个使得 c 为 true,另一个使得 c 为 false.

条件覆盖并不一定包含(subsume)(关于测试准则之间的"包含"关系这一概念将在本文的第 5 节给出)判定覆盖,判定覆盖也不一定包含条件覆盖.它们都是很弱的测试准则,对于安全性软件的测试根本不够.即使所有的判定和条件的真值取值都已检查过,也无法保证所有可能的条件真值的组合都能够被检查.为了检查判定中所有条件的真值组合,于是出现了组合覆盖准则.

定义 4(组合覆盖,combinational coverage,简称 CoC). 一组条件的真值组合构成的集合 TS 满足组合覆盖准则,当且仅当对每个判定 $p\in P$,TS 包含 C_p 中所有可能的条件真值的组合.

假设被测判定中不同条件的个数为 n,则组合覆盖测试用例集中的测试用例共有 2^n 个.显然,组合覆盖准则包含判定覆盖准则和条件覆盖准则.

2 决定性逻辑覆盖测试准则

组合覆盖测试准则过于严格,它只能用在条件个数不多的情况下.为了克服组合覆盖准则的缺陷,可以从条

件的真值对判定真值的作用出发来构造有效的测试准则.术语“决定”用来表达条件的真值和判定真值之间的关系.

定义 5(决定,determination). 对于判定 p 中的一个条件 c_i,如果 p 中其余的条件 $c_j \in p, j \neq i$ 的真值使得改变 c_i 的真值会引起 p 的真值发生变化,那么称 c_i 决定判定 p,其中,c_i 为主要条件(major condition),c_j 为次要条件(minor condition).

利用条件和判定之间的决定关系(我们称为逻辑覆盖的决定性),可以定义如下活动条件覆盖准则:

定义 6(活动条件覆盖,active condition coverage,简称 ACC). 一组条件的真值组合构成的集合 TS 满足活动条件覆盖准则,当且仅当对每个判定 $p \in P$ 和每个主要条件 $c_i \in C_p$,TS 中至少含有两个 C_p 中条件真值组合,使得 c_i 一次为 true,一次为 false.

活动条件覆盖意味着判定值的改变随着某一条件值的改变而改变.该准则要求测试各种条件值的组合(但不是所有的,这与组合覆盖不同),测试用例的个数仍然可以保持线性增长.

对于具有 n 个相互独立条件的判定,测试用例集中最多包含(n+1)个不同的测试用例就可以满足活动条件覆盖准则[1].在应用该准则时存在一些问题,即在选取满足活动条件覆盖准则的测试用例集,当主要条件的值为 true 和 false 时,次要条件是否应该取相同的值?对于这一问题的不同回答产生了 3 个不同的逻辑覆盖测试准则(定义 7~定义 9).

定义 7(一般活动条件覆盖,general active condition coverage,简称 GACC). 一组条件的真值组合构成的集合 TS 满足一般活动条件覆盖准则,当且仅当对每个判定 $p \in P$ 和每个主要条件 $c_i \in C_p$,TS 中至少含有两个 C_p 中条件的真值组合,使得 c_i 一次为 true,一次为 false,并且对这两个条件真值组合中次要条件的真值取值不作要求.

由于一般活动条件覆盖准则并不一定包含判定覆盖准则,所以给出下面相关活动条件覆盖的概念.

定义 8(相关活动条件覆盖,correlated active condition coverage,简称 CACC). 由一组条件的真值组合构成的集合 TS 满足相关活动条件覆盖准则,当且仅当对每个判定 $p \in P$ 和每个主要条件 $c_i \in C_p$,TS 中至少含有两个 C_p 中条件的真值组合,使得 c_i 一次为 true,一次为 false,并且这两个条件真值组合必须分别使得 p 取不同的真值.

定义 9(受限活动条件覆盖,restricted active condition coverage,简称 RACC). 由一组条件的真值组合构成的集合 TS 满足受限活动条件覆盖准则,当且仅当对每个判定 $p \in P$ 和每个主要条件 $c_i \in C_p$,TS 中至少含有两个 C_p 中条件的真值组合,使得 c_i 一次为 true,一次为 false,并且这两个条件真值组合中次要条件必须取相同的值.

受限活动条件覆盖等同于 MC/DC(修正的条件/判定覆盖)[2,7],当主要条件的取值发生变化时,都规定次要条件应该取相同的值.对于一些实际问题,定义 9 并不是很精确.比如,如何处理当改变主要条件的值而无法保持次要条件的值不变的情况.假定该主要条件不满足,则受限活动条件覆盖可能不是处理该问题的最好方法.但无论如何,在测试中必须检查这样的条件.

根据定义 7~定义 9,显然,满足受限活动条件覆盖准则的测试用例集一定满足相关活动条件覆盖准则,而满足相关活动条件覆盖准则的测试用例集一定满足一般活动条件覆盖准则.

在活动条件覆盖的相关定义(定义 6~定义 9)中,当次要条件取一定的真值时,主要条件都是“活动的”,即判定的真值改变随主要条件真值的不同而改变.但在很多情况下,希望一个条件真值的改变对判定的真值没有影响,即,改变某个条件的真值时保持判定的真值不变.现在来看下面的两个例子,它们都摘自于文献[3],但为方便描述,有些地方作了相应的改动与补充.

例 1:铁路交叉点.在铁路计算机控制系统中,通过交叉点的交换开关可以控制火车从一个方向驶向另一个方向.假设有两个轨道,一个是主要的,一个是预留的,条件决定轨道的状态(是否被占用),判定决定火车路线的改变(从主要轨道驶向预留轨道或者相反).现在考虑两种故障类型:一个是无操作,即该发生的未发生;一个是误激励,即不该发生的却发生了.第 1 种情况是当主要轨道被占用时(条件改变),需要转换到预留轨道(判定改变).若此时判定的值保持不变,则意味着系统无操作,可能导致其崩溃.第 2 种情况是当预留轨道被占用时(条件改变),需要保持主要轨道的路线(判定保持).若此时判定的值发生了变化,则意味着系统发生误激励,同样可能导致系统崩溃.

例 2:核反应堆防护系统.考虑一个判定和一个条件,判定用于触发一个核电厂的核反应堆防护系统(如反应堆的关闭),条件用于描述触发的某个标准(如对于某个规定的级别,压力过大).需要测试当条件改变时,判定也要改变的情况,因为此时出现故障,表明系统在紧急条件下无操作,会导致核反应堆事故.然而保持判定的值也是非常重要的,此时的故障意味着在正常操作期间系统的误激励,这可能会导致非强制性的反应堆关闭、物理设备的损坏以及电流供应不足.核反应堆防护系统典型的体系结构考虑了这种特殊情况,即“3 选 2”逻辑投票制.使用 3 个等价的信道降低了系统不正确操作的概率.然而,如果仅仅是需要考虑这个因素,用“3 选 1”逻辑更可靠.用“3 选 2”逻辑的目的是对系统的误激励提供保护,因为在这种情况下,来自一个信道的错误信号不会触发系统的行为.

这些例子表明,仅测试条件取值的变化引起判定值的变化是不够的,从安全性的角度考虑更是如此.因此,在基于规格说明的测试中,除了测试条件取值变化对判定的值“改变”的情况,还有必要测试条件取值变化对判定的值“不改变”的情况.

3 掩盖性逻辑覆盖测试准则

当使用逻辑联结词∧和∨时,一般都要计算判定中的条件的真值.某些编程语言也提供“短路(short circuit)”的控制形式.一旦结果的值被确定,短路控制逻辑就不需要计算判定中其他条件的值.在 Ada 语言中,and then 和 or else 是两个短路控制形式,除了总是先计算左边的条件之外,它们分别产生与逻辑联结词∧和∨相同的结果,右边的条件只有当其会对整个判定的结果产生变化时才会进行计算.C,C++中的算子&&和||分别与 Ada 语言中的 and then 和 or else 表达的意思相同.一旦∧(或∨)条件的取值为 false(或 true),判定的结果将保持 false(或 true),不需要进行进一步的计算,也不会再改变最终结果.

3.1 相关定义

条件真值取值的变化若不改变判定的真值取值,则说明该条件的真值取值对判定的作用被其他某些条件的真值取值“掩盖”了.例如,对于判定 $X\vee Y$,当条件 X=true 时,条件 Y 被条件 X 所掩盖,因为无论条件 Y 的真值取值如何,判定 $X\vee Y$ 的取值始终为 true.相反地,当条件 Y=true 时,条件 X 被条件 Y 所掩盖.掩盖是指对条件的特殊的输入会隐蔽其他条件输入的作用.条件的掩盖性引起错误的测试可用于测试短路控制逻辑.

定义 10(掩盖,mask). 对于判定 p 中的一个条件 c_i,如果 p 中其他的某个条件 $c_j\in p,j\neq i$ 的真值,使得改变 c_i 的真值不会引起 p 的真值发生变化,那么称 c_j 掩盖 c_i(或称 c_i 被 c_j 所掩盖).其中,c_i 为显式条件,c_j 为隐式条件.

利用条件之间的掩盖关系(我们称其为逻辑覆盖的掩盖性),可以定义如下掩盖条件覆盖准则.

定义 11(掩盖条件覆盖,mask condition coverage,简称 MCC). 由一组条件的真值组合构成的集合 TS 满足掩盖条件覆盖准则,当且仅当对每个判定 $p\in P$ 和每个显式条件 $c_i\in C_p$,TS 中至少含有两个 C_p 中的条件真值组合,使得 c_i 一次为 true,一次为 false.

例如,对于判定 $X\vee Y$,测试用例集 $TS\equiv\{X$=true$\wedge Y$=true,X=false$\wedge Y$=true,X=true$\wedge Y$=false$\}$需要 3 个测试用例满足掩盖条件覆盖准则.其中,X=true$\wedge Y$=true 和 X=false$\wedge Y$=true 分别使得 X=true 和 X=false,而 X=true$\wedge Y$=true 和 X=true$\wedge Y$=false 分别使得 Y=true 和 Y=false.并且该测试用例集使得判定 $X\vee Y$ 的值始终保持 true,而无论显式条件的取值如何变化.但无论如何,条件 X 或 Y 的取值都不能使判定 $X\vee Y$ 的值始终保持 false.

对于判定 $X\wedge Z\vee Y\wedge Z$,通过分析可知,条件 X 或 Y 的取值可使判定 $X\wedge Z\vee Y\wedge Z$ 的值始终保持 true 或者 false,而条件 Z 的取值只能使该判定的值始终保持 false,不能保持 true.一般地,对于一个特定的判定,不是它的所有的显式条件都能使其值同时保持 true 和 false.因此在下面的讨论中,我们不再分别分析显式条件让判定保持 true 和 false 这两种情况,而只要保持其中的一种取值就可以了.

掩盖条件覆盖准则和活动条件覆盖准则都用于考察一个条件真值的变化是否会引起判定取值的变化.在应用掩盖条件覆盖准则时,与活动条件覆盖准则一样,也存在同样的问题,即在选取满足掩盖条件覆盖准则的测试用例集,当显式条件的值为 true 和 false 时,隐式条件是否应该取相同的值?对于这一问题的不同回答产生了两个不同的逻辑覆盖测试准则(定义 12 和定义 13).

定义 12(一般掩盖条件覆盖,general mask condition coverage,简称 GMCC). 由一组条件的真值组合构成的集合 TS 满足一般掩盖条件覆盖准则,当且仅当对每个判定 $p\in P$ 和每个显式条件 $c_i\in C_p$,TS 中至少含有两个 C_p 中的条件的真值组合,使得 c_i 一次为 true,一次为 false,并且对这两个条件真值组合中隐式条件的真值取值不作要求.

定义 13(受限掩盖条件覆盖,restricted mask condition coverage,简称 RMCC). 由一组条件的真值组合构成的集合 TS 满足受限掩盖条件覆盖准则,当且仅当对每个判定 $p\in P$ 和每个显式条件 $c_i\in C_p$,TS 中至少含有两个 C_p 中的条件的真值组合,使得 c_i 一次为 true,一次为 false,并且这两个条件真值组合中隐式条件必须取相同的真值.

注意,根据定义 10,由于显式条件真值的改变不会引起判定的真值发生变化,即显式条件和判定之间不存在决定关系,因此没有"相关掩盖条件覆盖"的概念.

3.2 测试生成算法

对于一个判定 $p\in P$,在生成满足受限掩盖条件覆盖准则的条件真值组合时,如何确定隐式条件的取值呢?令 $X\in C_p$ 为判定 p 的显式条件,当分别给定 X=true 和 X=false 时,判定 p 的值应该相同(即要么都为 true,要么都为 false),因为显式条件 X 的取值被其他隐式条件的取值所掩盖.若将这两次得到的 p 值进行⊕(异或)操作,得到的结果必为假.当用 $p(X)$表示显式条件 X 取值为真时,判定 p 的值;当 $p(\neg X)$表示显式条件 X 取值为假时,判定 p 的值;$X(p)$表示 $p(X)$和 $p(\neg X)$的异或,即 $X(p)\equiv p(X)\oplus p(\neg X)$.这样,通过计算 $X(p)$并使其取值为假,就可以确定隐式条件的取值.我们把这种方法称为"掩盖异或求值法".

例如,设 $p\equiv X\wedge Y$,

(1) 当 X 为显式条件时,

$$\begin{aligned}X(p)&\equiv p(X)\oplus p(\neg X)\\&\equiv(\text{true}\wedge Y)\oplus(\text{false}\wedge Y)\\&\equiv Y\oplus\text{false}\\&\equiv Y.\end{aligned}$$

要使得 $X(p)$为假,隐式条件 Y 的取值必须为假.也就是说,在该判定 p 中,当 X 为显式条件时,隐式条件 Y 应取假,即有两组真值组合:(X=true,Y=false),(X=false,Y=false)测试显式条件 X.

(2) 当 Y 为显式条件时,

$$\begin{aligned}Y(p)&\equiv p(Y)\oplus p(\neg Y)\\&\equiv(X\wedge\text{true})\oplus(X\wedge\text{false})\\&\equiv X\oplus\text{false}\\&\equiv X.\end{aligned}$$

要使得 $Y(p)$为假,隐式条件 X 的取值必须为假.也就是说,在该判定 p 中,当 Y 为显式条件时,隐式条件 X 应取假,即有两组真值组合:(X=false,Y=true),(X=false,Y=false)测试显式条件 Y.

由定义 12 和定义 13 可知,满足受限掩盖条件覆盖准则的测试用例集一定满足一般掩盖条件覆盖准则.下面给出掩盖性逻辑覆盖测试准则的一个可行的测试生成算法,见算法 1.

算法 1. 掩盖性逻辑覆盖测试准则的测试生成.

输入:判定Θ;

输出:满足掩盖条件覆盖准则的一个抽象测试用例集Γ.

```
begin
    Γ=∅;                                   //初始化抽象测试用例集为空
    while (Θ中还有条件未充当过显式条件)
        explicit←selectExplicitCondition(Θ);   //在Θ中选定一个未充当过显式条件的条件作为
                                               显式条件
```

```
        computeValuesOfImplicitConditions(Θ);      //用“掩盖异或求值法”计算隐式条件的值
        setTrue(explicit);                         //显式条件的取值为真
        tc←combineConditions(Θ);                   //将条件的取值组合构成一个抽象测试用例
        if (tc 不在Γ中)                            //去掉相同的条件组合
            Γ=Γ∪{tc};
        endif;
        setFalse(explicit);                        //显式条件的取值为假
        tc←combineConditions(Θ);                   //将条件的取值组合构成一个抽象测试用例
        if (tc 不在Γ中)                            //去掉相同的条件组合
            Γ=Γ∪{tc};
        endif;
    endwhile;                                      //Θ中的条件都充当过显式条件
    output Γ;                                      //输出抽象测试用例集
end.
```

3.3 实例分析

考察判定$\mathcal{K}_1\equiv X\vee Y$,其真值表见表 1.从表中可以看出,测试用例(false,false)应该剔除,因为它使得判定的值为 false,而又没有第 2 个测试用例使得判定的值也为 false.现考察将条件 X 作为显式条件的情况,若保证 Y 的取值不变,则测试用例(true,true)和(false,true)使得条件 X 的取值一次为真,一次为假,但不改变判定的值,满足 X 的被掩盖性.同理,保证 X 的取值不变,得到显式条件 Y 的测试用例(true,true)和(true,false),因此测试用例集{(true,true),(true,false),(false,true)}满足掩盖条件覆盖准则.

Table 1 Truth table of $X\vee Y$

表 1 $X\vee Y$ 的真值表

X	Y	Result
True	True	True
True	False	True
False	True	True
False	False	False

利用配对表(pairs table)[2]可以更清晰、准确地表达测试用例之间的成对关系.表 1 的真值表可以扩展为表 2 所示的配对表形式.在表 2 中,第 1 列是测试用例名,标签为(X,Y)的列表示条件 X 和 Y 的真值组合,标签为 X(或 Y)的列表示条件 X(或 Y)的被掩盖性.例如,测试用例 t_1:(true,true)(或者测试用例 t_2:(true,false))可以和测试用例 t_3:(false,true)配对使用考察条件 X 的被掩盖性.测试用例 t_1:(true,true)(或者测试用例 t_3:(false,true))可以和测试用例 t_2:(true,false)配对使用考察条件 Y 的被掩盖性.测试用例 t_4:(false,false)对考察条件 X 和 Y 的被掩盖性都不起作用.

Table 2 Pairs table of $X\vee Y$

表 2 $X\vee Y$ 的配对表

Test case	(X,Y)	Expected output	X	Y
t_1	(true,true)	True	t_3	t_2
t_2	(true,false)	True	t_3	t_1, t_3
t_3	(false,true)	True	t_1, t_2	t_2
t_4	(false,false)	False		

使用配对表设计测试用例集需要为每个条件选择成对的测试用例,选择的成对测试用例有些是重叠的.如在表 2 中,当条件 X 选择测试用例 t_1 和 t_3,而条件 Y 选择测试用例 t_1 和 t_2 时,则 t_1 就重叠了.当配对表的某一行在所有条件对应的列都不为空时,该行就定义了一个满足掩盖条件覆盖准则的测试用例集.例如,对应于 t_1 行,它在

所有条件对应的列都不为空,得到测试用例集$\{t_1,t_2,t_3\}$,该测试用例集就满足定义在判定$X\vee Y$上的掩盖条件覆盖准则.注意,对应于 t_2(或 t_3)行,可以得到两个测试用例集$\{t_1,t_2,t_3\}$和$\{t_2,t_3\}$,它们都满足掩盖条件覆盖准则,但$\{t_1,t_2,t_3\}$还满足受限掩盖条件覆盖准则,而$\{t_2,t_3\}$不满足.

现考察稍微复杂的有 3 个条件的判定$\mathcal{K}_2\equiv X\wedge Y\vee Z$,其配对表见表 3.在表 3 中,$t_1,t_2,t_3$可分别和 t_5或 t_7配对,t_4可分别和 t_6或 t_8配对来考察条件 X的被掩盖性.即,要测试 X,测试用例集$\{t_1,t_5\},\{t_1,t_7\},\{t_2,t_5\},\{t_2,t_7\},\{t_3,t_5\},\{t_3,t_7\},\{t_4,t_6\},\{t_4,t_8\}$都满足要求.同理,要测试 Y,测试用例集$\{t_1,t_3\},\{t_1,t_7\},\{t_2,t_3\},\{t_2,t_7\},\{t_3,t_5\},\{t_4,t_6\},\{t_5,t_7\},\{t_6,t_8\}$都满足要求.而要测试 Z,测试用例集$\{t_1,t_2\},\{t_2,t_3\},\{t_2,t_5\},\{t_2,t_7\}$都满足要求.注意到,测试用例集$\{t_2,t_7\}$同时满足测试 X,Y和Z这 3 个条件的被掩盖性,它是满足此特性的最小测试用例集.但$\{t_2,t_7\}$只满足一般掩盖条件覆盖准则,不满足受限掩盖条件覆盖准则.

Table 3 Pairs table of $X\wedge Y\vee Z$

表 3 $X\wedge Y\vee Z$ 的配对表

Test case	(X,Y,Z)	Expected output	X	Y	Z
t_1	(true,true,true)	True	t_5, t_7	t_3, t_7	t_2
t_2	(true,true,false)	True	t_5, t_7	t_3, t_7	t_1, t_3, t_5, t_7
t_3	(true,false,true)	True	t_5, t_7	t_1, t_2, t_5	t_2
t_4	(true,false,false)	False	t_6, t_8	t_6	
t_5	(false,true,true)	True	t_1, t_2, t_3	t_3, t_7	t_2
t_6	(false,true,false)	False	t_4	t_4, t_8	
t_7	(false,false,true)	True	t_1, t_2, t_3	t_1, t_2, t_5	t_2
t_8	(false,false,false)	False	t_4	t_6	

在表 3 中,对于条件 X的所有测试用例集,$\{t_1,t_5\},\{t_3,t_7\},\{t_4,t_8\}$对应的$(Y,Z)$真值取值相同;对于条件 Y的所有测试用例集,$\{t_1,t_3\},\{t_5,t_7\},\{t_6,t_8\}$对应的$(X,Z)$真值取值相同;对于 Z 的所有测试用例集,$\{t_1,t_2\}$对应的(X,Y)真值取值相同.因此,可以取测试用例集$\{t_1,t_2,t_3,t_5\}$,它满足受限掩盖条件覆盖,当然也满足一般掩盖条件覆盖,并且它是最精简的.这里定义“最精简测试用例集”为能满足某个给定的测试准则而又无多余测试用例的测试集,即它的任何真子集不满足该测试准则.注意,对于这个例子,满足受限掩盖条件覆盖的最精简测试用例集不只 1 个,还有$\{t_1,t_2,t_3,t_7\}$和$\{t_1,t_2,t_5,t_7\}$.另外,假设 TS_1和 TS_2都是满足某个测试准则的最精简测试用例集,但 TS_1和 TS_2中测试用例的个数不一定相等.

3.4 掩盖条件覆盖准则的特点

配对表具有一定的局限性:(1) 配对表在表达条件较多的判定时是难以处理的,且对于具有 n 个不同条件的判定一般仅需要$(n+1)$个测试用例,而表中却有 2^n 行;(2) 一张配对表一次只能用于表达一个判定;(3) 配对表没有将基于规格说明的测试用例的输入和输出与源代码的结构联系起来.然而,配对表对于表达条件的掩盖性是一种简便而有效的方法.通过分析,掩盖条件覆盖准则具有下列特点:

- 测试用例个数的线性增长.当判定中条件个数增加时,满足掩盖条件覆盖准则所需的测试用例个数呈线性增长.对每个条件来说,掩盖条件覆盖准则最多需要 4 个测试用例,两个使判定保持 true,两个使判定保持 false,但这样选择测试用例过于严格.我们只需考虑一种情况,即,使判定或者保持 true,或者保持 false.因此,对于有 n 个相互独立条件的判定,最多需要 $2n$ 个测试用例就可以满足掩盖条件覆盖准则.此时,每个条件对应的测试用例都不同,但往往没有必要选择这么多,因为有些测试用例可以用于测试不同的条件.通常,测试用例集中最多包含$(n+1)$个不同的测试用例即可;
- 条件对判定的非作用性.掩盖条件覆盖准则考察的是条件的被掩盖性.即,测试一个条件取值的改变对判定没有影响,这在许多情况下非常有用,特别是在测试安全性要求较高的系统时更是如此(见第 2 节最后两个例子).需注意,当改变一个条件真值取值时,有些判定不会保持 true 和 false 中的某个值.例如,$X\wedge Y$不会保持 true,而$\neg X$不会保持 true 和 false.

4 判定的结构分析

有关“逻辑覆盖”的文献中只考虑了一个判定中每个条件(或条件的组合)的覆盖关系,而很少考虑一个判定中条件间的约束关系对判定的作用(例如,对于判定 $X\wedge Y$,假设当 X=true 时,Y 不能为 true,其约束为$\neg X\vee\neg Y$,即 Y 的取值受 X 取值的约束),未考虑将一个判定分成几个部分,分析各个部分间的合成关系(例如,将一个判定划成它的合取范式,考虑各项之间的关联),也未考虑判定之间的关联.为此,我们进一步分析判定的结构,考虑如下 3 个方面的问题:(1) 一个条件的取值约束另一个条件的取值,即条件与条件的关联;(2) 判定的分解与合成,即多个条件的组合对其所在判定的影响;(3) 判定与判定的关联.

问题(1)阐明了逻辑覆盖中的一个问题:如何考虑条件之间的耦合性(coupling),即一个条件的取值不能独立地改变,而会受到其他条件的约束.我们采用将“耦合性”关系加入到判定中的方法来解决.

问题(2)阐明了逻辑覆盖中的另一个问题:如何理解同一个条件在判定中的多次出现.例如,对于判定 $\mathcal{K}_3\equiv(X\vee Y)\wedge(X\vee U\vee V)$,一种观点认为包含 4 个条件($X,Y,U,V$),另一种观点认为包含 5 个条件(第 1 个 X,Y,U,V 和第 2 个 X).我们采用前面的观点,因为后面的观点使条件的个数增加,分析更为复杂.

问题(3)分析了判定在程序中的位置问题.此时,一个判定可以认为是另一个判定及其否定的前置条件.

4.1 条件之间的约束关系

判定的条件之间存在两种约束类型,即弱约束和强约束.对于判定 $p\in P$,p 中存在弱约束,当且仅当 p 中存在条件 $c_i,c_j\in C_p$,c_i(或 c_j)的真值取值部分地限制了 c_j(或 c_i)的真值取值;而 p 中存在强约束,当且仅当 p 中存在条件 c_i, $c_j\in C_p$,c_i(或 c_j)的真值取值变化必定引起 c_j(或 c_i)的真值取值变化.若一个判定的条件之间存在多个约束,那么这些约束就构成该判定的一个约束集,且多个约束之间是“与”的关系.

在判定 $\mathcal{K}_2\equiv X\wedge Y\vee Z$ 中,若条件 X 和 Y 之间存在这样一个约束关系:当 X=true 时,Y 不能为 false(即约束为 $\neg X\vee Y$)(比如 X 代表 $score\geqslant 85$,Y 代表 $score\geqslant 60$),则在表 3 中,测试用例 t_3 和 t_4 的输入不满足约束$\neg X\vee Y$,而其他测试用例的输入则满足.因此,满足受限掩盖条件覆盖的最精简测试用例集只有 1 个,即$\{t_1,t_2,t_5,t_7\}$.但若 3 个条件是互斥的,即它们的取值不能同时为 true(即约束为$\neg X\vee\neg Y\vee\neg Z$)时,则测试用例 t_1 的输入不满足约束$\neg X\vee\neg Y\vee\neg Z$,此时也就根本不存在满足受限掩盖条件覆盖的测试用例集.也就是说,条件之间的约束可能使得某些测试用例的产生根本不满足指定的条件,这就有可能不存在测试用例集满足某个规定的测试准则的情况,表明该测试准则的测试覆盖率不可能达到 100%.例如,对应于具体的程序代码,有的路径根本不可达.这两种约束都属于弱约束类型.但若规定 X 的取值必须和 Y 的取值相反,即约束为$(\neg X\wedge Y)\vee(X\wedge\neg Y)$,则它属于强约束类型,在表 3 中,测试用例 t_1,t_2,t_7 和 t_8 的输入不满足约束$(\neg X\wedge Y)\vee(X\wedge\neg Y)$.特别地,若 X 和 Y 之间存在强约束,则 $X=Y$ 或 $X=\neg Y$.如果一个判定的约束不被该判定的任何一个测试用例满足,则它对于该判定是不可行的.

注意,约束和前置条件是有区别的,因为约束是针对于判定的每一个测试用例,而前置条件是针对于某个具体的测试用例.对于某个具体的测试用例来说,只有当它的前置条件为 true 时,该测试用例的输入才是有意义的;但当测试用例不满足判定的约束时,还可以认为它的期望输出为 false.

实际上,可把约束加入到判定来对判定进行刻画(进行“与”操作).例如,把$\neg X\vee\neg Y\vee\neg Z$ 加入$\mathcal{K}_2$,使得$\mathcal{K}_2$ 变为 $\mathcal{K}_2'\equiv(\neg X\vee\neg Y\vee\neg Z)\wedge(X\wedge Y\vee Z)$,约束$\neg X\vee\neg Y\vee\neg Z$ 就看成是新判定 $\mathcal{K}_2'$ 的子判定(此时,$X\wedge Y\vee Z$ 也是 $\mathcal{K}_2'$ 的子判定),然后考察该新判定的测试用例,但需把所有在原判定中不满足约束的测试用例的期望输出改为 false.因此,我们得出下面的结论:

判定及其约束集中,每个约束的“与”构成一个新判定,这些约束构成新判定的子判定.新判定的每个测试用例都是满足所有约束的测试用例.这些测试用例的产生只需将在原判定中不满足约束的测试用例的期望输出改为 false,其他测试用例保持不变即可.

约束也可以用条件真值的组合来表示.例如,对于判定$\mathcal{K}_2\equiv X\wedge Y\vee Z$,在表 3 中只有测试用例 t_3 和 t_4 满足约束(true,false,*),它表示条件 X 取值必须为 true,条件 Y 取值必须为 false,而条件 Z 的取值任意.在将这类约束加入到判定中时,首先要将其转化为判定的形式.例如,约束(true,false,*)的判定形式为 $X\wedge\neg Y$.

4.2 判定的分解与合成

上面的结论把约束看成子判定,实际上考察的是判定的合成操作.同样,判定一般也可以分解成许多子判定,然后重新组合以简化测试.下面来考察判定的分解.

在进行分解之前,首先将判定转换成 $p_1\wedge p_2\wedge\ldots\wedge p_m$ 的形式,它是不可归约的合取范式(判定的合取范式称为是不可归约的,假定在不改变判定意思(或真值)的情况下,它的每个条件(及子判定)都是不能删除的),第 i 项表示为 $p_i=c_{i.1}\vee c_{i.2}\vee\ldots\vee c_{i.k}$ 的形式,其中,$c_{i.j}$ 表示第 i 项的第 j 个条件.判定 $\mathcal{K}_3\equiv(X\vee Y)\wedge(X\vee U\vee V)$ 就是一个不可归约的合取范式,其中,$p_1=c_{1.1}\vee c_{1.2}$,$c_{1.1}=X$,$c_{1.2}=Y$;$p_2=c_{2.1}\vee c_{2.2}\vee c_{2.3}$,$c_{2.1}=X$,$c_{2.2}=U$,$c_{2.3}=V$.若 c 同时出现在子判定 p_i 和 p_j 中,则 c 称为 p_i 和 p_j 的公共条件.

在逻辑覆盖中,当两个等价判定的结构形式不一样时,可能产生不同的测试用例.我们的策略是测试一个规格说明的任何一个实现,根据该策略选择的测试用例与规格说明的格式没有关系(注意,假定规格说明写成了判定的形式,可以认为是判定的集合).因此,虽然规格说明有各种各样的形式,但该策略首先将判定转化为不可归约的合取范式,然后利用该范式生成测试用例.这样,对于前面的具有 X,Y,U 和 V 这 4 个条件的判定 $\mathcal{K}_3\equiv(X\vee Y)\wedge(X\vee U\vee V)$,其本身就是不可归约的合取范式.该范式有两个项:$p_1=X\vee Y$,$p_2=X\vee U\vee V$.

我们将判定 $\mathcal{K}_3$ 分解为两个子判定:$p_1=X\vee Y$ 和 $p_2=X\vee U\vee V$.每个子判定的条件个数都比较少,对每个子判定可以利用已有的测试准则检查覆盖度.就掩盖条件覆盖来说,对于非公共条件 Y,U 和 V,它们在子判定中的掩盖性就是在原判定中的掩盖性.对于公共条件 X 还不能确定,需要把子判定的测试用例进行合成之后才能解决,可采用“自然组合”的方法来合成.

自然组合是在公共条件取值相等的基础上进行的合成操作,去掉其中的公共条件,当两个子判定中没有公共条件时,自然组合就转化为笛卡尔积(两两合成)了.例如,假设得到 p_1 的测试用例集 $TS_1=\{t_1,t_2\}$,其中:$t_1=(X=\text{true},Y=\text{false})$,$t_2=(X=\text{false},Y=\text{true})$;$p_2$ 的测试用例集 $TS_2=\{t_3,t_4,t_5\}$,其中:$t_3=(X=\text{true},U=\text{false},V=\text{false})$,$t_4=(X=\text{false},U=\text{true},V=\text{true})$,$t_5=(X=\text{false},U=\text{true},V=\text{false})$,则 t_1 可以和 t_3 自然组合,t_2 可以分别和 t_4,t_5 自然组合,得到判定 $\mathcal{K}_3\equiv p_1\wedge p_2$ 的测试用例集为{(X=true,Y=false,U=false,V=false),(X=false,Y=true,U=true,V=true),(X=false,Y=true,U=true,V=false)}.

此外,判定 $\mathcal{K}_3$ 为真,当且仅当它的所有子判定为真,而对于每个子判定,若为真,只要其中一个条件为真即可.判定 $\mathcal{K}_3$ 为假,只要其中一个子判定为假即可,而对于每个子判定,若为假,当且仅当它的所有条件为假.这对于考察判定取值全为真和全为假的情况非常方便.因此,下面定义全真判定覆盖和全假判定覆盖.

定义 14(**全真判定覆盖**,**full true decision coverage**,**简称 FTDC**). 由一组条件的真值组合构成的集合 TS 满足全真判定覆盖准则,当且仅当对每个判定 $p\in P$,TS 中至少包含 C_p 中条件的真值组合使得 p 得到所有可能为 true 的取值.

定义 15(**全假判定覆盖**,**full false decision coverage**,**简称 FFDC**). 由一组条件的真值组合构成的集合 TS 满足全假判定覆盖准则,当且仅当对每个判定 $p\in P$,TS 中至少包含 C_p 中条件的真值组合,使得 p 得到所有可能为 false 的取值.

满足全真判定覆盖(或全假判定覆盖)准则的测试用例集可以检查出判定的取值应该为 true(或 false)而结果不为 true(或 false)所带来的错误.另外,我们根据判定的分解与合成中子判定的概念,提出下面的完全子判定覆盖.

定义 16(**完全子判定覆盖**,**all sub-decisions coverage**,**简称 ASDC**). 由一组条件的真值组合构成的集合 TS 满足完全子判定覆盖准则,当且仅当对每个判定 $p\in P$ 和它的每个子判定 p_i,TS 中至少含有两个子判定的真值组合,分别使得 p_i 一次为 true,一次为 false,而对这两个子判定真值组合中其他子判定的真值取值不作要求.

注意,一个判定 p 也是其自身的一个子判定.例如,判定 $\mathcal{K}_2\equiv X\wedge Y\vee Z$ 的子判定有 5 个:$X,Y,Z,X\wedge Y$ 和 $X\wedge Y\vee Z$.$\mathcal{K}_2$ 的完全子判定覆盖的测试用例集可以是 $TS=\{t_1,t_2\}$,见表 4.

Table 4 Test cases for $X\wedge Y\vee Z$ in term of the all sub-decisions coverage

表 4 $X\wedge Y\vee Z$ 的完全子判定覆盖的测试用例

Test case	(X,Y,Z)	Expected output
t_1	(true,true,true)	True
t_2	(false,false,false)	False

满足完全子判定覆盖准则的测试用例集可以检查出子判定的取值可能带来的错误.

完全子判定覆盖和活动条件覆盖以及掩盖条件覆盖之间并没有什么直接的关系,因为完全子判定覆盖中的条件和判定之间没有联系,即,它不讨论一个条件真值的变化是否会引起判定取值的变化.

进一步地,在逻辑覆盖测试准则中,还可以考察一个条件为真(或假),其他条件为假(或真)时的情况,称为唯一条件真(或假)覆盖准则.

定义 17(唯一条件真覆盖,unique condition true coverage,简称 UCTC). 由一组条件的真值组合构成的集合 TS 满足唯一条件真覆盖准则,当且仅当对每个判定 $p\in P$ 和它的每个条件 $c\in C_p$,TS 中至少存在一个条件真值组合,使得 c 为 true,其他条件为 false.

定义 18(唯一条件假覆盖,unique condition false coverage,简称 UCFC). 由一组条件的真值组合构成的集合 TS 满足唯一条件假覆盖准则,当且仅当对每个判定 $p\in P$ 和它的每个条件 $c\in C_p$,TS 中至少存在一个条件真值组合,使得 c 为 false,其他条件为 true.

满足唯一条件真覆盖(或唯一条件假覆盖)准则的测试用例集考察的是每个测试用例只允许一个条件为真(或假)对判定结果所带来的影响.

4.3 判定之间的关联

下面讨论最后一个问题:一个判定与其他判定之间的关联.这一问题涉及到判定在程序中的位置问题.考虑下面的程序代码段中判定 p_1,p_2 和 p_3 之间的关系:

```
IF p1
    segment 1
    IF p2 segment 2 ELSE segment 3 ENDIF
ELSE
    segment 4
    IF p3 segement 5 ENDIF
ENDIF
```

在该程序段中,只有当 p_1 为真时,才考察 p_2,否则考察 p_3.p_1 可以认为是 p_2 和$\neg p_2$ 的前置条件,而$\neg p_1$ 可以认为是 p_3 和$\neg p_3$ 的前置条件.令$\mathcal{K}_4\equiv(p_1\wedge p_2)\vee(p_1\wedge\neg p_2)\vee(\neg p_1\wedge p_3)\vee(\neg p_1\wedge\neg p_3)$,则判定$\mathcal{K}_4$ 表达了整个程序代码段的覆盖情况.先将判定$\mathcal{K}_4$ 转化为它的不可归约的合取范式,然后采用前面的分解与合成的方法,即,先分解为子判定 p_1,p_2 和 p_3 的条件之间的测试,再利用测试用例自然组合的方法得到判定$\mathcal{K}_4$ 的测试用例.因此,在基于规格说明的测试中,要考察一个判定对另一个判定的影响,特别是当判定之间有公共条件的情况时.

5 相关工作

研究人员已经提出了多种测试准则,如 MC/DC[2,7],RC/DC[3],MUMCUT[10,11]等.MC/DC 是在 C/DC(条件/判定覆盖)[13]的基础上发展起来的,它比 C/DC 多了一条:判定中的每一个条件能够独立地影响一个判定的结果.RC/DC 比 MC/DC 更严格,在其基础上又增加了一条:判定中的每一个条件能够独立地保持一个判定的结果.MUMCUT 是一个混合的测试准则,集成了 MUTP(multiple unique true point),MNFP(multiple near false point)和 CUTPNFP(corresponding unique true point and near false point pair)策略,以发现 IDNF(irredundant disjunctive normal form)表示的判定中相同类型的错误.IDNF 是一个析取范式,其中不能有冗余的布尔变量.

Vilkomir 和 Bowen[8]详细分析了 MC/DC 和 RC/DC 测试准则,用实例阐述了测试准则的形式化表示可以消

除非形式化表示带来的模糊性.Yu 和 Lau[9]将 MC/DC,MUMCUT 以及其他逻辑覆盖测试准则进行了比较研究,并建议了不同的应用场合.Offutt 等人[4]提出了基于状态的规格说明的 4 个测试准则:迁移覆盖、全谓词覆盖、迁移对覆盖和完全序列覆盖.Zhu[6]分析了基于单元测试的覆盖准则间的包含关系,提出了评估错误检测能力、发现错误的概率以及期望错误数这两种方法.

MC/DC 要求判定的真值改变随主要条件真值的不同而改变,未考虑一个条件真值的改变对判定的真值没有影响的情况,MUMCUT 也是如此.RC/DC 虽然考虑了测试条件取值变化对判定的值“不改变”的情况,但它过于严格,在测试显式条件时,要求其他隐式条件的取值保持不变,而我们提出的掩盖条件覆盖测试准则对此没有限制,也不像 RC/DC 那样同时考虑了一个条件真值的改变对判定的真值有影响的情况.另外,我们提出的全真判定覆盖、全假判定覆盖、完全子判定覆盖、唯一条件真覆盖以及唯一条件假覆盖等测试准则与 MC/DC,RC/DC 以及 MUMCUT 的概念都不一样.本文的工作和其他有关测试准则方面的成果,如文献[4]和文献[6]等也有所不同(与文献[6]提出的测试准则的类别不一样,他们分析的是基于程序的,我们提出的是基于规格说明的;而与文献[4]提出的测试准则在概念上有所不同).因此,本文的研究和上述文献都有不同之处.

为了更详细地对测试准则进行比较,可以考察它们之间的包含关系.假设 S 是所有测试规格说明的集合,R 是所有测试需求的集合,T 是所有测试用例的集合,$\mathbb{P}(T)$表示 T 的幂集,C 是所有测试准则的集合.令 $c_1,c_2\in C$,若对于 S 和 R 中的任何一对相应的元素,$\mathbb{P}(T)$中的任何一个测试集对于测试准则 c_1 是充分的就蕴含着它对于 c_2 是充分的,则称 c_1 包含 c_2,记作 $c_1\leqslant c_2$(或 $c_2\leqslant c_1$).简单地说,测试准则 c_1 包含 c_2,当且仅当满足 c_1 的每个测试集也同时满足 c_2.可以看出,包含关系定义了测试准则之间的一个偏序关系.因此,包含关系具有传递性.

包含关系的一个基本属性是:测试准则 c_1 包含 c_2,当且仅当对任意的测试规格说明和测试需求,测试准则 c_2 的测试集是 c_1 的测试集的子集.若测试准则 c_2 的每个测试集具有属性 ρ,则包含 c_2 的测试准则 c_1 的任何测试集也具有该属性.包含关系实际上是根据测试方法的严格性所进行的充分性准则的比较.关于测试准则之间包含关系的更详细的阐述见文献[6].

下面给出的是判定覆盖(DC)、条件覆盖(CC)、组合覆盖(CoC)、活动条件覆盖(一般活动条件覆盖(GACC)、相关活动条件覆盖(CACC)、受限活动条件覆盖(RACC))、掩盖条件覆盖(一般掩盖条件覆盖(GMCC)、受限掩盖条件覆盖(RMCC))、全真判定覆盖(FTDC)、全假判定覆盖(FFDC)、完全子判定覆盖(ASDC)、唯一条件真覆盖(UCTC)、唯一条件假覆盖(UCFC)、MC/DC、RC/DC 以及 MUMCUT 等测试准则之间的包含关系,如图 1 所示.在图中,包含关系以从上到下的箭头方向来表示.从图中可以看出,判定覆盖和条件覆盖都比较弱,且都不能互相包含对方,而组合覆盖是最强的.

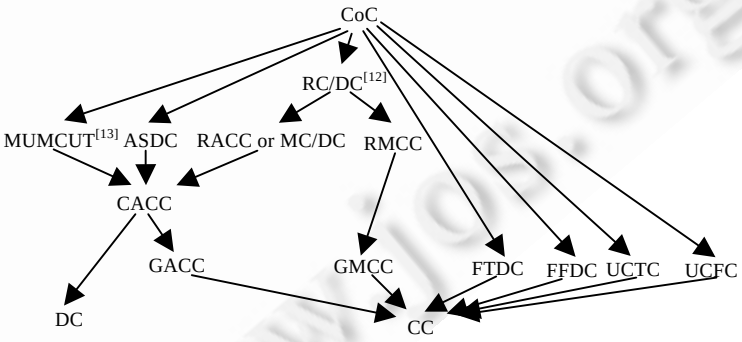

Fig.1 Subsumption relation graph among testing criteria

图 1 测试准则间的包含关系图

各种测试准则之间的包含关系可以根据它们的相关定义直接得到.例如,RC/DC 就包含 RMCC.假设 TS 是满足 RC/DC 的任意一个测试集,根据 RC/DC 的定义,TS 中的测试用例必定测试了每个条件真值的改变分别对判定的值“不改变”和“改变”的情况;而又根据 RMCC 的定义,TS 必定也满足了 RMCC.

不同的测试准则有其自身的优缺点,表现出不同的错误检测能力.研究测试准则间的包含关系对于在具体的软件测试中应该采用何种测试准则对软件进行查错以及查错的效率分析等方面具有重要的意义.一般来说,要测试判定中条件的所有组合是不大可能的,因此,组合覆盖是一个理想的情况.判定覆盖和条件覆盖又太弱,不能查找到大多数错误.而一般活动条件覆盖、一般掩盖条件覆盖、受限掩盖条件覆盖、全真判定覆盖、全假判定覆盖、唯一条件真覆盖、唯一条件假覆盖等又不包含判定覆盖,检测错误的能力也比较弱.对于安全性要求不是太高的应用场景,可以使用这些测试准则.若安全性要求较高,则建议使用 RC/DC 准则,也可以将 MUMCUT、完全子判定覆盖、受限活动条件覆盖、受限掩盖条件覆盖等准则结合使用.在基于规格说明的软件测试中,通常先测试条件取值变化对判定的值“改变”的情况,然后再测试条件取值变化对判定的值“不改变”的情况.也就是说,先使用活动条件覆盖准则,后使用掩盖条件覆盖准则,这样就能保证找到绝大多数常见的错误.还可以根据软件的不同要求,在不同时间、基于不同的测试准则测试软件的不同部分.

6 结束语

本文研究的是基于形式规格说明的测试用例生成方法,所针对的是 Z 和 VDM 一类的基于模型的形式规格说明,它们的主体(前置、后置条件)是判定,所以我们研究的测试准则是逻辑覆盖测试准则.根据逻辑覆盖测试准则,从形式规格说明产生的测试用例是抽象测试用例,对其进行实例化即可产生具体的包括测试数据和预期输出的测试用例.本文的主要贡献如下:

- 针对已有决定性逻辑覆盖测试准则的不足,提出了掩盖性逻辑覆盖测试准则,并对其进行了详细分析.这种准则是逻辑覆盖测试准则的补充.根据该准则生成的测试用例能够检测出条件的掩盖性引起的错误;
- 分析了条件之间的约束关系.阐明了逻辑覆盖中的一个问题:如何考虑条件之间的耦合性,即一个条件的取值不能独立地改变,而会受到其他条件的约束.采用将“耦合性”关系加入到判定中的方法来考察;
- 提出了复杂判定分解与合成的方法.阐明了逻辑覆盖中的另一个问题:如何理解同一个条件在判定中的多次出现;
- 研究了判定之间的关系.讨论了判定在程序中的位置问题.把一个判定看成是另一个判定及其否定的前置条件;
- 提出了全真判定覆盖、全假判定覆盖、完全子判定覆盖、唯一条件真覆盖以及唯一条件假覆盖等测试准则;
- 分析了提出的测试准则与部分相关的测试准则之间的包含关系,并建议了不同测试准则适用的应用场景.

形式规格说明中的判定可以用来开发黑盒测试用例,也可以使用数据流和控制流技术,通过逐步精化的方式增量地产生测试用例.下一步的研究工作是把本文提出的基于规格说明的测试准则应用到 Web 应用中来.根据这些测试准则生成针对于 Web 应用的测试用例.另外,Zhu[6]引入了软件测试中单调度量的概念来分析一个测试用例集能发现错误的期望个数与测试准则之间的关系.文献[14]从测试需求约简的角度考虑测试用例集的优化.今后的研究也包括对本文提出的测试准则进行度量与评估,根据提出的测试准则寻求生成最小的测试用例集的方法.

References:

[1] Amman P, Offutt J. Coverage criteria for logical expressions. In: Stephanie K, ed. Proc. of the 14th Int’l Symp. on Software Reliability Engineering. Washington: IEEE Computer Society Press, 2003. 99−107.

[2] Chilenski J, Miller S. Applicability of modified condition/decision coverage to software testing. Software Engineering Journal, 1994,9(5):193−200.

[3] Vilkomir SA, Bowen JP. Reinforced condition/decision coverage (RC/DC): A new criterion for software testing. In: Proc. of the 2nd Int'l Conf. of Z and B Users. LNCS 2272, Heidelberg: Springer-Verlag, 2002. 295−313. http://core.ecu.edu/vilkomirs/Papers/zb2002_Vilkomir.pdf

[4] Offutt J, Xiong Y, Liu SY. Criteria for generating specification-based tests. In: Proc. of the 5th IEEE Int'l Conf. on Engineering of Complex Computer Systems. Washington: IEEE Computer Society Press, 1999. 119−129. http://www.ise.gmu.edu/~offutt/rsrch/papers/SpecsICECCS99.pdf

[5] Vilkomir SA, Bowen JP. Formalization of software testing criteria using the Z notation. In: Proc. of the 25th IEEE Annual Int'l Computer Software and Applications Conf. Washington: IEEE Computer Society Press, 2001. 351−356. http://reference.kfupm.edu.sa/content/f/o/formalization_of_software_testing_criter_95390.pdf

[6] Zhu H. A formal analysis of the subsume relation between software test adequacy criteria. IEEE Trans. on Software Engineering, 1996,22(4):248−255. [doi: 10.1109/32.491648]

[7] Hayhurst KJ, Veerhusen DS, Chilenski JJ, Rierson LK. A practical tutorial on modified condition/decision coverage. Technical Report, NASA/TM-2001-210876, Hampton: Langley Research Center, 2001.

[8] Vilkomir SA, Bowen JP. From MC/DC to RC/DC: Formalization and analysis of control-flow testing criteria. Formal Aspects Computing, 2006,18(1):42−62. [doi: 10.1007/s00165-005-0084-7]

[9] Yu YT, Lau MF. A comparison of MC/DC, MUMCUT and several other coverage criteria for logic decisions. Journal of System and Software, 2006,79(5):577−590. [doi: 10.1016/j.jss.2005.05.030]

[10] Yu YT, Lau MF, Chen TY. Automatic generation of test cases from Boolean specifications using the MUMCUT strategy. Journal of System and Software, 2006,79(6):820−840. [doi: 10.1016/j.jss.2005.08.016]

[11] Chen TY, Lau MF, Yu YT. MUMCUT: A fault-based strategy for testing Boolean specifications. In: Proc. of the 6th Asia-Pacific Software Engineering Conf. Washington: IEEE Computer Society Press, 1999. 606−613. http://www.computer.org/portal/web/csdl/doi/10.1109/APSEC.1999.809656

[12] Liu L, Miao HK. Axiomatic assessment of logic coverage software testing criteria. Journal of Software, 2004,15(9):1301−1310 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/15/1301.htm

[13] Zhu H, Jin LZ. Software Quality Assurance and Quality Testing. Beijing: Science Press, 1997 (in Chinese).

[14] Zhang XF, Xu BW, Nie CH, Shi L. An approach for optimizing test suite based on testing requirement reduction. Journal of Software, 2007,18(4):821−831 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/18/821.htm [doi: 10.1360/jos180821]

附中文参考文献:

[12] 刘玲,缪淮扣.对逻辑覆盖软件测试准则的公理化评估.软件学报,2004,15(9):1301−1310. http://www.jos.org.cn/1000-9825/15/1301.htm

[13] 朱鸿,金凌紫.软件质量保障与测试.北京:科学出版社,1997.

[14] 章晓芳,徐宝文,聂长海,史亮.一种基于测试需求约简的测试用例集优化方法.软件学报,2007,18(4):821−831. http://www.jos.org.cn/1000-9825/18/821.htm [doi: 10.1360/jos180821]

钱忠胜(1977－),男,江西鹰潭人,博士,讲师,主要研究领域为软件工程,Web 软件的测试.

缪淮扣(1953－),男,教授,博士生导师,CCF 高级会员,主要研究领域为软件形式化方法,软件工程.