修正的条件 / 判定覆盖分析工具评估与选择

张卫民　孙　军
（北京航天飞行控制中心）

摘　要　使用工具进行修正的条件/判定覆盖（MC/DC）分析可以简化软件验证工作，但必须对候选的工具进行认真评估，确定每种工具的功能性和局限性,着重考虑的因素包括：监测语句的类型、在何处监测语句（源码还是目标码）、可监测的条件和判定的最大个数、确定独立影响所用的算法、关系运算符的处理、位运算符的处理、组合表达式的处理、单步逻辑运算的处理和插装影响等。

关键词　修正的条件/判定覆盖　覆盖分析工具　工具评估
中图分类号　TP311.56　**文献标识码**　A　**文章编号**　1674-5825（2009）03-0040-04

1 引　言

MC/DC 分析是一项非常复杂、繁琐的工作,利用工具来执行 MC/DC 分析这种重复、复杂和耗时的软件验证任务有助于消除因精神疲劳而可能产生的错误,还可以为软件验证人员节省大量的时间。尽管使用工具来自动进行结构覆盖分析能够带来明显的好处,但是,如果未对工具的作用进行适当的确认,使用工具也可能具有潜在的危害。

一旦决定要使用工具自动进行覆盖分析，就应当对候选的工具进行认真的评估，确定每种工具的功能性和局限性。对于工具的所有局限性,都应当执行适当的验证过程来加以说明。文章[1]中介绍的 MC/DC 分析方法可以用来帮助评价一个工具是否恰当、完全地执行了所有功能。

为了避免出现与自动覆盖分析相关的一些问题，本文概要地介绍了结构覆盖分析工具是如何工作的，以及在选择和认证结构覆盖分析工具时应该考虑的一些重要因素。但是,这里介绍的因素并没有覆盖在使用自动工具进行结构覆盖分析时应考虑的所有的问题。另外,也没有局限于讨论特定的覆盖分析工具。

2 覆盖分析工具的工作方式

结构覆盖分析工具通常是利用插装代码或其它的插入技术来增加测试的可见性。一般情况下,分析工具通过在源代码或目标代码中加入一系列的探针、标记或其它的监测机制来进行代码插装。代码插装使得分析工具能够准确地确定执行了代码的哪些部分。代码插装完毕后执行测试用例,这时覆盖分析工具就可以追踪测试用例执行了哪些代码,如果需要进行更为复杂的分析,则还要追踪这些代码是如何被执行的。如果指定了结构覆盖的通过或失败标准,分析工具则根据这些标准对代码进行分析,并给出通过或失败报告。如果没有指定通过或失败标准,分析工具则应该报告测试用例达到的结构覆盖水平。

因为插装改变了代码，所以必须证明插装没有隐藏错误,也没有引入错误。可以通过对源代码和插装后的代码分别运行测试用例并比较执行结果来说明插装是否引入或隐藏了错误。这就意味着,在未插装代码以及被测设备和测试设备中，要具有充分的可见性,这样才能观察到与插装版本相对应的结果。当然，可能还有其它的方法来说明插装对期望结果

来稿日期:2008-02-25
作者简介:张卫民(1962.11—),男,硕士,研究员,主要从事软件工程、软件可靠性、过程改进、项目管理、实时软件系统研究。

不会产生影响。不管采用什么方法,须要牢记的一点是，覆盖工具确认的是经插装后的代码满足 MC/DC 标准。这里的假设是未插装代码也满足 MC/DC 标准。这一假设是否正确依赖于所使用的插装方法和编译器。

不同的覆盖分析工具使用不同的插装方案。重要的是，我们要知道某一特定工具使用的是什么方案。

3 选择或认证工具考虑因素

插装方案并不是工具选择和认证时唯一须要了解的事项,下面的因素也非常重要：

(1)监测语句的类型；

(2)在何处监测语句(源码还是目标码)；

(3)可监测的条件和判定的最大个数；

(4)确定独立影响所用的算法；

(5)关系运算符的处理；

(6)位运算符的处理；

(7)组合表达式的处理；

(8)单步逻辑运算的处理；

(9)插装影响。

下面分别叙述有关这些因素的关键问题，以及为什么要提出这些问题。

3.1 监测语句的类型

工具监测所有的覆盖点吗?

工具能够完全、适当地处理一个单一语句产生的所有覆盖需求的覆盖点吗?

为了证明 MC/DC，结构覆盖分析工具应该监测顺序语句、入口和出口点、判定和分支语句,以及布尔条件。有的工具并不是支持 MC/DC 要求的所有覆盖点。例如,不是所有的结构覆盖工具都支持入口和出口点覆盖，如果有其它的手段可用来覆盖入口和出口点，这类工具可以用来支持部分的结构覆盖分析。编程语言也可能影响工具监测的语句的类型。例如，有的语言就没有布尔或逻辑类型，对于这类语言，分析工具可能必须根据使用的布尔运算符推断哪些表达式是布尔表达式,哪些不是布尔表达式。在这种情况下，覆盖分析工具就有可能无法监测所有的布尔表达式。

一个结构覆盖分析工具还应该能够监测语句的多种覆盖点。例如,考虑下面的返回语句：

return (A and B) or C;

对这个语句,应该监测下面的覆盖点：

(1)语句–必须至少调用一次；

(2)出口点–必须至少调用一次；

(3)判定–所有可能结果(false,true)必须至少取值一次；

(4)条件–每个条件(A、B、C)的所有可能结果(false,true)至少取值一次,每个条件(A、B、C)必须显示出其独立影响。

不同的返回语句(例如,return(x+y)/z;)需要监测不同的覆盖点。

3.2 源代码监测与目标码监测

分析显示出了目标代码级的覆盖等价于源代码级的覆盖吗?

有的结构覆盖工具在源代码级监测覆盖，而有的在目标码一级监测覆盖。在目标代码级达到了 MC/DC,并不一定等价于在源代码级达到了 MC/DC。

如果能够提供分析，证明在目标码级进行的覆盖分析等价于源代码级的同种覆盖分析，那么就可以在目标码一级证明 MC/DC。因此,如果使用的工具是在目标码一级监测覆盖，则需要额外的分析来确认目标码级和源代码级覆盖间的等价性。这种分析通常都不是很简单，并且需要在项目的早期由认证权威部门对这种分析进行评审。

3.3 可监测条件与判定的最大个数

工具对一个给定的布尔表达式中监测的条件的个数有限制吗?

工具对被监测的条件、判定或语句的总的个数有限制吗?

有的工具限制一个布尔表达式中能够监测的条件的个数,有的还限制被监测的条件、判定或语句的总的个数。对于一个单一判定,监测方案有可能依赖于该判定中条件的个数。比如,一种工具可能对于 8 个以下条件的判定使用一种方案,对 9 到 16 个条件的判定使用另一个方案，对 17 到 32 个条件的判定使用又一种方案,并且可能不处理多于 32 个条件的判定。在有的情况下，工具可能完全忽略大的表达式,或者是仅监测表达式的一部分。应当明确地识别和了解工具对监测覆盖点的限制。在这些限制对覆盖分析产生影响的情况下，应当定义和记载缓解策略和过程。例如,如果要监测的元素的总数超出了工

具的限制，可以采用的一种方法是监测系统的不同的子集，对每个监测子集运行一次测试用例，然后将多次分析的结果合成为一个对系统的分析。如果一个表达式对于工具来说太大，那么有可能需要进行人工分析。

3.4 确定独立影响使用的算法

确定独立影响的依据是什么？

不同的结构覆盖分析工具使用不同的算法来确定条件的独立影响。文章[1]中介绍了确定条件独立影响的基本方法。其它的常用方法使用表达式树、布尔差分函数、KV-图或功能树等技术来确定条件的独立影响。

3.5 关系运算符处理

工具完全、适当地监测了存在关系运算符的条件的独立影响了吗？

当关系运算符用于布尔表达式时，例如（x<y）and（x>z），必须证明表达式中条件的独立影响。因为关系运算符不是布尔运算符，因此，应当对工具进行检查，确保在关系运算符存在时它能适当地对条件进行监测。

3.6 位运算符的处理

工具适当地将位运算表达式当作判定考虑了吗？

在一些编程语言中，位运算使用的是与布尔运算不同的运算符。但是，这时针对每一位的运算实质上都是布尔运算。因此，在进行结构覆盖分析时，对于位运算也要进行分析。所以，对结构覆盖分析工具应该进行检查，确保它对位运算表达式进行正确的处理和分析。

3.7 组合表达式的处理

结构覆盖分析工具支持组合表达式的分析吗？

有时为了使逻辑概念或表示形式更加清晰，可能需要将一个表达式分解为多个等价的表达式，或者反过来，将多个表达式合成为一个等价的表达式。例如，考虑下面的三个语句：

E:= B and C;

A:= E or D;

A:=（B and C）or D;

这三个语句都包含有一个判定，且前两个语句在逻辑上等价于第三个语句。但是，如果将一个复杂的判定分解为了等价的多个简单的判定，对各个简单判定满足 MC/DC 的测试集，对于原有整体的复杂判定不一定满足 MC/DC。因此，必须对工具进行检查，确定它是将各个部分判定看作是独立的判定，还是将它们看作是一个整体判定。

3.8 单步逻辑运算的处理；

工具适当地分析单步逻辑运算了吗？

假设我们要计算一个布尔表达式的值：

A:=（B and C）or D;

对于汇编语言这样的低级语言，由于不存在布尔表达式，所以无法直接在程序中书写上面的复杂逻辑表达式，每个逻辑运算都需要用一条独立的汇编指令表示。上述处理过程的汇编语言编码为：

GET B;

AND C;

OR　D;

PUT A;

对于像汇编语言表示的这种由单步逻辑运算实现的布尔表达式，同样需要证明各个条件的独立影响。因此，必须对工具进行检查，确定它是否支持这种单步逻辑运算的处理与分析。

3.9 插装影响

插装影响了结构覆盖分析了吗？

用于结构覆盖分析的探针的影响可以分为以下几类：

（1）为了支持探针需要增加的资源、存储和吞吐量；

（2）由于软件探针的存在而产生的编译器影响；

（3）由于插装和非插装代码间的不同引起的环境、编译器或目标因素。

下面分别讨论这几类影响。

3.9.1 对资源的影响

插装对存储、吞吐量，及其它资源有什么影响？

基于时间差的实时计算有不同的结果吗？

插装软件需要的额外存储会引起不同的存储页面越界吗？

理想地，插装应该只增加为支持探针本身所需的开销，而使软件和系统的所有其它方面保持不变。对于硬件探针，能够在很大程度上实现这种理想情况。如果运行软件的硬件被设计为以硬件的速度直接从硬件提供实时执行数据，则可以完全达到这种理想情况。关于硬件探针的一个主要问题是，它们捕

捉的是软件中实际正在发生的情况信息吗？不幸的是，硬件探针一般是监测指令存取，而指令存取监测有可能被带有高速缓存的硬件体系结构产生的假象所欺骗。这是因为，一条指令被取到了高速缓存并不意味着它被实际地执行了。还因为，一条指令被执行了，并不意味着它被正确、适当地执行了。像对待软件探针一样，对硬件探针也需要进行分析，确定它们实际上产生什么信息，在什么环境下产生这些信息，以及附带有什么假设条件。

软件探针会增加存储和 CPU 周期开销，因此，在存储或吞吐关键部件中，插装后的软件有可能执行不正常。吞吐关键部件可能没有能力处理执行探针所需的额外的 CPU 周期。存储关键部件可能没有能力处理探针指令所需的额外存储。

除了存储和吞吐量，为了给探针提供支持，可能还需要其它的资源。例如，有可能需要一种通信机制来捕获和传输由探针提供的执行数据。对这些额外增加的资源需要进行认真的研究，以确定它们对开发、验证和结构覆盖分析过程的影响。

3.9.2 对编译器的影响

编译器的行为受软件探针的影响吗？

插入到源代码中的软件探针有可能改变编译器的行为和产生出的可执行代码。理想地，可执行代码中唯一的变化应该只是为探针提供支持，软件的所有其它功能方面仍应该正确执行（除了吞吐量和存储使用之外）。但是，探针的存在有可能导致编译器产生出不能正常执行的代码。

例如，一个表达式需要调用两个函数，比如 A 和 B，其中函数 B 依赖于函数 A 运行产生的结果。对于未插装软件，因为编译器产生的代码总是先调用 A，后调用 B，所以它总是能够正常执行。经过插装之后，编译器产生的代码有可能是调用 B 早于调用 A。这时，在系统环境下插装后的代码就不能正确执行了，而且这种不正确的行为与探针引起的存储和时间影响没有关系。

3.9.3 环境因素

软件环境的哪些成分会因为结构覆盖分析工具的需要而改变？

使用自动结构覆盖分析工具时，对于已插装软件可能需要与未插装的软件所不同的工具或工具设置。其差别可能包括：

（1）编译器–不同的编译器，或者是不同的编译器设置；

（2）连接器–不同的连接器，或者是不同的连接器设置；

（3）目标–不同的目标环境（比如，仿真器或模拟器）。

上述三项中的每一项都可能影响插装与未插装代码间执行结果的保真度。对于插装的保真程度应当进行评估，确定结构覆盖分析工具的特有作用。

4 结束语

本文简单讨论了 MC/DC 评估和选择工具时应考虑的一些主要技术因素。但是，在具体选择覆盖分析工具时，不能仅仅局限于只考虑这些因素，而是应当根据项目的具体情况，综合技术和经济等多方面的因素，选择适合于具体项目的工具。另外，在评估覆盖分析工具时，还应对其标称的各项功能进行认真、详细的分析和测试，确保其正确地实现了各项功能。 ◇

参 考 文 献

[1] Kelly J H, Dan S V, John J C, et al. A Practical Tutorial on Modified Condition / Decision Coverage [R]. NASA/TM–2001–210876.

[2] RTCA/DO–178B. Software Considerations in Airborne Systems and Equipment Certification[S]. RTCA, Inc. Washington, D. C.. December 1992.

[3] RTCA/DO–248B. Final Report for Clarification of DO–178B. Software Considerations in Airborne Systems and Equipment Certification [S]. RTCA, Inc. Washington, D. C.. October 12, 2001

（下转第 53 页）