测试任务流中或分支的完整性验证

方清华[1] 苏锦海[1] 凌祖让[2] 滑冬冬[1]
(解放军信息工程大学 郑州 450004)[1] (78003 部队 昆明 650000)[2]

摘 要 测试任务流中或分支的完整性验证是保证任务流模型正确、稳定、完备的必要条件。基于测试任务流中或分支的完整性分析,给出或分支约束条件集的完整性定义,将问题转换为约束条件集的完整性验证。借鉴哈夫曼树的思想,构造一棵或分支完整性判定树,完成测试任务流或分支的完整性验证。

关键词 或分支,完整性,哈夫曼,测试任务流
中图法分类号 TP14 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2015.10.036

Integration Verification of Or-split in Test Flow

FANG Qing-hua[1] SU Jin-hai[1] LING Zu-rang[2] HUA Dong-dong[1]
(PLA Information Engineering University, Zhengzhou 450004, China)[1] (78003 Troops, Kunming 650000, China)[2]

Abstract Integration verification of or-split in test flow is the necessary condition to insure correctness, stability and maturity of the model. Based on the analysis of the integration in test flow, we gave the integration definition of condition-constrained set representing or-split to convert the problem from or-split to the verification of condition-constrained set. According to the Huffman tree, we constructed a tree to decide whether the or-split in test flow is integrated or not.

Keywords Or-split, Integration, Huffman, Test flow

1 引言

至今,芯片测试系统已经发展到第三代,通过软件控制测试资源来完成测试过程的部分或全部自动化是第三代系统追求的目标。为了实现这一目标,测试任务流模型就成为了芯片测试系统中的关键。测试任务流模型是将现实世界中的测试过程的逻辑及测试活动间的依赖关系等抽象出来,并用一种形式化的、计算机可识别处理的方式表示出来的一种模型。它包含了芯片测试系统中测试任务流执行所需的各种信息,如测试活动、测试资源、执行者、测试相关数据以及测试活动间的依赖关系等[1,2]。

目前,国内外在建模方法的研究上比较成熟。主要的建模方法有:基于状态和活动图的建模方法、基于活动网络的工建模方法、基于 Petri 网的建模方法、基于事务的建模方法等[3-5]。这些方法在描述现实世界中任务过程的能力方面以及模型的形式化语义等方面存在差异。有些方法侧重于现实世界过程描述,有些侧重于模型的形式化语义分析。比较普遍的问题是模型的建立没有坚实的理论基础,因此模型的分析验证就比较困难[6]。

由于 Petri 网拥有严格的形式化语义以及数学理论的支撑,因此基于 Petri 网的分析验证方法的研究比较热门。在国外,基于 Petri 网分析模型的活性(Liveness)、有界性(Boundedness)、覆盖性(Covering)、可达性(Reachability)等较好地解决了模型结构特性和行为特性的分析验证问题[7]。而在国内,由于对 Petri 网的研究起步相对较晚,对结构特性和行为特性进行分析验证的研究较少一些,而主要集中在模型的性能分析上[8]。

这些分析方法都比较依赖于 Petri 网,而且主要从模型结构特性、行为特性和性能分析方面来分析验证模型。本文给出一种测试任务流模型中的或分支完整性验证方法,通过或分支的完整性分析来保证模型可以正确、完备地执行。该方法不依赖所使用的建模方法,通用性好,并且有一定的数学理论基础,可靠实用。

2 测试任务流或分支的完整性分析

测试任务流模型是将现实世界中的测试过程的逻辑及测试活动间的依赖关系等抽象出来而构成的一种流程模型,主要由测试活动及其连接关系组成。在该模型中,或分支是比较常见的一种控制结构,也是比较容易带来问题的一种结构。它需要完整地描述一个测试活动的所有可能情况,而且还不能带来歧义,否则可能导致测试任务流的异常执行,带来不可估量的损失。在现实世界中,每一条分支都是由一定的约束条件来控制的,因此,或分支的完整性问题可以转换为约束条件集的完备性问题来研究。

到稿日期:2014-10-29 返修日期:2015-01-23 本文受核高基重大专项(2012ZX01027004)资助。
方清华(1990—),男,硕士生,主要研究方向为芯片自动测试系统、自动处理技术,E-mail:993288482@qq.com;苏锦海(1963—),男,博士,教授,主要研究方向为自动化基础理论、自动化系统理论、自动控制理论、自动信息理论;凌祖让(1974—),男,硕士,工程师,主要研究方向为专用芯片测试;滑冬冬(1986—),男,硕士生,主要研究方向为专用芯片自动测试系统。

在测试任务流模型中，常见的两种情况是分支的遗漏和冗余。分支的遗漏是指约束条件的集合并不是一个完整的约束集合，也就是缺少某种现实中可能存在的情况的描述，这容易导致测试任务流在运行时无法选择执行的路径。如图 1 所示，图中左半部分很明显，约束对象 X 在大于等于 10 的情况出现遗漏，随着约束对象的增加，如图中右半部分所示，要判断分支的遗漏就变得困难起来。

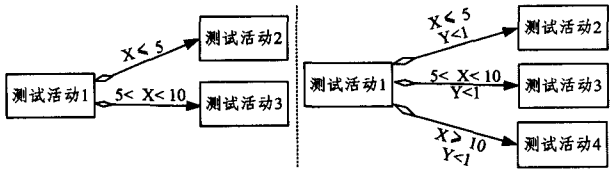

图 1　分支遗漏

分支的冗余是指约束条件的集合中约束条件之间存在交叉，即存在某种情况有多个约束条件满足条件导致多个测试活动满足执行条件，这容易导致测试任务流在运行时会选择多条路径同时执行，从而违背了在模型中使用或分支的本意。图 2 左半部分很明显有一个分支冗余；右半部分的情况就更复杂了，既存在分支遗漏又存在分支冗余，这时光靠人的直观感觉已经不能可靠地判断出该或分支是否存在完整性问题。

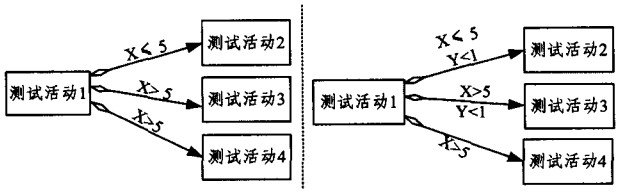

图 2　分支冗余

通过以上分析可以知道，测试任务流中或分支的完整性问题是一个复杂的问题，同时也是一个迫切需要解决的问题，因为如果这个问题解决不好，在后期任务流执行时会带来不可预估的损失。为了完成或分支的完整性验证，给出测试任务流或分支形式化的完整性定义及其判定方法。

3　测试任务流或分支的完整性定义

在测试任务流模型中，对于任意一个或分支选择结构，设其对应的约束条件集为：$COND=\{cond_1, cond_2, cond_3, \cdots, cond_n\}$，如果该约束选择集具有完整性，那么说明对应的或分支具有完整性。其完整性可描述为：任意一个 $cond_i \in COND$ 都能找到对应的唯一的分支，并且所有可能的情况都被考虑到。其相关定义如下所示。

定义 1（受约束变量）　约束条件中的谓词称为受约束变量。

在测试任务流中，测试数据的长度和测试活动的状态等都可能是选择分支的依据，这里以受约束变量来抽象这些判断依据。对于约束集 $COND$ 来说，其中的 $cond_i$ 可能是一个受约束变量，也可能是多个受约束变量的组合。这里借鉴数理逻辑中的概念，分别称之为原子谓词和复合谓词。如图 1 所示，如果 X 表示测试数据 X 的长度，X_1 表示 $X\leqslant 5$，X_2 表示 $X>5$；Y 表示测试资源的可用状态，Y_1 表示 $Y<1$，Y_2 表示 $Y\geqslant 1$，那么 $X_1 \wedge Y_1$ 表示测试数据 X 的长度小于等于 5 并且测试资源 Y 不可用。

定义 2（约束条件的全域）　任意 $cond_i(C) \in COND$，其中约束条件变量集 $C=\{X, Y, \cdots, Z\}$。使 $cond_i(C)$ 为真和为假的约束条件变量集 C 的个体域集分别为 D 和 $\neg D$，把 $D \cup \neg D$ 称为约束条件 $cond_i(C)$ 的全域，记为 $Q(cond_i(C))=D \cup \neg D$，代表了变量集 $C=\{X, Y, \cdots, Z\}$ 的全域。

假设 $COND=\{X_1 \wedge Y_1, X_2 \wedge Y_1, X_3 \wedge Y_1\}$，对约束条件 $X_1 \wedge Y_1$ 个体域集 $D=\{X_1 \wedge Y_1\}$，有 $\neg D=\neg X_1 \vee \neg Y_1=\{X_2 \wedge Y_1, X_3 \wedge Y_1, X_1 \wedge Y_2, X_2 \wedge Y_2, X_3 \wedge Y_2\}$，根据本文给出的定义，约束条件 1 的全域 $Q(cond_1(C))=D \cup \neg D=\{X_1 \wedge Y_1, X_2 \wedge Y_1, X_3 \wedge Y_1, X_1 \wedge Y_2, X_2 \wedge Y_2, X_3 \wedge Y_2\}$。

定义 3（约束条件集的完整性）　在约束条件集 $COND=\{cond_1, cond_2, cond_3, \cdots, cond_n\}$ 中，$cond_i(C) \in COND$，其中约束条件变量集 $C=\{X, Y, \cdots, P, \cdots, Z\}$，使 $cond_i(C)$ 为真的约束条件变量集 C 的个体域集为 D_i，使其它 $n-1$ 个约束条件 $cond_1(C), \cdots, cond_{i-1}(C), cond_i(C), \cdots, cond_n(C)$ 为真的约束条件变量集 C 的个体域集为 $D_1 \cdots, D_{i-1}, D_i, \cdots, D_n$，如果任意 $Q(cond_i(C))=D_1 \cup D_2 \cdots \cup D_n$，那么称约束条件集在约束条件变量集 C 上具有完整性。

证明：首先需要证明约束条件的受约束变量拥有同一个受约束变量集 C，然后证明这些约束条件的个体域集的并集代表了约束条件集 $COND$ 的完整性。

1°假设存在约束条件 $cond_i(C_i) \in COND$，其中约束条件变量集 $C_i=\{X, Y, \cdots, Z\}$，受约束变量 P 没有出现在变量集中，也就是说 P 变量的值不影响 $cond_i$ 的真值，那么会有 $cond_i(C_i)=cond_i(C)$。假设约束条件变量集 $C=\{X, Y, \cdots, P, \cdots, Z\}$ 是变量最全的集合，那么可以把所有的约束条件变量集统一成 $C=\{X, Y, \cdots, P, \cdots, Z\}$，因此约束条件的变量集拥有同一个受约束变量集 C。

2°证明该逆否命题成立，即证明如果约束条件集在约束条件变量集 C 上不具有完整性，那么至少存在一个 $Q(cond_i(C)) \neq D_1 \cup D_2 \cup \cdots \cup D_n$。由前面分析知道不具有完整性的情况有两种，一种是分支遗漏，另一种是分支冗余，当存在分支遗漏时，假设只是 $cond_{n+1}(C)$ 被遗漏了（补上该条件就是完整的），那么 $COND=\{cond_1, cond_2, cond_3, \cdots, cond_n\}$，存在 $Q(cond_i(C))=(D_1 \cup D_2 \cdots \cup D_n \cup D_{n+1})$。再来考虑分支冗余的情况，假设 $cond_n(C)$ 是冗余的（去掉该条件就是完整的），那么 $COND=\{cond_1, cond_2, cond_3, \cdots, cond_n\}$，存在 $Q(cond_i(C))=(D_1 \cup D_2 \cdots \cup D_{n-1})$。由以上证明可知，该逆否命题成立，那么原命题得证。

如第 2 节中图 1 右半部分所示，$COND=\{X_1 \wedge Y_1, X_2 \wedge Y_1, X_3 \wedge Y_1\}$ 中约束条件个体域集分别为 D_1, D_2, D_3，$Q(cond_1(C))=D_1 \cup \neg D_1=\{X_1 \wedge Y_1, X_2 \wedge Y_1, X_3 \wedge Y_1, X_1 \wedge Y_2, X_2 \wedge Y_2, X_3 \wedge Y_2\}$；$Q(cond_2(C))=D_2 \cup \neg D_2=\{X_1 \wedge Y_1, X_2 \wedge Y_1, X_3 \wedge Y_1, X_1 \wedge Y_2, X_2 \wedge Y_2, X_3 \wedge Y_2\}$；$Q(cond_3(C))=D_3 \cup \neg D_3=\{X_1 \wedge Y_1, X_2 \wedge Y_1, X_3 \wedge Y_1, X_1 \wedge Y_2, X_2 \wedge Y_2, X_3 \wedge Y_2\}$。很明显，$D_1 \cup D_2 \cup D_3$ 并不等于约束条件的全域，说明该约束条件集并不具有完整性。

4　测试任务流或分支的完整性验证方法

基于第 2 节的或分支完整性分析和第 3 节的或分支完整

性定义，这一节尝试给出或分支的完整性验证方法。在给出验证方法之前，先介绍约束条件集完整性的相关性质。

4.1 相关性质

性质 1 测试任务流模型中具有完整性的约束条件集 $COND$ 中的约束变量具有顺序无关性。

证明：首先，显而易见，同一个约束变量的不同状态在 $COND$ 集中具有顺序无关性。其次，证明约束条件集中由多个约束变量组成的约束条件具有顺序无关性。假设满足完整性的约束条件集合 $COND=\{cond_1(X,Y,\cdots,P,\cdots,Z),cond_2(X,Y,\cdots,P,\cdots,Z),\cdots,cond_n(X,Y,\cdots,P,\cdots,Z)\}$ 有共同的约束变量集 $C=\{X,Y,\cdots,P,\cdots,Z\}$；而且由完整性可知，$Q(cond_1(C))=Q(cond_2(C))=\cdots=Q(cond_n(C))=D_1\cup D_2\cdots\cup D_n$，任意变换 $Q(cond_i(C))$ 及其对应 D_i 的顺序，对应的 $COND$ 集合做相应的顺序变换，约束条件集 $COND$ 的完整性并不会受到影响。

性质 2 约束条件集 $COND$ 可以表示成 $COND=\{c_i\mid c_i=X_{i1}\wedge Y_{i2}\wedge\cdots\wedge Z_{im}\}$，其中 $1\leqslant i\leqslant n$，$X_{ij},Y_{ij},\cdots,Z_{ij}$ 表示某个约束变量的某一状态（原子谓词）或者这些原子谓词的析取范式，$1\leqslant j\leqslant m$。

证明：在约束条件集中 $cond_i$ 可能是原子谓词、原子谓词组合而成的合取范式、原子谓词组合而成的析取范式、原子谓词组合而成的混合范式，而根据范式存在定理，这些都可以转化成 $c_i=X_{i1}\wedge Y_{i2}\wedge\cdots\wedge Z_{im}$ 形式的合取范式。

以构成约束条件的不同约束变量为行、约束变量的不同状态为列构成一个矩阵 $C=\begin{pmatrix}X_{11}&Y_{12}&\cdots&Z_{1m}\\X_{21}&Y_{22}&\cdots&Z_{2m}\\\vdots&\vdots&\vdots&\vdots\\X_{n1}&Y_{n2}&\cdots&Z_{nm}\end{pmatrix}$，其中 $cond_i=X_{i1}\wedge Y_{i2}\wedge\cdots\wedge Z_{im}$，$X_{11},X_{21},\cdots,X_{n1}$ 表示约束变量 X 的不同状态，其它约束变量同理。如果矩阵中某一位置出现了约束变量的某一状态，则将该位置记为 1；若没有出现任何约束变量的任何状态，则将该位置记为 0。将矩阵中各列值相加得到的和值记为 ω_i，i 表示约束变量名，最后得到约束变量权值集合 $W=\{\omega_x,\omega_y,\cdots,\omega_z\}$。

4.2 验证思想

构造一棵完整的判定树，将 $COND$ 集与之对比，如果一致，则 $COND$ 集合具有完整性；如果不一致，$COND$ 集合则不具有完整性。在构造判定树时，最简单的方法是穷举法，由于约束变量的顺序无关性，不需要考虑约束变量的出现顺序，将所有约束变量的所有状态穷举组合出来，即可构造一棵完整的判定树。这里借鉴哈夫曼的思想，根据约束变量出现次数的多少，少的在下，多的在上，自底向上构造一棵完整性判定树。判定树构造完成后，将从其根节点开始到叶节点的每条路径上的约束变量的状态（原子谓词）按 $c_i=X_{i1}\wedge Y_{i2}\wedge\cdots\wedge Z_{im}$ 的方式组合起来（此处的 i 可以代表不同的值），构成一个完整的 $COND'$ 集。对判定树进行相应的裁剪，保证完整性不变，使其形式上与 $COND$ 中的约束变量组合形式一样，在 $COND'$ 集中也做相应的处理。如果判定树裁剪组合成的 $COND'$ 集与 $COND$ 集完全一样，那么说明 $COND$ 集是完整的；如果 $COND'$ 集中存在约束条件与 $COND$ 集无法对应即 $COND'\subset COND$，说明或分支有冗余；如果 $COND$ 集中存在约束条件无法与 $COND'$ 对应即 $COND\subset COND'$，则说明或分支有遗漏。

4.3 验证方法

或分支的完整性验证方法如图 3 所示。

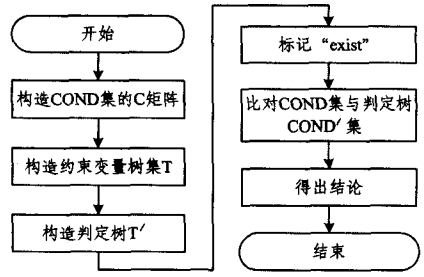

图 3 验证完整性流程

1）构造 $COND$ 集的 C 矩阵。根据提供的 $COND$ 集，一个约束条件为一行，将约束条件各约束变量的状态放入行中相应的位置，构造形如 C 的 $COND$ 集矩阵。将出现约束变量的地方标 1，没有的标 0，计算各列的值，得到约束变量权值集合 $W=\{\omega_x,\omega_y,\cdots,\omega_z\}$。

2）构造约束变量树集 $T=\{T_x,T_y,\cdots,T_z\}$。假设在构造 $COND$ 集的 C 矩阵中出现了 m 个约束变量，一个约束变量的所有不同的状态构成一棵树。其根节点为 T_i，i 位于 x 和 z 之间，共有 m 个约束变量，每条边表示约束变量的每一状态，叶节点为空，如图 4 所示。

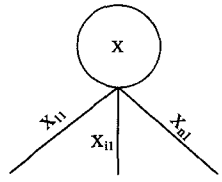

图 4 约束变量 X 树

3）构造判定树 T'。选取 W 中最小值 $w_i(1\leqslant i\leqslant m)$，且位于 x 和 z 之间，找出约束变量树集 T 中对应的树 T_i 放入 T'，在 W 集合和约束变量树集 T 中分别删去对应的元素 ω_i 和 T_i。继续选取 W 中的最小的 $w_j(1\leqslant j\leqslant m)$ 且位于 x 和 z 之间，找出对应的约束变量树集 T 中对应的树 T_j，以树 T_j 的根节点为根节点、树 T_j 的边为边，每条边末端以 T' 中的 T_i 子树作为该树的叶节点，形成一棵新树 T'。在 W 集合和约束变量树集 T 中分别删去对应的元素 ω_j 和 T_j。如此反复，直到 W 中元素被删除为空，此时判定树 T' 初步形成。

4）标记“exist”。查看判定树 T' 从根节点到到未端叶节点的路径上由边代表的约束变量的状态合取组成的 c_i'，如果在 $COND$ 集中存在 c_i 有 $c_i=c_i'$，则将 c_i 从 $COND$ 集中删去，并将该路径标记为“exist”；如果不存在，则将该路径标为“no”。将树中同一级别路径标完后，只有当一棵子树 $T_i\in\{T_x,T_y,\cdots,T_z\}$ 的所有边都被标为“no”，该子树才可以被裁剪，根据这一规则对判定树进行裁剪。裁剪后的树的未被标记的路径与 $COND$ 集合中剩余的 c_j 对比，重复此过程，直到 $COND$ 集合为空或者判定树不能再被裁剪，此时再将裁剪好的判定树从根节点到叶节点的所有路径的边合取组成的范式组成 $COND'$ 集合。

5)将原始 $COND$ 集合与 $COND'$ 集合进行比对。如果 $COND'=COND$，说明 COND 集是完整的；如果 $COND'\subset COND$，说明或分支有冗余；如果 $COND\subset COND'$，则说明或分支有遗漏。

4.4 验证方法的有效性证明

分两个步骤，首先证明步骤 3)中构造的判定树 T' 是完整的，然后证明裁剪的过程保留树的完整性，因此最后得到的判定树也是完整的，那么 $COND$ 集合与之进行对比得出的结论就是正确的，从而证明了该验证方法有效。

假设 $COND$ 集中有 m 个约束变量，每个约束变量的状态数分别为 $N_1,N_2,\cdots,N_m$。由数学排列原理和约束变量顺序无关性可知，从每个约束变量中随机不重复选取一个状态组成的排列组合中，共 $N_1\times N_2\times\cdots\times N_m$ 个组合组成的 $COND$ 集是完整的。本文提供的验证方法中由步骤 3)构成的判定树共有 $N_1\times N_2\times\cdots\times N_m$ 条不同的路径，与数学排列组合出来的完整集是一致的，因此该判定树 T' 是完整的。

假设判定树每一条由各边合取组成的路径为 $c_i=X_{i1}\wedge Y_{i2}\wedge\cdots\wedge Z_{in}$，当 c_i 为真时组成的域集为 D，记为 D_i，因此可以将判定树的全集表示为 $D_1\cup D_2\cdots\cup D_{(N_1\times N_2\times\cdots\times N_m)}$。假设约束变量 Z 共有 $Z_1,Z_2,\cdots,Z_{N_m}$ 种状态，那么初步的判定树中肯定存在这些路径：$X_i\wedge Y_j\wedge\cdots P_k\wedge Z_1$，$X_i\wedge Y_j\wedge\cdots P_k\wedge Z_2,\cdots,X_i\wedge Y_j\wedge\cdots P_k\wedge Z_{N_m}$，其中 $1\leqslant i\leqslant N_1$，$1\leqslant j\leqslant N_2$，$1\leqslant k\leqslant N_{m-1}$，这些路径的前 $m-1$ 个都是一样的，而最后约束变量 Z 的状态是完整而不同的。假设这些路径取真值对应的个体域集为 $D_1,D_2,\cdots,D_{N_m}$，如果某棵 Z 树被裁剪掉，那么存在一条路径 $X_i\wedge Y_j\wedge\cdots\wedge P_k$，其对应个体域集为 D'。现在来证明 $D_1\cup D_2\cup\cdots\cup D_{N_m}\Leftrightarrow D'$，如果这是成立的，那么 $D_1\cup D_2\cdots\cup D_{(N_1\times N_2\times\cdots\times N_m)}\Leftrightarrow D'\cup D_{(N_m+1)}\cdots\cup D_{(N_1\times N_2\times\cdots\times N_m)}$，也就意味着裁剪后，其完整性还得到保留。先证充分性，如果 $D_1\cup D_2\cup\cdots\cup D_{N_m}$ 成立，即 $D_1\cup D_2\cup\cdots\cup D_{N_m}$ 中一个或多个成立，当 $X_i\wedge Y_j\wedge\cdots P_k\wedge Z_1,X_i\wedge Y_j\wedge\cdots P_k\wedge Z_2,\cdots,X_i\wedge Y_j\wedge\cdots P_k\wedge Z_{N_m}$ 中的一个或多个取值为真时，对应的 $D_i(1\leqslant i\leqslant N_m)$ 成立，不管哪个合取范式取值为真，它们共有的原子谓词 $X_i\wedge Y_j\wedge\cdots\wedge P_k$ 都是真值，此时 D' 也成立。因此充分性得证。再证必要性，如果 D' 成立，那么 $X_i\wedge Y_j\wedge\cdots\wedge P_k$ 取值为真，因为约束变量 Z 共有 $Z_1,Z_2,\cdots,Z_{N_m}$ 种状态，因此 $Z_1,Z_2,\cdots,Z_{N_m}$ 必有一个为真值，那么 $X_i\wedge Y_j\wedge\cdots P_k\wedge Z_1,X_i\wedge Y_j\wedge\cdots P_k\wedge Z_2,\cdots,X_i\wedge Y_j\wedge\cdots P_k\wedge Z_{N_m}$ 就必有一个取值为真，对应的 $D_1,D_2,\cdots,D_{N_m}$ 必有一个成立，此时 $D_1,D_2,\cdots,D_{N_m}$ 成立，因此必要性得证。

通过以上证明可以知道，裁剪后的树也是一棵具有或分支完整性的树，因此将 $COND$ 集与之进行比较，就可以知道 $COND$ 集是否完整。

5 实例

假设测试任务流模型中有一个或分支由 3 个约束变量 X,Y,Z 来控制，约束变量 X 有 X_1,X_2,X_3 3 种状态，约束变量 Y 有 Y_1,Y_2 两种状态，约束变量 Z 有 Z_1,Z_2 两种状态。其在测试任务流模型中的形式如图 5 所示。

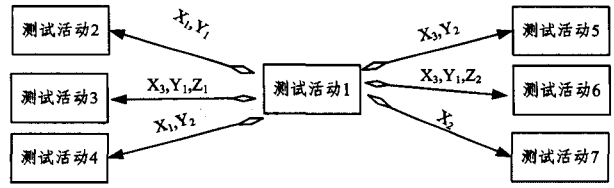

图 5 测试活动 1 的或分支

由图 5 可以得到该或分支的约束条件集 $COND=\{X_1\wedge Y_1,X_3\wedge Y_1\wedge Z_1,X_1\wedge Y_2,X_3\wedge Y_2,X_3\wedge Y_1\wedge Z_2,X_2\}$，下面通过上一节给出的或分支完整性验证方法来验证该或分支的完整性。

构造约束条件集矩阵 $C=\begin{pmatrix}X_1 & Y_1 & \\ X_2 & & \\ X_3 & Y_1 & Z_1\\ X_1 & Y_2 & \\ X_3 & Y_2 & \\ X_3 & Y_1 & Z_2\end{pmatrix}$，将存在约束变量的地方标为 1，不存在的地方标为 0，于是得到存在矩阵 $C'=\begin{pmatrix}1&1&0\\1&0&0\\1&1&1\\1&1&0\\1&1&0\\1&1&1\end{pmatrix}$，将矩阵的各列相加得到约束变量权值集合 $W=\{\omega_x=6,\omega_y=5,\omega_z=2\}$。

构造约束变量树集 $T=\{T_x,T_y,T_z\}$，各子树如图 6 所示。

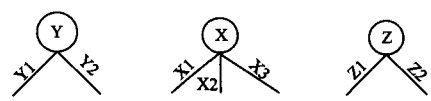

图 6 约束变量树集

构造判定树 T'。根据 W 集合中元素的大小，依次构造判定树 T'，如图 7 所示。

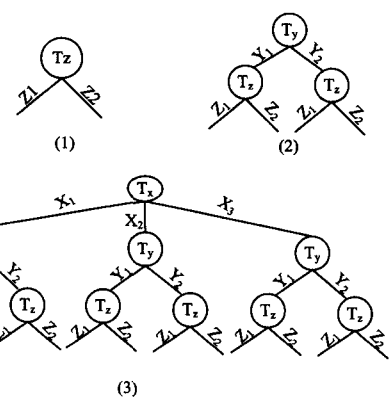

图 7 判定树 T' 的构造

标记“exist”，如图 8 所示。

由图 8 可以得到 $COND'=\{X_1\wedge Y_1,X_1\wedge Y_2,X_2,X_3\wedge Y_1\wedge Z_1,X_3\wedge Y_1\wedge Z_2,X_3\wedge Y_2\}$。由约束变量顺序无关性可知 $COND'=COND$，因此该 $COND$ 集是完整的，说明其对应的测试任务流中的或分支也是完整的。

（下转第 192 页）

[7] Rathfelder C, Klatt B, Sachs K. Modeling event-based communication in component-based software architectures for performance predictions [J]. Software & Systems Modeling, 2014, 13 (4): 1291-1317

[8] Koziolek A, Koziolek H, Reussner R. Peropteryx: automated application of tactics in multi-objective software architecture optimization[C]// Proceedings of the joint ACM SIGSOFT Conference — QoSA and ACM SIGSOFT Symposium — ISARCS on Quality of Software Architectures. Boulder, Colorado, USA: ACM, 2011: 33-42

[9] Smith C U, Williams L G. Software performance antipatterns [C]// Proceedings of the 2nd International Workshop on Software and Performance. New York, NY, USA: ACM, 2000: 127-136

[10] Smith C U, Williams L G. Performance solutions: a practical guide to creating responsive, scalable software [M]. Addison-Wesley, 2002

[11] Tribastone M. A fluid model for layered queueing networks[J]. IEEE Transactions on Software Engineering, 2013, 39(6): 744-756

[12] Xu J. Rule-based automatic software performance diagnosis and improvement [J]. Performance Evaluation, 2012, 69(11): 525-550

[13] Mcgregor J D, Bachmann F, Bass L, et al. Using arche in the classroom: One experience (CMU/SEI-2007-TN-001) [R]. Pittsburgh: Software Engineering Institute, Carnegie Mellon University, 2007

[14] Bondarev E, Chaudron M R V, de Kock E A. Exploring performance trade-offs of a JPEG decoder using the DeepCompass framework[C]// Proceedings of the 6th international workshop on software and performance, 2007. Buenos Aires, Argentina: ACM, 2007: 153-163

[15] Cortellessa V, Frittella L. A framework for automated generation of architectural feedback from software performance analysis[C]// Proceedings of Fourth European Performance Engineering Workshop (EPEW 2007). Berlin: Springer, 2007: 171-185

[16] Cortellessa V, Martens A, Reussner R, et al. A process to effectively identify "guilty" performance antipatterns [M]. Fundamental approaches to software engineering. Berlin Heidelberg: Springer, 2010: 368-382

[17] Trubiani C, Koziolek A. Detection and solution of software performance antipatterns in palladio architectural models[C]// Proceedings of the 2nd ACM/SPEC International Conference on Performance Engineering. New York, NY, USA: ACM, 2011: 19-30

[18] Cortellessa V, Di Marco A, Trubiani C. An approach for modeling and detecting software performance antipatterns based on first-order logics [J]. Software & Systems Modeling, 2014, 13 (1): 391-432

(上接第 183 页)

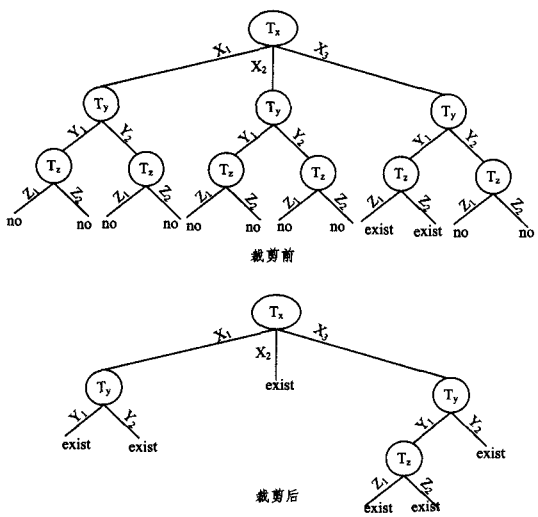

图 8 标记完的判定树

结束语 测试任务流模型的结构特性、行为特性和性能方面的分析是很重要且必要的，但是其中或分支的完整性的分析也是很有必要的。本文从分析或分支的完整性入手，给出其完整性定义，提供一种验证或分支完整性的判定树方法。该方法不依赖于具体的模型，适用性好；算法借鉴哈夫曼树的思想，有一定理论基础，易于编程实现。当然，该方法的前提是需要保证每一约束变量自身是完整的，这一点由人保证问题不大。本文的研究只是针对测试任务流模型中的或分支完整性，没有对与分支和循环等控制结构进行分析验证，这些在将来的研究中需要进一步的完善。

参 考 文 献

[1] IEEE Standards Coordinating Committee 20. IEEE Std 1671-2010, IEEE Standard for Automatic Test Markup Language (ATML) for Exchanging Automatic Test Equipment and Test Information via XML [S]. USA: IEEE-SA Standards Board, 2010

[2] IEEE Standards Coordinating Committee 20. IEEE Std 1671. 1-2009, IEEE Trial-Use Standard for Automatic Test Markup Language (ATML) for Exchanging Automatic Test Equipment and Test Information via XML: Exchanging Test Descriptions [S]. USA: IEEE-SA Standards Board, 2009

[3] Muth P, Wodtke D, Weissenfels J. Enterprise-Wide workflow management based on state and activity charts[OL]. http://paris. cs. uni-sb. de/public_html/papers/nato-wf: ps

[4] Scholz-Reiter B, Stichel E. Business process modeing[M]Berlin: Springer-Verlag, 1996

[5] 袁崇义. Petri 网原理与应用[M]. 北京：电子工业出版社，2005: 225-258
Yuan Cong-yi. The Theory and Application of Petri Nets[M]. Beijing: Publishing Company of Electric Industry, 2005: 225-258

[6] 罗海滨，范玉顺，吴澄. 工作流合理性验证中的事件平衡分析[J]. 软件学报，2002, 13(8): 1686-1691
Luo Hai-bin, Fan Yu-shun, Wu Cheng. Analysis of event balance in the verification of workflow soundness[J]. Journal of Software, 2002, 13(8): 1686-1691

[7] Ernst W M, Jeremias W. Results on Equivalence, Boundedness, Liveness, and Covering Problems of Bpp-Petri Nets[C]// Application and Theory of Petri Nets and Concurrency 2013. Milan, Italy Jose-Manuel Colom, 2013: 70-90

[8] 庞善臣，蒋昌俊. 一种基于不变量结构分解的工作流性能分析方法[J]. 计算机学报，2010, 33(5): 908-917
Pang Shan-chen, Jiang Chang-jun. Workflow Performance Analysis Based on Invariant Decomposition Algorithm [J]. Chinese Journal of Computer, 2010, 33(5): 908-917