第35卷 第17期 计 算 机 工 程 2009年9月
Vol.35 No.17 Computer Engineering September 2009
·软件技术与数据库· 文章编号：1000—3428(2009)17—0040—03 文献标识码：A 中图分类号：TP311

MC/DC 最小测试用例集快速生成算法

段飞雷，吴 晓，张 凡，董云卫
(西北工业大学计算机学院，西安 710072)

摘 要：以条件判定组合的语法树为研究对象，总结语法树的左右分支对判定结果的影响方式及结果，根据其影响特点提出超越语法树的快速生成改进判定/条件覆盖最小测试用例集的算法。与其他算法在在空间及时间方面进行比较，结果表明该算法具有一定的优越性。
关键词：改进判定/条件覆盖；最小测试用例集；快速生成算法

Rapidly Generating Algorithm for Minimum Test Case Set on MC/DC

DUAN Fei-lei, WU Xiao, ZHANG Fan, DONG Yun-wei
(School of Computer, Northwestern Polytechnical University, Xi'an 710072)

【Abstract】This paper studies syntactic tree of combination of conditions and decision, and what is influence of decision of the left and right branches of the syntactic tree, and how it does on results. According to the influence, an algorithm is presented to rapidly generate minimum test case on MC/DC without concerning structure of syntactic tree. An implementation of this algorithm is developed to do an experiment. Comparing with others, experimental data shows that it is excellent to rapidly generate test case set with MC/DC.
【Key words】MC/DC; minimum test case set; rapidly generating algorithm

1 概述

改进条件/判定覆盖(MC/DC)是软件结构覆盖测试准则之一，是在判定/条件覆盖(C/DC)的基础上发展起来的，但是它在测试用例选取的方法、所对应的测试集、测试中的覆盖面以及用途等方面与 C/DC 相比有很多增强。与其他结构的覆盖准则相比，MC/DC 具有很多优点，它继承多重条件覆盖的优点，同时只是线性增加了测试用例的数量，并且对操作数及非等式条件变化反应敏感，还可以得到理想的目标代码覆盖率[1]。近年来，MC/DC 得到了广泛的应用。本文针对 MC/DC 测试用例的生成方法进行探讨，介绍了一种快速生成测试用例的算法思想及过程，并与其他算法在生成空间以及时间上进行了比较。

2 相关研究

条件、判定是描述软件结构和控制流的基本元素，也是软件测试过程中重点测试的对象。

条件：不包含逻辑操作符的逻辑表达式。仅由关系操作符(<, >, =等)构成的逻辑表达式属于条件。

判定：由条件及零个或多个逻辑操作符(如 and, or, not, xor)组成的逻辑表达式。如果一个判定中的某一条件多次出现，那么其每次出现均为不同的条件。

对程序中条件和判定测试充分性的不同造成了不同的软件测试准则，有 SC, CC, DC, C/DC, MC/DC, MCC 等[2]，但这些准则的覆盖等级都不一样，其中，MC/DC 以多重条件覆盖、线性增加测试用例数量[1]等特点而受到重视。航空电子标准 RTCA/DO-178B 的 A 级认证要求程序的每一行代码都要进行 MC/DC 覆盖测试[3]。

MC/DC 要求判定中每一个条件的所有可能结果至少出现 1 次，每一个判定本身的所有结果也至少出现 1 次，每个入口与出口点至少被唤醒 1 次，并且每个条件都能单独影响判定的结果，即在其他条件不变的情况下，改变这个条件的值，使得判定结果改变[4]。

对于一个具有 N 个条件的布尔表达式，测试集合中至少包括 N+1 组元素[5]。设计条件独立的影响结果的最小测试用例集(最小真值表)通常有唯一原因法、屏蔽法和最小真值表法 3 种方法。唯一原因法仅关心条件值和判定结果可以改变，而其他条件必须固定；屏蔽法对一个逻辑操作符的特定输入能隐藏对该操作符的其他输入的影响[1-2]；最小真值表法结合前 2 个方法的思想，将原始的布尔表达式转换为逆波兰式，归约得到从原始的布尔表达式到最小化测试集合的算法[6]。

本文融合最小真值表法与布尔表达式的树型结构思想，设计了一个快速生成测试用例集的算法，该算法本身并不对布尔表达式进行任何转化，而是直接从原始的布尔表达式得出最小化测试用例集。

3 快速生成算法

3.1 算法思想

任何一个布尔表达式都可以归结为一个抽象语法树，其中，树的叶节点为布尔表达式的条件，树的分支节点为布尔表达式的判定。对于布尔表达式 A&&B||C，其抽象语法树如图 1 所示。分析此语法树，要使其左子树的结果(不管具有多少个叶节点)能单独地影响判定的结果，必须使其右子树的结果尽可能少影响判定的结果。对于根节点为&&的子树来说，

基金项目：国家“863”计划基金资助项目“构件化嵌入式软件测试方法及其工具研究”(2008AA01Z142)
作者简介：段飞雷(1983－)，男，硕士研究生，主研方向：测控技术，嵌入式系统；吴 晓、张 凡、董云卫，博士
收稿日期：2009-02-08 **E-mail：**lightningfly@gmail.com

只有其右子树的结果为 1，才能满足其左子树单独地影响判定结果。对于根节点为||的子树，其右子树的结果为 0 才能满足其左子树能够单独地影响判定结果。在某种意义上，抽象语法树所有右子树的判定结果已经被根节点的判定限制了。

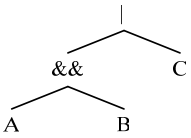

图 1　A&&B||C 的抽象语法树

在所有节点都尽可能被限制的情况下，整个抽象语法树只有最左节点没有被限制，因为最左节点不属于任何右子树。那么在除最左节点外的所有节点都被限制的情况下，最左节点便是单独影响判定结果的那个节点，此时可生成最左节点单独影响判定结果的 2 组用例。同理，影响每个子树判定结果的便是每个子树的最左节点。

对于每棵语法树来说，置换左右子树，其判定结果不受影响。此时，以前根节点所制约的右子树的最左节点变成了此树的最左节点，而以前左子树的最左节点便由根节点的判定情况所制约，并且不管根节点的判定如何制约，所生成的 2 组用例中必定有 1 组包含于置换前生成的 2 组测试用例集中(2 个最左节点用例相同的那组)。这样把每个子节点都置换为最左节点 1 次，每个节点都可以单独地影响判定结果。

在算法实现的过程中，可根据布尔表达式与语法树对应特征(叶子节点都为条件，分支节点都为判定)绕过语法树直接对布尔表达式进行归约。

3.2　生成算法中的概念

当前条件：当前独立影响判定结果的条件。

直接判定：条件的直接判定为条件之前离条件最近的判定，对应于语法树中父节点的判定。一个条件的直接判定将直接影响该条件用例的选取结果。第 1 个条件对应于语法树中的最左子树，其前面不可能出现任何判定，因此，第 1 个条件没有直接判定。

前置条件：条件的前置条件由条件的直接判定确定：若直接判定之前为一个条件，则此条件就是直接判定的前置条件；若直接判定前为条件判定组合，则此组合的第 1 个条件为前置条件。前置条件对应于语法树中直接判定的左子树的最左节点。与直接判定一样，第 1 个条件没有前置条件。

默认用例：根据算法思想，条件的默认用例由条件的直接判定确定：若直接判定为&&，则此条件的默认用例为 1；若直接判定为||，则此条件的默认用例为 0。第 1 个条件因为对应于语法树的最左节点，所以没有默认用例。

自由条件：需要确定其用例的条件，其指向置换过程中需要被置换的右子树的最左节点(生成过程中能有 1 个条件为自由条件)。

3.3　生成算法

生成算法主要包括以下 6 步：

(1)设置第 1 个条件为当前条件，分别选取 1 和 0 作为此条件的用例，与其后条件的默认用例相组合生成 2 组测试用例，每组测试用例的判定结果由当前条件的用例确定：若当前条件的用例为 1，则此组测试用例的判定结果也为 1；若当前条件的用例为 0，则此组测试用例的判定结果也为 0(此时已经生成 2 组测试用例)。

(2)生成下一组测试用例：若当前条件为最后一个条件，则测试用例集生成过程完成；否则，后移 1 个条件作为当前条件。

(3)当前条件的测试用例由其默认用例翻转得出，设置当前条件为自由条件。

(4)自由条件的默认用例向前传递给其前置条件，并设置其前置条件为自由条件。

(5)此时若自由条件为布尔表达式的第 1 个条件，则继续；否则转向(4)。

(6)此组用例中其余未确定的用例由其条件的默认用例填充，判定结果由当前条件的用例确定。此组用例生成完成，转向(2)。

具体的生成算法流程如图 2 所示。

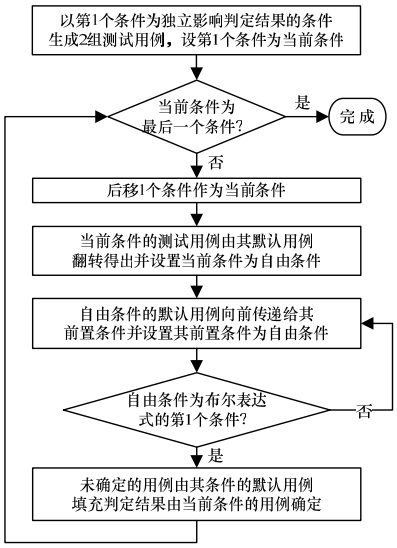

图 2　生成算法流程

3.4　举例说明

以 S=((A&&B)||(C&&D))为例说明算法的执行过程。此布尔表达式共有 4 个条件，因此，生成 5 组测试用例。

步骤 1 根据布尔表达式生成第 1 个条件单独影响判定结果的 2 组测试用例，除了第 1 个条件外，其余条件的用例为其默认用例。

A	B	C	D	Result
1	1	0	1	1
0	1	0	1	0

步骤 2 条件 A 不是最后一个条件，当前条件指向条件 B。当前条件 B 的测试用例由其默认用例翻转得出，设置条件 B 为自由条件。

A	B	C	D	Result
-	0	-	-	-

此时自由条件 B 的直接判定&&前面为条件 A，其前置条件为 A。则自由条件 B 的默认用例向前传递给其前置条件 A，并设置 A 为自由条件。

A	B	C	D	Result
1	0	-	-	-

此时自由条件为第 1 个条件，则此组用例的其余用例为其默认用例，判定结果由当前条件 B 的用例得出。

A	B	C	D	Result
1	0	0	1	0

步骤 3 条件 B 不是最后一个条件，当前条件指向条件 C。当前条件 C 的测试用例由其默认用例翻转得出，设置条件 C 为自由条件。

A	B	C	D	Result
-	-	1	-	-

此时自由条件 C 的直接判定||前面为(A&&B)，其前置条件应为 A。则自由条件 C 的默认用例向前传递给其前置条件 A，并设置 A 为自由条件。

A	B	C	D	Result
0	-	1	-	-

此时自由条件为第 1 个条件，则此组用例的其余用例为其默认用例，判定结果由当前条件 C 的用例得出。

A	B	C	D	Result
0	1	1	1	1

步骤 4 条件 C 不是最后一个条件，当前条件指向条件 D。当前条件 D 的测试用例由其默认用例翻转得出，设置条件 D 为自由条件。

A	B	C	D	Result
-	-	-	0	-

此时自由条件 D 的直接判定&&前面为条件 C，则条件 C 为自由条件 D 的前置条件。自由条件 D 的默认用例向前传递给其前置条件 C，并设置 C 为自由条件。

A	B	C	D	Result
-	-	1	0	-

此时自由条件 C 的直接判定||前面为(A&&B)，条件 A 为自由条件 C 的前置条件。自由条件 C 的默认用例向前传递给其前置条件 A，并设置 A 为自由条件。

A	B	C	D	Result
0	-	1	0	-

此时自由条件为第 1 个条件，则此组用例的其余用例为其默认用例，判定结果由当前条件 D 的用例得出。

A	B	C	D	Result
0	1	1	0	1

步骤 5 此时当前条件 D 为最后一个条件，则生成过程完成。

最终生成的测试用例集如表 1 所示。在该用例集中，前 2 组即可表明条件 A 的独立性影响，测试用例对(1, 3)表明条件 B 的独立性影响，测试用例对(2, 4)表明条件 C 的独立性影响，测试用例对(4, 5)表明条件 D 的独立性影响。

表 1　最终生成的测试用例集

组数	A	B	C	D	Result
1	1	1	0	1	1
2	0	1	0	1	0
3	1	0	0	1	0
4	0	1	1	1	1
5	0	1	0	0	0

4　算法分析与验证

根据上面算法的具体步骤便可实现自动生成 MC/DC 测试用例的具体方法。图 3 是由笔者实现的一个 MC/DC 测试用例集辅助生成软件的界面，此软件可分析任何源程序，并自动寻找源程序中的条件判定组合，根据找到的条件判定组合自动生成满足 MC/DC 测试要求的用例集组合，配合软件测试人员生成测试用例。

通过该应用程序可以使测试人员快速查找源程序中的所有条件分支语句以及循环语句中的条件判定组合，并且生成满足 MC/DC 覆盖要求的用例真值表以供参考。

与唯一原因法和屏蔽法相比，快速算法不需要额外的空间存储条件判定组合的完全真值表及存储表明条件独立性影响的测试用例对，其所生成的每一组测试用例都是最终测试用例组所必需的，大大减少了算法过程的存储空间。最小真值表法因为要用额外的堆栈保存条件判定组合的逆波兰式，而且算法过程中生成的真值表并不是最小真值表，还需进行化解，所以与快速算法只需使用 3 个指针相比，仍需要大量的存储空间。同时，快速算法在实现上非常简单，只需对原始布尔表达式的条件依据规则进行归约，即对条件判定组合的 N 个条件进行遍历；唯一原因法和屏蔽法不仅要预先生成完全真值表，还要对完全真值表的 2^n 个测试用例组进行遍历，在遍历结束后还需对满足条件的测试用例组进行筛选，这就需要更多的测试用例生成时间。最小真值表法是对条件判定组合的所有条件与判定进行遍历，但是在遍历之前需要对条件判定组合进行规约，遍历结束后仍需化解真值表为最小真值表，这些过程都极大影响了测试用例组的生成时间。

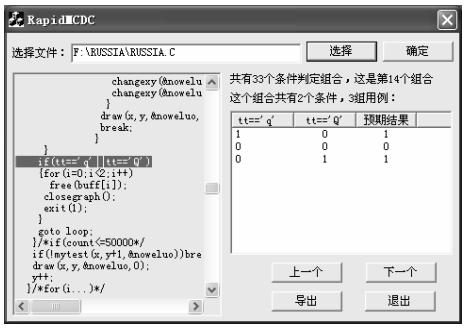

图 3　源程序分析界面

本文对快速生成算法与最小真值表算法的用例生成时间进行了对比。实验选取不同条件数的条件判定组合进行比较，结果显示快速生成算法在计算时间上优于最小真值表算法，特别是在条件数目繁多、组合复杂的情况下，快速生成算法的优越性更加明显，选取用例如表 2 所示。

表 2　选取用例及计算时间结果　　s

选取用例	快速算法时间	最小真值表时间
A&B	0.001 398	0.002 601
A&(B|C)	0.002 120	0.003 464
(A|B)&(C|D)	0.002 787	0.005 048
(A&B)|(C&D&E)	0.003 373	0.006 727
(A&B&C)|(D&E&F)	0.003 569	0.009 905
(A&B&C)|(D&(E|F)|G)	0.004 486	0.018 761

实验计算结果比较如图 4 所示，对于同样的条件判定组合，因为最小真值表方法要经过归约、遍历以及化解 3 个步骤，而快速算法只用扫描条件判定组合一遍即可，所以，最小真值表法所花费的时间多于快速算法，特别是在条件判定组合复杂的情况下，时间上的差异更加明显。

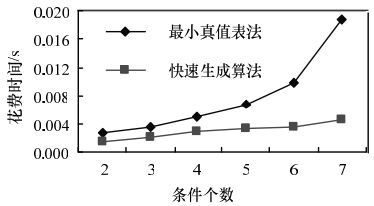

图 4　快速算法与最小真值表算法时间比较

5　结束语

唯一原因法和屏蔽法是通过所有条件的完全真值表得到的，这 2 种方法每次可以得出多个满足 MC/DC 要求的真值表，但是这些真值表里有些并不符合复杂逻辑语句[7]；最小真值表法和快速算法每次虽然只能生成 1 组，但是最小真值表法在对条件判定组合进行归约时，已经考虑了逻辑语句的复杂关系。本文的快速算法是从对条件判定组合逻辑语法

（下转第 45 页）