一种适用于具有相互依赖基本事件和重复事件的动态故障树独立模块识别方法

张红林[1)] 张春元[1)] 刘 东[2)]

[1)](国防科学技术大学计算机学院 长沙 410073)

[2)](装备指挥技术学院国防科技重点实验室 北京 101416)

摘 要 为减小时间开销,动态故障树经常被模块化分解为独立的静态子树和独立的动态子树,然后分别使用二叉决策图和马尔卡夫模型求解;其中的一个关键问题便是识别具有相互依赖基本事件和重复事件的动态故障树中所有的独立模块和最小独立模块.文中提出了一个基于亲戚依赖关系的独立模块识别方法IIMKDR,该方法将故障树按照其事件之间的依赖关系转换为依赖树;基于面向对象思想,为依赖树及其节点构建对象;通过对各节点对象属性的分析,得出独立模块集并进而求出最小独立模块集.最后对IIMKDR方法进行了理论分析和实验验证,分析表明该方法适用于具有相互依赖基本事件和重复事件的动态故障树独立模块识别.与其它方法从多个角度所进行的实验对比显示,当故障树没有相互依赖基本事件和重复事件时,该方法的开销略高于其它方法;当故障树具有相互依赖基本事件和重复事件时,该方法能够正确识别其中所有的独立模块,而其它方法不具备这样的功能.

关键词 动态故障树;独立模块识别;相互依赖基本事件;重复事件;可靠性分析

中图法分类号 TP302 **DOI号**: 10.3724/SP.J.1016.2012.00229

An Identification Method of Independent Module Applying to Dynamic Fault Tree with Interdependent Basic Events and Repeated Events

ZHANG Hong-Lin[1)] ZHANG Chun-Yuan[1)] LIU Dong[2)]

[1)](*School of Computer, National University of Defense Technology, Changsha* 410073)

[2)](*Key Laboratory of National Defense Technology, Academy of Equipment Command & Technology, Beijing* 101416)

Abstract In order to reduce the time cost, dynamic fault tree is always modularized into independent static modules and independent dynamic modules, then solved by BDD and Markov model separately. One of the key questions is how to identify all the independent modules and minimum independent modules in the dynamic fault tree with interdependent basic events and repeated events. An identification method, called IIMKDR, is proposed based on kinship dependency relation. IIMKDR converts fault tree to dependent tree according to the dependence relations among the events. Both dependent tree and nodes are then constructed based on the object-oriented idea, and all the independent modules and minimum independent modules are identified through the analysis of the properties of nodes. Finally theoretical analysis and experimental verification are carried out on this method. The analysis indicates that this method can be applied to the dynamic fault tree with interdependent basic events and repeated events. The comparisons with other methods from various aspects show that the time cost of this method is slightly high than other methods given without interdependent basic events or repeated events, and this method can exact-

收稿日期:2009-12-14;最终修改稿收到日期:2011-04-12.本课题得到国家自然科学基金(60904082)、国家预研项目(51320010201)资助.
张红林,男,1977年生,博士研究生,主要研究方向为计算机系统可靠性分析、容错技术.E-mail: honglin518@gmail.com. **张春元**,男,1964年生,教授,博士生导师,研究领域为计算机体系结构、高性能计算. **刘 东**,男,1981年生,博士,讲师,主要研究方向为计算机系统可靠性分析、容错技术.

ly identify all the independent modules while other methods cannot do given with interdependent basic events and repeated events.

Keywords dynamic fault tree; identification of independent module; interdependent basic events; repeated events; reliability analysis

1 引 言

故障树(Fault Tree,FT)被广泛应用在系统可靠性分析中,为系统的失效行为提供了一个简单直观的描述方式.传统的静态故障树(Static Fault Tree,SFT)通过对各基本事件失效的布尔组合逻辑可得到系统的失效函数.静态故障树一般包括AND、OR、VOTE等布尔逻辑门,通常可以使用二值决策图(Binary Decision Diagram,BDD)方法求解.随着实践中对可靠性要求的提高以及一些可靠性设计技术的应用,出现了越来越多的动态系统,这些系统中各元件之间具有一定的动态顺序依赖关系.静态故障树无法对动态系统进行建模分析,文献[1]通过引入动态逻辑门将故障树扩展为动态故障树(Dynamic Fault Tree,DFT),从而使其具有对动态系统进行建模分析的能力.由于动态故障树中各事件之间具有时序依赖关系,因此无法使用数值组合逻辑进行求解,一般使用马尔卡夫模型进行求解[2].然而马尔卡夫模型具有固有的局限性,其所涉及到的系统状态数与系统基本事件数呈指数级关系,当系统的规模比较大时(实际应用的系统大都属于此类),为马尔卡夫模型建立状态转移矩阵将变成一个不可完成的任务,即所谓的状态空间爆炸问题.状态空间爆炸问题大大限制了马尔卡夫模型的应用.

为解决状态空间爆炸问题,业界进行了许多研究.文献[3]给出了一种数值分析方法(Digital Numerical Analysis,DNA)来对动态故障树进行分析,该方法具有简单、直观、快速和准确的优点.针对现在许多系统包含大量冗余和备份部件,这些部件或模块可靠性特征完全相同,文献[4]提出了一种参数化故障树(Parametric Fault Tree,PFT)模型,利用BDD方法的简单快捷特性,给出了一种参数化二叉决策图(Parametric BDD,PBDD)方法.文献[5]使用一个混合贝叶斯网络框架来分析求解动态故障树,该框架并不需要数值积分和模拟.文献[6]提出了一种对动态故障树优先与门进行定量分析的方法.文献[2]将马尔卡夫模型中的状态转移图分解为几个状态转移链,根据每个链的长度获得该链的概率公式.文献[7]把动态逻辑门转化为相应的动态贝叶斯网络,以计算节点失效概率和进行重要度分析.

以上方法都需要识别出故障树中的独立模块、尤其是最小独立模块;模块化[8]是解决状态空间爆炸问题最根本的途径,其通过将整个故障树分解为多个独立模块,静态模块使用数值组合或BDD方法进行求解,动态模块使用马尔卡夫模型进行求解,最后再综合求出故障树顶层的可靠性参数,从而将求解时间开销从指数级降低为多项式级.另外,包括灵敏性分析在内的许多可靠性分析领域也依赖于故障树中独立模块的识别.故障树中独立模块的识别方法研究在可靠性分析领域具有极其重要的意义.

静态故障树仅包含表征组合逻辑关系的静态逻辑门,文献[9]给出了一个识别静态故障树中独立模块的方法,它对故障树进行一次深度遍历,并为每个节点标注首次访问时间和末次访问时间,若某个节点的所有子节点的最先首次访问时间和最后末次访问时间分别大于该节点的首次访问时间和小于该节点的末次访问时间,则该节点及其子孙节点构成一个独立模块.使用深度遍历算法,其计算复杂度为$O(N)$,其中N为故障树的节点个数.该方法的缺点有:只能应用于静态故障树,在遍历过程中,一旦发现动态逻辑门就会停止遍历;故障树中各基本事件相互独立,不含重复事件.文献[10]在文献[9]的基础上,提出了故障树中含有相互依赖基本事件的模块识别方法.它为每个事件建立依赖信息,并根据故障树的结构使依赖信息在各事件中传递,各事件的依赖信息结合访问时间确定故障树的独立模块.其缺点在于:若故障树含有重复事件或动态逻辑门,则该方法仍然无能为力.该方法本质上是属于构建独立模块,而不是识别独立模块,而如何构建仅仅通过一个实例说明,并未形成一个严谨的形式化算法.

动态故障树不仅包含表征组合逻辑关系的静态逻辑门,还包含表征顺序依赖关系的动态逻辑门,另外动态故障树可能还含有重复事件,基本事件间可能还存在依赖关系,识别动态故障树中的独立模块因此变得非常困难.文献[11-12]在文献[9]的基础上,虽然提出了一个适用于动态故障树的独立模块识别算法,但其并未考虑动态故障树中包含相互依

赖基本事件和重复事件的情形，无法在这种情形下应用.

本文在以上研究的基础上，提出了一个适用于动态故障树的基于亲戚依赖关系的独立模块识别方法(Identification of Independent Module based on Kinship Dependency Relation，IIMKDR)，其与以上研究不同之处在于：将动态故障树转化为依赖树，依赖树只包含事件之间的依赖信息，去除了故障树中各逻辑门的逻辑含义；改进深度优先遍历算法，使其适用于含有重复事件的情况；为每个节点加入祖先集属性和依赖集属性，当进行深度优先遍历时填充祖先集属性，将所有基本事件划分为不同的依赖类型，根据各基本事件的祖先集属性更新所有节点的依赖集属性. 本方法抛弃了以前研究中以深度遍历访问顺序进行独立模块识别的思想，改以各节点的祖先集属性和依赖集属性进行识别，其与相互依赖基本事件、重复事件、动态逻辑门无关，适用于具有相互依赖基本事件和重复事件的动态故障树. 为了便于说明，在不引起混淆的情况下，后文对于文献[9-11]所给出的方法与利用其方法得出的数据分别用其作者姓氏 Dutuit、Sun 和 Huang 标示.

本文第 2 节介绍本文用到的基本概念；第 3 节简要概括 IIMKDR 方法的识别流程；第 4～6 节详细描述识别流程的各个步骤，分别是预处理、构建依赖树、识别等；第 7 节对 IIMKDR 方法进行详细的时间复杂性分析，并与其它方法在性能上进行理论分析和对比；第 8 节分别使用实际案例和仿真模拟对 IIMKDR 方法从多个角度进行验证，并将 IIMKDR 方法和其它方法的实验结果进行对比分析；最后得出本文的结论.

2　基本概念

动态故障树的构建是在静态故障树的基础上，通过引入多个动态逻辑门使得故障树具有了描述动态行为的能力. 本节首先介绍各动态逻辑门，然后阐述本文所提出的相关概念.

2.1　动态逻辑门

(1) PAND 门

优先与门(Priority-AND Gate，PAND)是 AND 门的拓展，它在 AND 门的基础上增加一个附加条件，这个条件规定了输入事件的发生次序. 例如，对于具有一个输出事件和两个输入事件(A 和 B)的 PAND 门，当且仅当下列两个条件同时满足时，输出事件发生：

① 事件 A 和 B 都发生；

② 事件 A 先于事件 B 发生.

PAND 门的图形符号如图 1(a)所示. 如果 A 或 B 没有发生或者 B 在 A 之前发生，则输出事件不会发生.

(2) WSP 门

温备门(Warm-Spare Gate，WSP)具有一个初始输入和若干替补输入. 初始输入是指在系统开始工作时就处于工作状态的部件，替补输入作为温储备，在工作部件失效前，替补处于温备状态，工作部件失效后，逐个依次替补. WSP 门具有一个输出事件，仅当所有输入事件(初始输入和替补输入)发生后，输出事件才会发生. WSP 门的图形符号如图 1(b)所示.

(3) FDEP 门

功能相关门(Functional-Dependency Gate，FDEP)由以下 3 种事件构成：

① 触发输入事件. 触发输入事件可以是一个基本事件，也可以是故障树中其它门的输出；

② 非相关输出事件. 主要反映触发事件的状态；

③ 若干相关基本事件. 相关基本事件在功能上依赖于触发事件，当触发事件发生时，相关基本事件强制发生.

FDEP 门的图形符号如图 1(c)所示. FDEP 门的特征是：当触发事件发生时，直接产生输出，而所有相关事件随即成为不可达或无法使用. FDEP 门的非相关输出事件并不对 DFT 中的其它结构产生影响，FDEP 门主要通过约束相关基本事件的行为达到控制系统工作过程的目的.

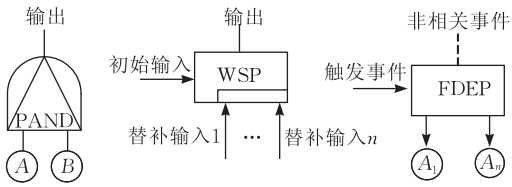


图 1　部分动态逻辑门示意图

2.2　其它相关概念

(1) 重复事件

若某个事件同时是两个或两个以上逻辑门的输入事件，则该事件为重复事件.

(2) 依赖关系

有两个事件 a、b，若 a 的失效影响 b 的失效概率或 b 的失效影响 a 的失效概率，则称 a 和 b 具有依赖关系.

基本事件间相互依赖的原因来自于基本事件间共因故障的存在. 设 $p(a)$ 和 $p(b)$ 分别表示基本事

件 a 和 b 的失效概率，$p(a|b)$ 表示在 b 失效的条件下 a 的失效概率，则 a 和 b 相互依赖用数学公式可以表示为 $p(a|b)\neq p(a)$ 或 $p(b|a)\neq p(b)$.

(3) 依赖类型

共因故障可以抽象为依赖类型，若某个共因故障 i 影响事件 a 的失效概率，则称 a 依赖于依赖类型 i.

(4) 节点

称 A 为节点，当且仅当 A 具有以下属性：祖先集 $ancestor\_set$，父亲集 $parent\_set$，$parent\_set\subseteq ancestor\_set$，依赖类型集(简称依赖集)$depend\_set$，遍历深度 $depth$，标号 ID.

考虑到故障树中可能含有重复事件，给节点定义属性 $ancestor\_set$ 后，便可从一个节点直接找到该节点的所有祖先节点. 属性 $depend\_set$ 为节点的依赖类型集合，依赖类型由基本事件之间的依赖关系给定. 属性 $depth$ 指的是构建依赖树时，该节点的遍历深度，使用该属性以便从独立模块集中找出最小独立模块集. 属性 ID 指的是该节点的代号，实际上本文在算法实现中，将节点的深度优先遍历次序赋予属性 ID.

顶节点的父亲集属性为空；中间节点的父亲集属性不为空，同时还属于其它某个节点的父亲集；叶节点不属于任何其它节点的父亲集.

(5) 直系血亲关系

两个节点 a、b，若 $a\in b.ancestor\_set$ 或 $b\in a.ancestor\_set$，则称 a 和 b 具有直系血亲关系.

(6) 亲戚依赖关系

两个节点 a、b，若 $a.depend\_set\cap b.depend\_set\neq\varnothing$，则称 a 和 b 具有亲戚依赖关系.

(7) 依赖树

依赖树具有与其对应故障树完全等价的依赖关系，含有一个顶节点、一个中间节点集合、一个叶节点集合.

3 识别流程

IIMKDR 方法包括依赖树构建算法、独立模块识别算法、最小独立模块识别算法等，这些算法会在后面详细阐述. IIMKDR 方法的识别流程可以分为以下几个步骤：

(1) 故障树预处理

负责对故障树进行一些预先处理，比如化简、变形等，以方便后续处理.

(2) 构建依赖树

动态故障树含有不同类型的逻辑门，各逻辑门的逻辑意义互不相同；然而在识别独立模块时，关注的是各事件之间是否相互依赖，并不关心各事件之间的逻辑关系；因此可从动态故障树中抽取各事件之间的依赖关系，剔除各事件之间的逻辑关系，构建依赖树. 依赖树完全保持了对应动态故障树的依赖信息，动态故障树的独立模块识别可转化为依赖树的独立模块识别.

在构建依赖树时填充各节点依赖集属性之外的所有其它相关属性，包括祖先集属性、父亲集、遍历深度等.

(3) 独立模块识别

叶节点的依赖集属性可从系统基本元件间的依赖关系获得，根据叶节点的依赖集属性、祖先集属性，填充其它各节点的依赖集属性. 然后利用各节点的依赖集属性、祖先集属性，识别出独立模块集和最小独立模块集.

4 故障树预处理

故障树结构的预处理旨在不改变其代表的逻辑函数情况下，将其形式简化，删减不必要的逻辑门和事件，以便后续处理. 本文提出的预处理方法包括以下阶段：

(1) 收缩

串接的同类型逻辑门可以收缩为一个逻辑门. 这些逻辑门包括 AND 门、OR 门、PAND 门和 SPARE 门等，如图 2 所示.

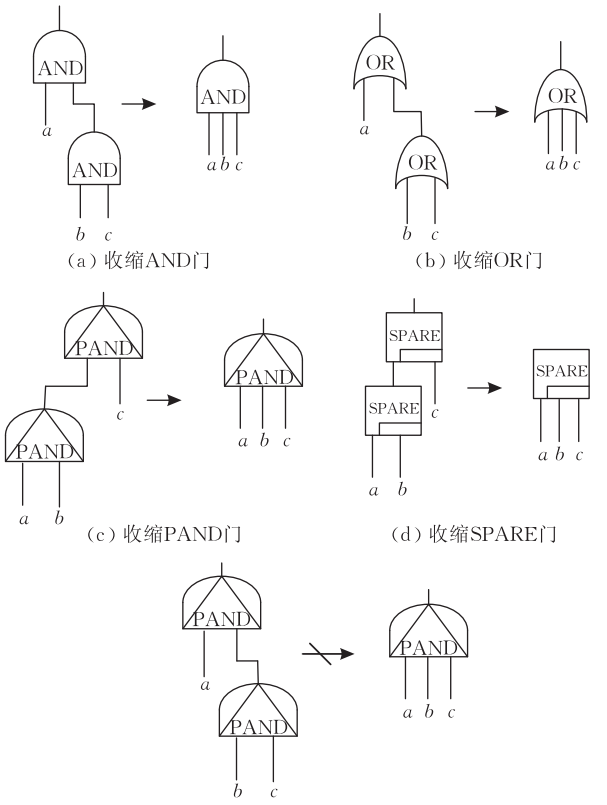
同类型串接的静态逻辑门都可以收缩为一个静态逻辑门，如图 2(a)和图 2(b)所示. 这样的结论对于动态逻辑门则不一定成立，如图 2(e)所示，PAND 门要求输入的事件按照规定的顺序失效才会导致输出失效，假如事件 a、b、c 的失效顺序为 $b\rightarrow a\rightarrow c$，则左边会导致输出失效，而右边则不会导致输出失效；对于图 2(c)和图 2(d)两种形式，收缩结论还是成立的. 因此同类型串接的动态逻辑门是否能够进行收缩需要进行专门的分析.

(2) 结合

故障树中符合图 3 和图 4 所示结构类型的(包括但并不限于)，可以进行结合处理，以消除一个逻辑门，同时尽可能地消除重复事件，简化后续处理. 需要注意的是，图 4(b)并不能成立，理由同上文收缩阶段.

(3) 消去

根据故障树所表达的静态或动态逻辑意义，消



(a) 收缩AND门　(b) 收缩OR门

(c) 收缩PAND门　(d) 收缩SPARE门

(e) 此PAND门结构不能收缩

图 2　故障树预处理收缩阶段

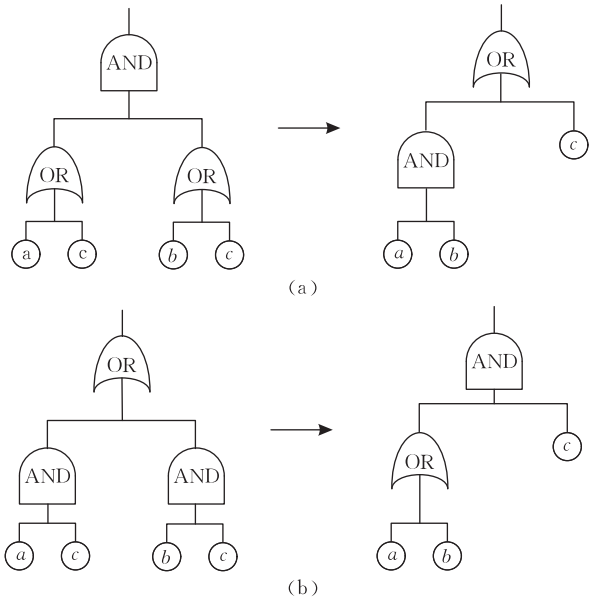


图 3　故障树预处理结合阶段(仅含静态逻辑门)

去一些冗余的逻辑门或事件,以简化处理. 比如符合图 5 所示结构类型的(包括但并不限于),便可以进行消去处理.

(4) 宏事件

始终成组以固定逻辑出现的基本事件可以看做是一个宏事件,在后续处理中,以此宏事件代替该组基本事件,减少事件数目.

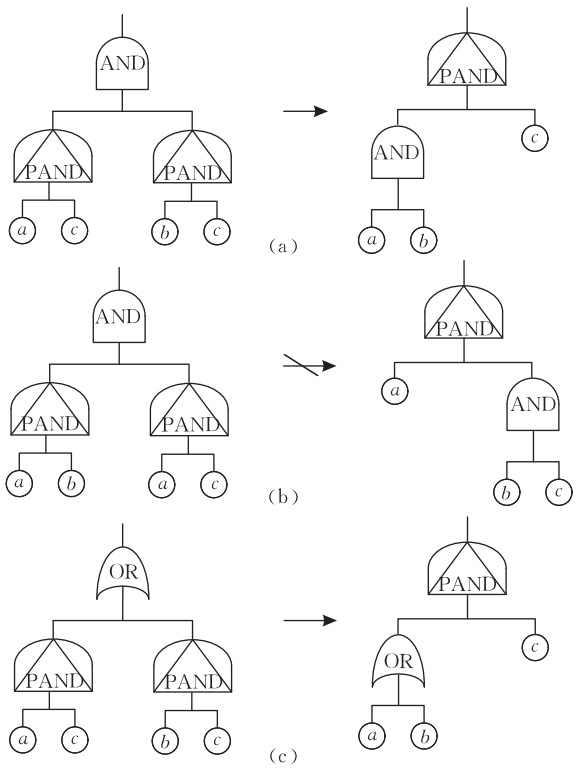


图 4　故障树预处理结合阶段(含动态逻辑门)

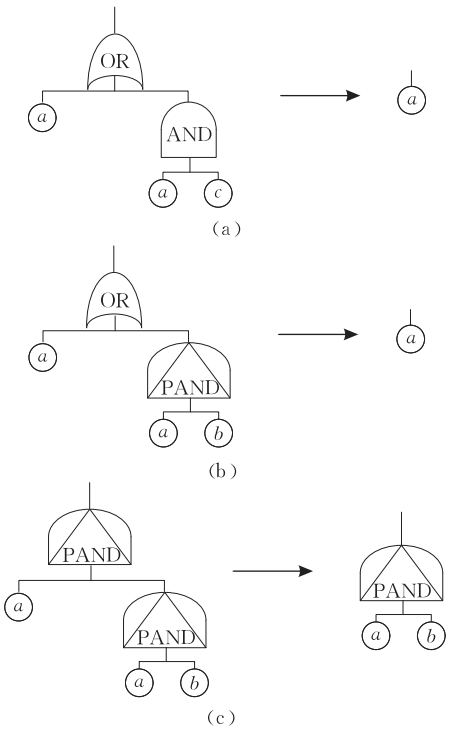


图 5　故障树预处理消去阶段

(5) FDEP 门

FDEP 决定了触发输入事件与其它输入事件之间的依赖关系,因其没有输出事件外联,此 FDEP 等效于多个 OR 门串接,如图 6 所示. 在故障树中可分别以 $\underline{I}_1$、$\underline{I}_2$ 代替 I_1、I_2.

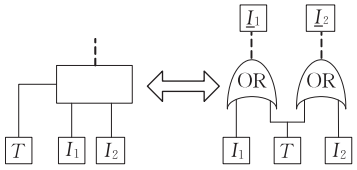


图 6 FDEP 门预处理

5 依赖树构建

故障树刻画了事件之间的各种关系，包括静态组合逻辑关系和动态时序关系，但是当进行独立模块识别时，关注的仅仅是节点之间是否具有依赖关系. 为了提高识别效率，有必要从故障树中提取节点之间的依赖关系构建依赖树，之后进行的独立模块识别都是建立在依赖树的基础上. 基于面向对象思想为节点和依赖树构建对象，节点对象具有以下属性：祖先集 *ancestor_set*，父亲集 *parent_set*，依赖集 *depend_set*，遍历深度 *depth*，标号 *ID*；依赖树对象具有以下属性：顶节点 s，中间节点集 *MNS*，叶节点集 *LNS*.

通过对故障树进行深度优先遍历来构建依赖树. 算法 1 为依赖树构建算法 DTC：前 2 行构建着色数组 *color*；第 4 行的 *count* 变量表征各节点的遍历次序，此值将赋给相应节点的 *ID* 属性；依赖树节点构建算法 CreateDTNode，如算法 2 所示，根据依赖树节点的对象大小，在内存中分配相应的空间，并将所有属性值设为空集或 0，这些属性值在后续遍历时进行填充；在开始遍历前，依赖树的 *LNS* 和 *MNS* 属性为空，然后调用深度遍历算法 DFS_VISIT.

算法 1. 依赖树构建算法 DTC.

DTC(FT,DT)
功能：从故障树构建依赖树对象
输入：故障树 $FT(V,E)$，s 为顶事件；
输出：依赖树对象 DT；
1. for each event $u \in V$
2. do $color[u] \leftarrow$ WHITE;
3. $u \leftarrow s$;
4. $count \leftarrow 0$;
5. $time \leftarrow 0$;
6. $u_0 \leftarrow$ CreateDTNode();
7. $DT.s \leftarrow u_0$;
8. $DT.LNS \leftarrow \varnothing$; $DT.MNS \leftarrow \varnothing$;
9. DFS_VISIT(u, u_0);
10. return DT.

算法 2. 依赖树节点构建函数 CreateDTNode.

CreateDTNode()
功能：构建依赖树节点
输入：无
输出：依赖树节点
1. $u_0 = new\ DTNode()$;
2. $u_0.parent\_set \leftarrow \varnothing$;
3. $u_0.ID \leftarrow 0$;
4. $u_0.depend\_set \leftarrow \varnothing$;
5. $u_0.depth \leftarrow 0$.

算法 3 深度优先遍历函数 DFS_VISIT 为一个递归算法，对故障树从事件 u 开始遍历，并相应从依赖树中节点 u_0 开始构建依赖树节点，填写各节点的有关属性值. 在遍历过程中，通过对事件进行着色来表示事件的状态. 开始时，每个事件均为白色，搜索中被发现时即置为灰色.

每次调用 DFS_VISIT(u,u_0)时，事件 u 和节点 u_0 分别成为故障树和依赖树中深度优先森林中一颗新树的根，事件 u 开始为白色，第 2 行置 u 为灰色，表示开始对事件 u 进行遍历；当事件 u 的邻接表 adj 为空时，表示该事件为故障树的基本事件，第 7 行将相应节点加入依赖树的叶节点集合；第 9～25 行检查和 u 相邻接的每个节点 v，如果 v 为白色节点，表示该节点还未被访问，需构建相对应的依赖树节点 v_0，设置 v_0 相关属性，并从节点 v 递归遍历；否则，仅需填充 v_0 相关属性并从节点 v 递归遍历即可. 传统的深度遍历算法没有第 20～25 行代码，为了支持含有重复事件的故障树，特加入这 5 行代码：即如果 v 并非白色，则表示该事件已被访问且 v 有两个或多个父事件，相应的依赖树节点 v_0 也已被创建，但需将 v_0 的属性 *parent_set* 和 *ancestor_set* 进行相应的更新，其子孙节点的相应属性也应进行相应的更新，故仍需递归遍历事件 v. 第 18 行为故障树事件和相应的依赖树节点映射表，第 21 行根据故障树事件获取依赖树节点对象.

算法 3. 深度优先遍历算法 DFS_VISIT.

DFS_VISIT(u,u_0)
功能：对故障树进行遍历，并相应构建依赖树节点
输入：故障树事件 u，邻接矩阵 adj，依赖树节点 $u0$
输出：从节点 $u0$ 开始构建的各依赖树节点
1. if ($color[u]==$WHITE)
2. $color[u] \leftarrow$ GRAY;
3. $time \leftarrow time+1$;
4. $d[u] \leftarrow time$
5. if $adj[u]==$NULL
6. then
7. $DT.LNS \leftarrow DT.LNS \cup \{u_0\}$;
8. else
9. for each $v \in adj[u]$
10. do if $color[v]=$WHITE

```
11.        then
12.          v0←CreateDTNode();
             DT.MNS←DT.MNS∪{v0};
13.          count←count+1;
14.          v0.parent_set←{u0}
15.          v0.ancestor_set←v0.ancestor_set∪
                              u0.ancestor_set∪{u0};
16.          v0.ID←count;
17.          v0.depth←u0.depth+1;
18.          FTMapDT(v, v0);
19.          DFS_VISIT(v, v0);
20.        else
21.          v0←MapToDTNode(v);
22.          v0.parent_set← v0.parent_set∪{u0}
23.          v0.ancestor_set←v0.ancestor_set∪
                              u0.ancestor_set∪{u0};
24.          time←time+2;
25.          DFS_VISIT(v,v0);
26.  if(color[u]==GRAY)
27.    color[u]←BLACK;
28.    time←time+1.
```

图 7 为一个简单故障树示例，图 8 说明了函数 DFS_VISIT 在图 7 上的工作过程. 执行完毕后，每个节点除去依赖集属性外其它属性(图 8 仅显示 *parent_set* 和 *ancestor_set* 属性)都已经得到填充，并且得到了依赖树的叶节点集合. 各节点的依赖集属性在独立模块识别算法中进行填充.

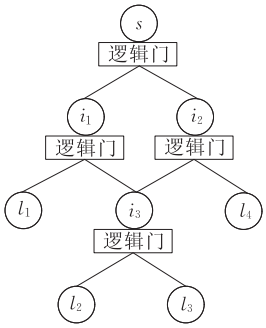


图 7　简单故障树示例

6　独立模块识别

随着故障树研究的深入，模块的定义也相应的发生变化. 文献[13]将模块定义为至少两个基本事件的集合，这些基本事件向上可到达同一逻辑门，并且必须通过此门才能达到顶事件，故障树的所有其它基本事件向上均不能到达该逻辑门；Dutuit 将模块定义为一个内部节点，其基本事件不会出现在故障树的任何其它地方. 以上两个定义适用于基本事件相互独立的故障树.

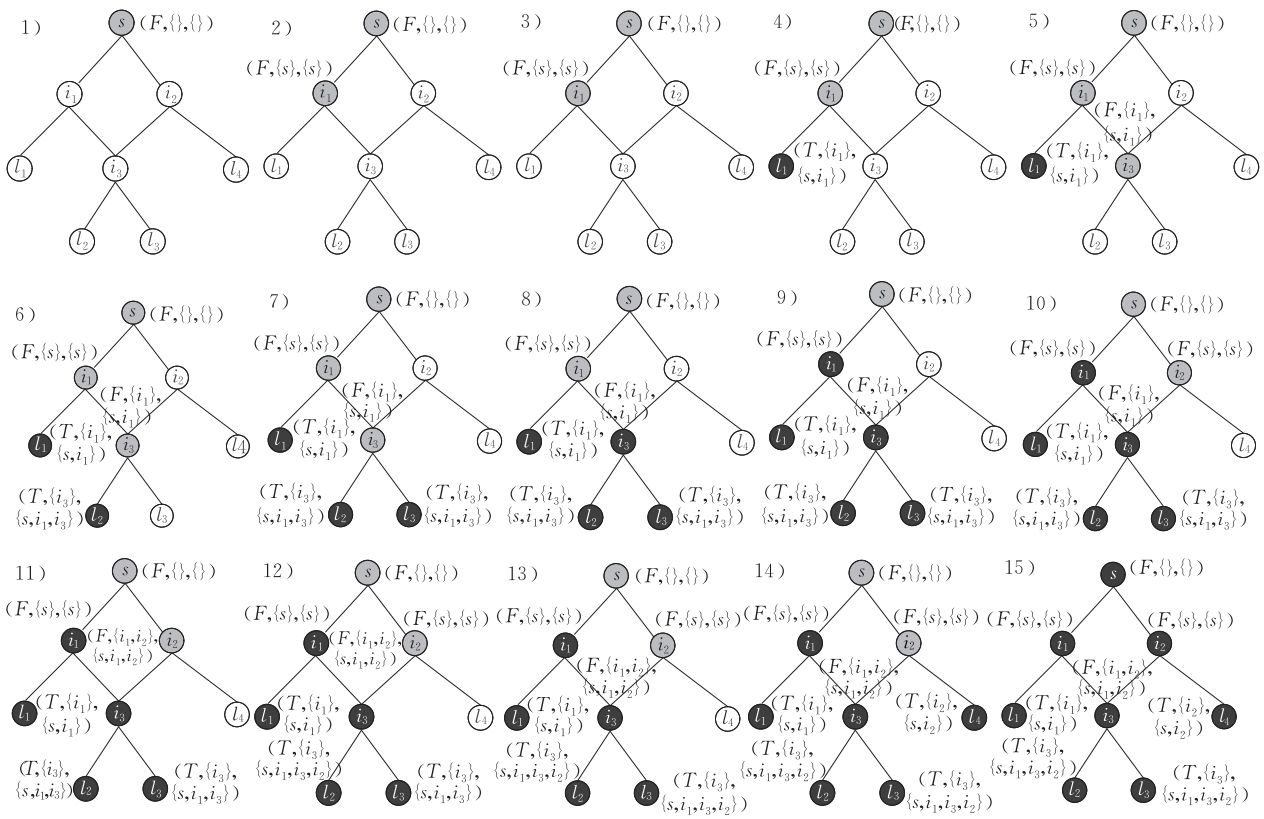


图 8　深度优先遍历函数 DFS_VISIT 在故障树上的工作过程. 每个事件 u 旁有三元组向量(v_1, v_2, v_3)，其中 v_1 表示对应故障树事件 u 的依赖树节点是否为叶节点，值 T 为是，值 F 为否；$v_2 = parent\_set[u_0]$，表示对应故障树事件 u 的依赖树节点 u_0 的父亲集；$v_3 = ancestor\_set[u_0]$，表示对应故障树事件 u 的依赖树节点 u_0 的祖先集.

无论模块如何定义,其基本思想是一脉相承的:将故障树分解成互不相关的部分分别进行处理,然后再对结果综合分析,从而降低故障树求解的代价.考虑到动态故障树所具有的动态依赖特性,本节给出一种更通用的独立模块定义和相应的独立模块识别算法,其中 $events(x)$ 为以 x 和所有 x 的子孙事件所组成子树的基本事件集合.

定义 1. 设 u 和 v 是故障树 FT 的两个事件,称 u 和 v 相互独立且记作 $u \perp v$,当且仅当:$\forall u' \in events(u)$,$\forall v' \in events(v)$,则 $prob(u'v') = prob(u') \times prob(v')$,其中 $prob(x)$ 为 x 的失效概率.

定义 2. 设 u 为故障树 FT 的一个中间事件,称 u 为 FT 的一个独立模块当且仅当任给 FT 的一个事件 v,必有下列一式成立:

(1) $events(u) \subseteq events(v)$;

(2) $events(v) \subseteq events(u)$;

(3) $u \perp v$.

定义 3. 设 L 是故障树 FT 的基本事件集,L_i $(i=1,2,\cdots,n)$ 是 L 的完全不相交子集,而且满足以下两个条件:

(i) $\forall u, v \in L_i$,则 $prob(uv) \neq prob(u) \times prob(v)$;

(ii) $\forall u \in L_i$,$\forall v \in L_j$ 且 $i \neq j$,则 $prob(uv) = prob(u) \times prob(v)$,

则

(1) 称 $L_1, L_2, \cdots, L_n$ 是 L 的完全独立划分;

(2) 若 $u \in L_i$,则称 u 依赖于依赖类型 i;

(3) v 是 FT 的一个事件,若 $\exists u \in L_i$ 且 $u \not\perp v$,则称 v 依赖于依赖类型 i;

(4) v 是 FT 的一个事件,$\phi = \{p_1, p_2, \cdots, p_m\}$ 是依赖类型集,若 $\forall i \in \phi$ 都有 v 依赖于类型 i,并且 $\forall j \notin \phi$ 都有 v 不依赖于依赖类型 j,则称 ϕ 为 v 的完全依赖集.

定理 1. 设 u、v 是故障树 FT 的任意两个事件,u_set 和 v_set 分别是 u 和 v 的完全依赖集,且 $u\_set \cap v\_set = \varnothing$,则 $u \perp v$.

证明. 假设命题不成立,根据定义 1,存在基本事件 $u_0 \in events(u)$ 和 $v_0 \in events(v)$,且 $prob(u_0 v_0) \neq prob(u_0) \times prob(v_0)$;根据定义 3,$u_0$ 和 v_0 都依赖于同一个依赖类型,假设 $u_0, v_0 \in L_i$;因为 $u_0 \in events(u)$ 和 $v_0 \in events(v)$,所以 $u_0 \not\perp u$ 且 $v_0 \not\perp v$,根据定义 3(3),u 和 v 都依赖于依赖类型 i;又因为 u_set 和 v_set 分别是 u 和 v 的完全依赖集,故而 $i \in u\_set$ 且 $i \in v\_set$,这与题设 $u\_set \cap v\_set = \varnothing$ 矛盾.命题得证. 证毕.

故障树转变为与其依赖关系等价的依赖树后,故障树的每一个事件对应依赖树的一个相应节点.本节前面的定义和命题同样适用于依赖树,故障树的独立模块识别也转化为相应依赖树的独立模块识别.通过对依赖树各节点的相关属性进行分析,可识别出依赖树的独立模块,从而得到相应故障树的独立模块,以便进行其它可靠性分析.

6.1 独立模块识别算法

经过预处理、构建依赖树后,依赖树各节点除去依赖集属性外其它属性都已填充,独立模块识别算法 IIM(Identification of Independent Module)根据依赖树各节点的祖先集属性和依赖集属性进行识别.其中心思想为:填充每个节点的依赖集属性,然后每个内部节点分别同与其非直系血亲的所有其它节点比对依赖集属性,若都不存在交集,则该内部节点即为独立模块,算法如算法 4 所示.

算法 4. 独立模块识别算法 IIM.

```
IdentificationIndependentModule(DT, inputdep)
功能:识别依赖树中的独立模块
输入:依赖树 DT(s,MNS,LNS),各叶节点的依赖集 inputdep
输出:独立模块集 IM
1.  IM←∅;
2.  SetLeafSetDep(DT.LNS, inputdep);
3.  for each u∈DT.LNS
4.      for each v∈u.ancestor_set
5.          v.depend_set← v.depend_set∪u.depend_set;
6.  for each u∈DT.MNS
7.      bIM=TRUE;
8.      for each v∈DT.MNS∪DT.LNS /u.ancestor_set
9.          if u∉v.ancestor_set
10.             if u.depend_set∩v.depend_set≠∅
11.                 bIM=FALSE;
12.                 break;
13.     if bIM==TRUE
14.         IM←IM∪{u};
15. return IM.
```

IIM 算法执行前,各节点的依赖集属性还未填充,因此第 1 行首先将独立模块集设为空,第 2 行将已知的故障树各基本事件依赖类型赋值给依赖树相应叶节点的依赖集属性.第 3~5 行根据每个叶节点的依赖集属性,更新该叶节点所有祖先的依赖集属性;这段代码执行完毕后,依赖树所有节点的依赖集属性完全确定.第 6~15 行将任一个内部节点的依赖集属性与它的所有非直系血亲节点的依赖集属性进行比较,如果没有交集,根据定理 1,该节点为独

立模块.

定理 2.　IIM 算法 3～5 行执行完毕后，各节点所得的依赖集属性为完全依赖集.

证明.

(1) 设 v 是依赖树的任一节点，若 $v\in DT.LNS$，由于叶节点的依赖集属性是根据系统各基本元件间的共因故障类型设置的，每个叶节点的依赖集包括了所有该叶节点所依赖的依赖类型，并且所有该叶节点所依赖的依赖类型都包括在该叶节点的依赖集中，根据定义 3(4)，很明显 $v.depend\_set$ 是 v 的完全依赖集.

(2) 若 $v\notin DT.LNS$，下面分两步进行证明：

(2.1) 本步证明 $v.depend\_set$ 的每个元素都是 v 的依赖类型.

反证法，假设 $\exists i\in v.depend\_set$ 且 v 并不依赖于 i.

则 $\forall u\in L_i$，由定义 3(3) 可知，$u\perp v$；　①

因为 $v.depend\_set$ 初始为空，由 3～5 行可知，依赖类型 i 必由某个基本事件 u_0 主导加入，即 $\exists u_0\in L_i$ 使得 $v\in u_0.ancestor\_set$，故而 $u_0\not\perp v$.　②

①和②互相矛盾，本步得证.

(2.2) 本步证明 v 的所有依赖类型都在 $v.depend\_set$ 中.

反证法. 假设 $\exists i$ 使得 v 依赖于 i 而 $i\notin v.depend\_set$.

由 3～4 行知，$events(v)\cap L_i=\varnothing$；否则假设 $u_0\in events(v)\cap L_i$，则 $v\in u_0.ancestor\_set$，$i\in u_0.depend\_set$，于是 $i\in v.depend\_set$，与先前题设矛盾. 所以 $events(v)\cap L_i=\varnothing$.　③

因为 v 依赖于 i，则 $\exists u_1\in events(v)$，$u_2\in L_i$ 且 $prob(u_1u_2)\neq prob(u_1)\times prob(u_2)$，由定义 3 可知，$u_1\in L_i$，于是 $u_1\in L_i\cap events(v)$.　④

③和④互相矛盾，本步得证.

由(1)和(2)可得 $v.depend\_set$ 是 v 的完全依赖集.

命题得证.　　证毕.

定理 3.　IIM 算法执行完毕后，所得的 IM 为完全独立模块集.

证明.　分两步进行，第 1 步证明 IM 中的所有模块都是独立模块，第 2 部证明依赖树的所有独立模块都在 IM 中.

(1) 本步证明 IM 中的所有模块都是独立模块.

$\forall u\in IM$，由第 6 行可知，u 不是依赖树的顶节点或叶节点；任一节点 v，

若 $v\in u.ancesor\_set$，则 $events(u)\subseteq events(v)$；

若 $u\in v.ancesor\_set$，则 $events(v)\subseteq events(u)$；

若 $v\notin u.ancesor\_set$ 且 $u\notin v.ancesor\_set$，则由第 10～14 行可知，$u.depend\_set\cap v.depend\_set=\varnothing$，由定理 1 可得 $u\perp v$；

由定义 2 可知 u 是依赖树的一个独立模块. 本步得证.

(2) 本步证明依赖树的所有独立模块都在 IM 中.

反证法，假设依赖树存在独立模块 u，但是 $u\notin IM$. 由定义 2 可知，u 既不是顶节点也不是叶节点；由第 8～12 行可知，存在一个节点 v，且 $v\notin u.ancestor\_set$ 和 $u\notin v.ancestor\_set$，有 $u.depend\_set\cap v.depend\_set\neq\varnothing$，使得变量 bIM=FALSE，以至于 u 未被加入 IM 中. 这又与 u 是独立模块相矛盾. 本步得证.

由(1)和(2)，命题得证.　　证毕.

6.2　最小独立模块识别算法

识别出独立模块集 IM 后，还要从中识别出最小独立模块. 最小动态独立模块是必须用 Markov 链求解的模块，识别出最小独立模块，能够显著缩减状态空间，降低求解开销.

定义 4.　设 u 是依赖树 DT 的一个独立模块，称 u 是一个最小独立模块当且仅当：任给另外一个独立模块 v，则 $u\notin v.ancestor\_set$.

最小独立模块识别算法 IMIM(Identification of Minimum Independent Module) 如算法 5 所示，第 1 行首先将最小独立模块集 MIM 设为空集，第 2 行对 IM 集中各节点的遍历深度属性 $depth$ 进行降序排序并赋值给 IM_0. 排在 IM_0 的首位模块必定为最小独立模块，第 4～8 行将首位模块 u 加入 MIM 的后面，并从 IM_0 中去除 u 及 u 的祖先节点，如此反复，直至 IM_0 为空，此时 MIM 即为最小独立模块集.

算法 5.　最小独立模块识别算法 IMIM.

```
IdentificationMinimizeIM(IM)
功能：识别最小独立模块
输入：独立模块集 IM
输出：最小独立模块集 MIM
1.  MIM←∅;
2.  IM0←SortIM(IM);
3.  while IM0≠∅
4.      u←FirstNode(IM0);
5.      IM0←RemoveFirstNode(IM0);
6.      MIM←AddLast(MIM, u);
7.      AncestorSet←u.ancestor_set∩IM0;
8.      IM0←RemoveSet(IM0, AncestorSet);
9.  return MIM.
```

定理 4. IMIM 算法执行完毕后，所得的 MIM 为完全最小独立模块集.

证明. 由第 4～6 行代码可知，$MIM \subseteq IM$，故而 MIM 是独立模块集. 下面分两步进行，第 1 步证明 MIM 的所有元素都是最小独立模块，第 2 步证明所有最小独立模块都是 MIM 中的元素.

(1) 反证法. 假设 $\exists u \in MIM$，但 u 不是最小独立模块，也就意味着 $\exists v \in IM$，使得 $u \in v.ancestor\_set$. 若 v 在第 6 行被加入 MIM，因为 $v.depth > u.depth$ 且 IM_0 为 IM 的 $depth$ 属性的降序排列，此时 u 必然还未被加入 MIM. 在第 7 行执行前，若 $u \in IM_0$，第 7 行执行后 $u \in AncestorSet$，则第 8 行执行后 $u \notin IM_0$，故而在后面的处理中 u 不可能被加入 MIM，这与题设矛盾，本步得证.

(2) 反证法. 假设存在 u 是最小独立模块，但 $u \notin MIM$. 因为 IM 是完全独立模块集，故而 $u \in IM$. 由第 4～8 行可知，必存在一次循环，在此次循环中 $u \in AncestorSet$，被从 IM_0 中去除；假设在此次循环中被加入 MIM 的是 v，则由第 7 行可知，$u \in v.ancestor\_set$，而这又与 u 是最小独立模块相矛盾. 本步得证.

由(1)和(2)，命题得证. 证毕.

7 算法分析

本节对 IIMKDR 方法的各算法进行时间复杂性理论分析，并将其与其它方法在性能上进行理论对比分析.

7.1 时间复杂性分析

设故障树 $FT=(V,E)$，其中 V 为事件集，E 为有向边集.

算法 1 构建依赖树算法 DTC 中第 1 行的循环占用的时间为 $\Theta(V)$. 与传统深度优先遍历算法不同，需对依赖树中的重复节点及其子节点更新祖先集属性，对于故障树中任一事件 v，过程 DFS_VISIT 可能被调用多次. 下面详细分析之：

(1) 故障树中不含有重复事件.

在这种情况下，每个事件至多仅有一个父事件. 对于任一事件 v，过程 DFS_VISIT 仅被调用一次. 在 DFS_VISIT(u,u_0) 的一次执行过程中，第 12～19 行被执行了 $|adj[u]|$ 次，而第 21～25 行从不执行. $\sum_{v \in V} |adj[v]| = \Theta(E)$，执行 DTC 函数中第 9 行的总代价为 $\Theta(E)$. 因此 DTC 的时间开销为 $\Theta(V+E)$.

(2) 故障树中含有重复事件.

在这种情况下，有多个事件含有多个父事件，情况稍显复杂. 假设故障树中含有 i 个重复事件，分别为 $m_1, m_2, \cdots, m_i$. 任给一个事件 $m_j (1 \leq j \leq i)$，则其父事件个数 $pnum[m_j] = |m_j.parent\_set|$. 另外对每一个事件 u，都对应一个发现时间和一个完成时间，其对应的时间戳分别为 $d[u]$ 和 $f[u]$，事件 u 在时刻 $d[u]$ 之前为白色，在时刻 $d[u]$ 和 $f[u]$ 之间为灰色，之后为黑色；同时也意味着以事件 u 开始的深度优先遍历共执行了 $(f[u]-d[u]+1)/2$ 次 DFS_VISIT 算法，$(f[u]-d[u]+1)/2$ 也等于以 u 为根所组成的子树中的节点数. 故而由于故障树中含有重复事件，DFS_VISIT 中第 20～25 行被多执行了 $\sum_{j=1}^{i}(pnum[m_j]-1)\times(f[m_j]-d[m_j]+1)/2$ 次. 设 p 为所有重复事件中父事件个数的最大值，q 为所有以重复事件为根的子树中事件个数的最大值，则 DFS_VISIT 中第 20～25 行最多被执行了 ipq 次. 因此，DTC 的时间开销为 $\Theta(V+E+ipq)$.

算法 4 独立模块识别算法 IIM 的第 3～5 行时间开销小于 $\Theta(V^2)$，第 6～14 行执行代价也小于 $\Theta(V^2)$，于是 IIM 算法的时间复杂度最坏情况下为 $\Theta(V^2)$；算法 5 最小独立模块识别算法 IMIM 的第 2 行根据节点的深度属性进行降序排序，最优排序算法时间复杂度为 $\Theta(|IM|\log_2|IM|)$，第 4～8 行最多执行了 $|IM|$ 次. 故而识别阶段时间复杂度最坏情况下为 $\Theta(V^2)$.

综合构建依赖树算法 DTC、独立模块识别算法 IIM、最小独立模块识别算法 IMIM 等的时间复杂性分析，IIMKDR 方法的时间复杂度最坏情况下为 $\Theta(V^2)$.

7.2 性能分析

由于 IIMKDR 方法在构建依赖树阶段将故障树转化为依赖树，然后在依赖树的基础上进行处理，从而使其具有处理动态故障树的能力；由于算法 3 深度优先遍历函数 DFS_VISIT 增加了 20～25 行，从而使得 IIMKDR 方法能够对具有重复事件的故障树进行独立模块识别；又由于 IIMKDR 方法基于各节点的依赖集属性进行识别，从而可对具有互相依赖基本事件的故障树进行独立模块识别. 因此，IIMKDR 方法不仅可对传统故障树进行独立模块识别，还可对具有互相依赖基本事件和重复事件的动态故障树进行识别.

Sun 方法也使用了依赖属性的思想，也可对具有互相依赖基本事件的故障树进行独立模块识别；

其基于静态故障树进行分析，没有针对动态逻辑门进行专门的预处理，不适合处理动态故障树；其独立模块识别的核心算法仍是基于 Dutuit 方法，无法处理具有重复事件的故障树.

Huang 方法仅对动态故障树进行了一定的预处理，其独立模块识别的核心算法也是基于 Dutuit 方法，无法处理具有相互依赖基本事件和重复事件的故障树.

由上节可知，IIMKDR 方法的时间复杂度最坏情况下为 $\Theta(V^2)$. Dutuit 方法的时间开销为 $\Theta(V+E)$. Sun 方法在其“模块化”步骤中完全采用 Dutuit 方法，但为了找出最小独立模块，又另加了“更新依赖信息”步骤和“为每个依赖信息重新模块化”步骤，时间总开销为 $\Theta(V^2)$. Huang 方法识别核心算法完全同 Dutuit 方法一样，时间开销为 $\Theta(V+E)$.

8　实验验证

本节使用两种方法对 IIMKDR 方法进行验证，首先将 IIMKDR 应用于我们所设计的一个星载计算机系统，由于该系统是一个动态系统，而且其基本元件并不完全独立，故而 Sun 方法和 Huang 方法无法对其识别；然后采用模拟仿真方法对 IIMKDR 方法、Sun 方法和 Huang 方法进行详细的求解开销、错误率的对比.

8.1　实际案例验证

图 9 和图 10 分别为一个星载计算机的系统结构图和动态故障树，星载计算机系统包括 4 个模块：处理模块、电源模块、存储模块和总线模块. 处理模块中含有 3 个处理单元 A1、A2 和 A3，A3 为 A1 和 A2 的冷备份，只要 3 个处理器全部失效就会导致系统失效. 电源模块包括 3 个功能单元 P1、P2 和 C，其中 P1 和 P2 为两个供电单元，C 为切换控制单元，电源模块为 3 个处理单元供电，只要 P1 或 P2 有一个正常运作并且 C 正常运作，便能给 3 个处理单元正常供电. 存储模块包括 5 个存储单元，若其中 3 个单元失效，则存储模块失效；存储单元 M1 和 M2 连

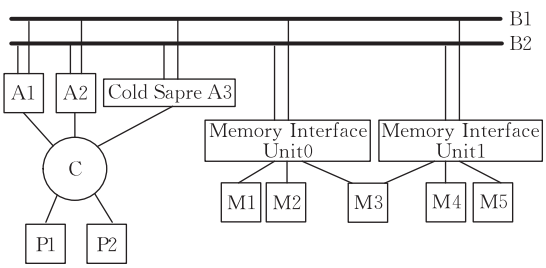


图 9　星载计算机系统结构

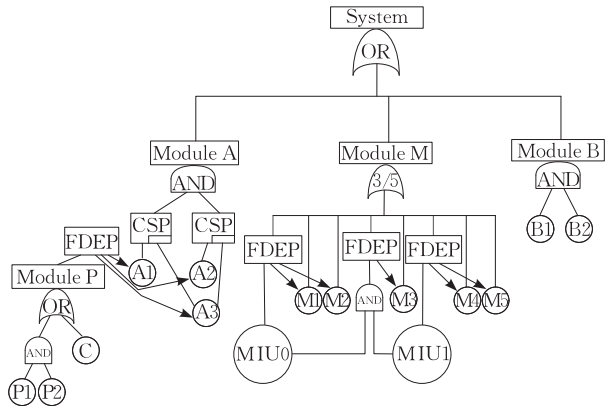


图 10　星载计算机故障树

接存储接口 MIU0，M4 和 M5 连接 MIU1，M3 即连接 MIU0 又连接 MIU1. 总线模块包括两个系统总线 B1 和 B2，只要两条总线全部失效就会导致系统失效.

另外假设基本元件 A1 和 A2 因为共因故障原因相互关联，设这两个元件依赖于依赖类型 1；A3 单独依赖于依赖类型 2；P1 和 P2 依赖于依赖类型 3，C 依赖于依赖类型 4；M1、M3 和 M5 依赖于依赖类型 5，M2 和 M4 依赖于依赖类型 6；MIU0 和 MIU1 依赖于依赖类型 7；B1 和 B2 依赖于依赖类型 8.

经过预处理，图 10 可转变为图 11.

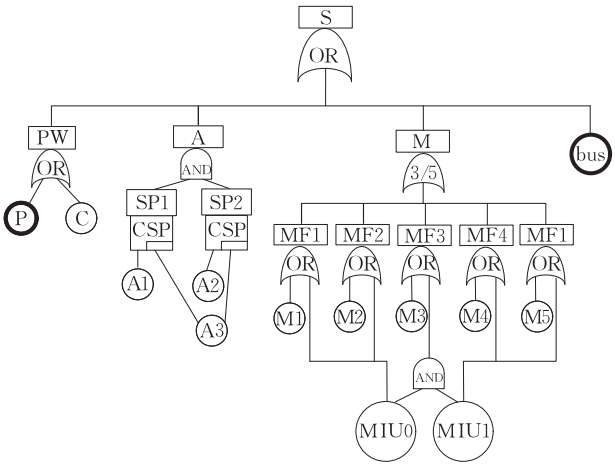


图 11　预处理后的星载计算机故障树

图 11 中粗线边框的事件为宏事件，因为 P1 和 P2 经 AND 门的组合事件在故障树中仅出现一次，因此可将此两基本事件（包括与其相连的 AND 门）以宏事件 P 代替，同理也可得到宏事件 bus. 原故障树中含有 FDEP 门，根据预处理中关于 FDEP 门的处理，将其转换为 OR 门并外联之公共父事件.

输入预处理后的星载计算机故障树，经过依赖树构建算法处理后，输出如图 12 所示的星载计算机依赖树；为了节省篇幅，仅显示 PW、A、M、bus 4 个

节点的各属性、首次遍历发现时间、首次遍历结束事件，如表 1 所示.

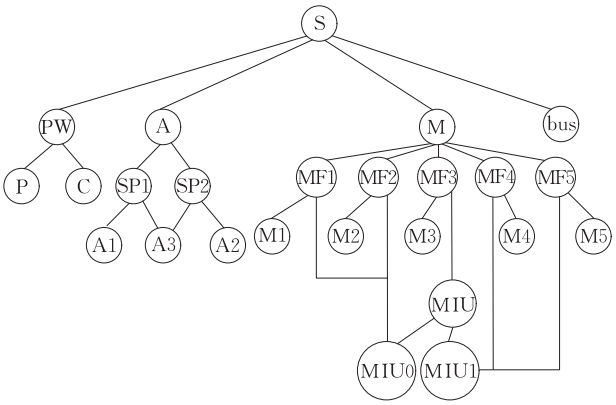


图 12 星载计算机依赖树图形表示

表 1 依赖树节点属性表

	ID	*parent_set*	*ancestor_set*	*depth*	*d*[*u*]	*f*[*u*]
PW	1	S	S	1	1	6
A	4	S	S	1	7	18
M	10	S	S	1	19	46
bus	24	S	S	1	47	48

经过独立模块识别算法 IIM 和最小独立模块识别算法 IMIM 处理后，星载计算机系统的独立模块集和最小独立模块集分别为 $IM=\{\text{PW},\text{A},\text{M},\text{bus}\}$，$MIM=\{\text{PW},\text{A},\text{M},\text{bus}\}$，直观上很容易看出这是正确的答案.

Sun 方法和 Huang 方法，由于其应用局限性，无法求解本案例.

8.2 仿真验证

独立模块识别的时间开销与许多因素都有关系，这些因素包括故障树结构、事件个数、重复事件位置及个数、依赖类型个数等. 每次实验都可看做是个特例，无法说明某个方法的优劣. 为了尽可能地与 Sun 方法、Huang 方法进行对比，将它们都在 Matlab 平台上使用相同的编程风格和库函数实现. 由于 Sun 方法仅应用于静态故障树，故而在本组实验中，剔除动态逻辑门在识别过程中的影响，求解的对象实际是静态故障树(也可看作 IIMKDR 方法中的依赖树)，对比 3 种方法的核心算法在具有重复事件、具有相互依赖基本事件的故障树上的适用性、求解开销.

静态故障树(亦可理解为依赖树)随机仿真生成过程如下：(1) 若无指定总结点数，随机生成总结点数；(2) 随机生成叶节点个数，叶节点个数要大于总结点数的一半；(3) 随机生成依赖类型个数，依赖类型个数要小于叶节点个数；(4) 随机生成最小独立模块，其个数要小于依赖类型个数，若依赖于同一个依赖类型的叶节点个数大于 1，则这些叶节点属于且仅属于某个最小独立模块；(5) 若依赖于某个依赖类型的叶节点只有一个，则该叶节点称为独立叶节点，以所有最小独立模块、独立叶节点为基本单位，随机生成上层节点(包括独立模块)，直至根节点. 由此仿真过程可知，当故障树随机生成完毕，故障树中含有的最小独立模块和独立模块皆是已知，将其与 3 种方法识别所得的结果进行比较，便可得到识别准确率.

仿真程序使得故障树的生成尽可能具有随机性，但是真正的随机数是不存在的，生成的过程中也有一定的条件限制，因此生成的故障树还不能称为完全随机，某个或多个故障树可能会更加倾向于适用某种方法，从而对实验结果造成误差，我们采用生成求解很多次并取平均值的方法来尽量减小这种误差. 为了对比的公平性，3 个方法识别的对象都是同一个随机仿真生成的故障树，正如前面所述，该故障树实际上仅含依赖信息，因此 3 种方法针对逻辑结构所采取的预处理过程可以忽略，本节仅是比较 3 个方法核心算法的优劣. 虽然 Sun 方法和 Huang 方法都比较复杂，但是其核心算法都还是比较直观简洁，而且核心算法在方法的正确性和效率上起着绝对主要的作用；所以尽管我们的实现不可避免会引入误差，但是我们仅是对比其核心算法，这些误差是可以容忍的.

将实验分成以下 3 组，为了克服每次实验的偶然性，每个实验数据都是多次随机取平均值得到. 其中遗漏率(Negative Error Rate)是指本应识别而未能识别为独立模块的数目所占的比重，错加率(Positive Error Rate)是指本非而被识别为独立模块的数目所占的比重，IMS(Independent Module Set)和 MIMS(Minimal Independent Module Set)分别指独立模块集和最小独立模块集.

实验 1. 仅含重复事件的故障树.

每次故障树生成时，总事件个数设定为 600，重复事件的个数及位置随机生成，重复 10 000 次. 当含有某重复事件数目的案例大于 20 次时，再求此重复事件数目下的时间开销、错加率和遗漏率等的平均值.

实验结果如图 13、图 14 和图 15.

含有重复事件的故障树中所有的独立模块都能被 3 种算法识别出来，故而 3 种算法的独立模块识别遗漏率都为零，为节省篇幅没有列入文中. 由图 13、图 14 和图 15 可知，IIMKDR 方法对于含有重复事件的故障树，其独立模块和最小独立模块的

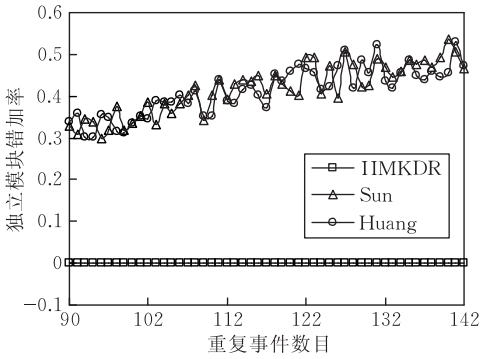


图 13　3 种方法对含有重复事件故障树的独立模块识别错加率

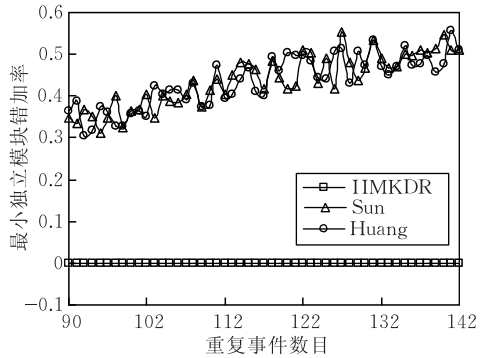


图 14　3 种算法对含有重复事件故障树的最小独立模块识别错加率

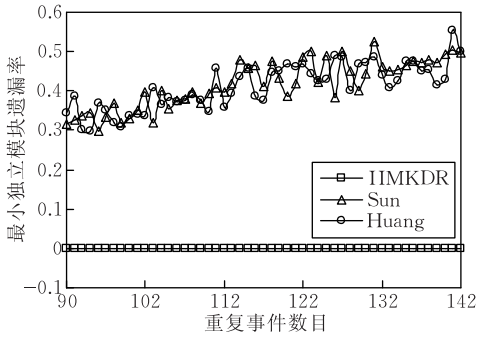


图 15　3 种算法对含有重复事件故障树的最小独立模块识别遗漏率

错加率和遗漏率都为零，也就是说该方法能够准确识别含有重复事件故障树的独立模块和最小独立模块. 而 Sun 方法和 Huang 方法对于独立模块的识别虽然遗漏率为零，但仍有错加率；且对最小独立模块的识别既有错加率又有遗漏率，也就是说这两种方法不适用含有重复事件故障树的模块识别.

实验 2.　仅含相互依赖基本事件的故障树.

每次故障树生成时，总事件个数设定为 600，依赖类型组合随机生成，重复 10000 次，仅保留每种依赖类型数目的案例数大于 20 的情况，并按照不同的依赖类型数目求识别时间、遗漏率、错加率等平均值.

实验结果如图 16、图 17 所示.

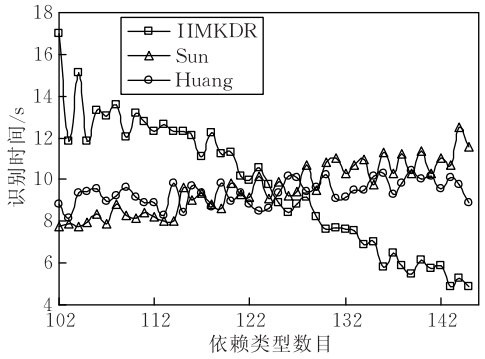


图 16　3 种算法对含有相互依赖基本事件故障树的独立模块识别时间开销

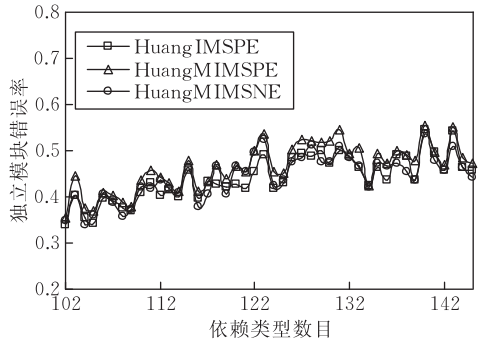


图 17　Huang 方法对含有相互依赖基本事件故障树的独立模块识别错误率

在 IIMKDR 方法的实现中，每个基本事件都归属于且仅归属于一个依赖类型，而平时所称的相互依赖基本事件是指多个基本事件归属于同一个依赖类型. 图 16 和图 17 中所指的依赖类型数目等价于 IIMKDR 方法中所关联基本事件数大于 1 的依赖类型数目. 在本次实验中，总结点数是一定的，当依赖类型数增加时，则 IIMKDR 实现中仅含一个基本节点的依赖类型数目会有相应减少的趋势，正如图 16 所示，在依赖类型数目处于区间(102～142)时，时间开销呈下降的态势. 当然该种下降态势并非是绝对的，但是足以说明 IIMKDR 方法的时间开销与依赖类型数目之间并没有递增的关系.

Huang 方法并不区分相互依赖基本事件和一般事件，故而其时间开销基本保持不变. 而当依赖类型增加时，Sun 方法处理依赖信息及有关寻觅最小模块所花费的开销就会增加，从而导致其与依赖类型数目有递增的关系.

对于仅含相互依赖基本事件的故障树，IIMKDR 方法和 Sun 方法均能正确识别，而 Huang 方法则会产生独立模块识别错加率和最小独立模块识别错加率及遗漏率. 也就是说只有 IIMKDR 方法 和 Sun 方法可以对仅含相互依赖基本事件的故障树进行模块识别.

实验 3. 不含相互依赖基本事件和重复事件的故障树

每次故障树生成时,总事件个数随机从 100 到 300 取值,每个实验数据都是重复 70 次的平均值.

实验结果如图 18 所示.

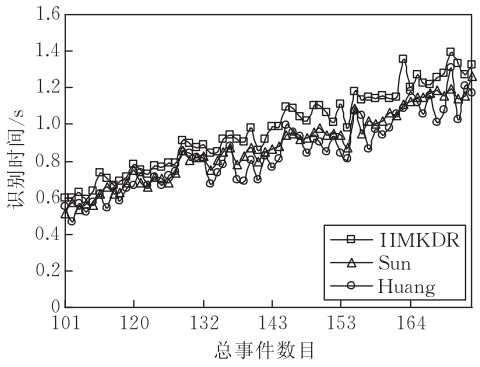


图 18 3 种算法对不含相互依赖基本事件和重复事件故障树的独立模块识别时间开销

3 种算法都能够正确识别不含相互依赖基本事件和重复事件故障树的独立模块和最小独立模块,由于错误率都是 0,为节省篇幅就不列图了.由图 18 可看出,3 种算法中,IIMKDR 方法时间开销略高于 Sun 和 Huang,Sun 的时间开销又略高于 Huang.

实验 4. 既含重复事件又含相互依赖基本事件的故障树.

因变量:识别时间、遗漏率、错加率;

实验参数:节点个数:600;重复事件数:100;依赖类型数:100;

实验方法:故障树生成时,总事件个数、重复事件个数、依赖类型数分别设置为 600、100、100,重复事件的位置和相互依赖基本事件的位置皆随机生成,重复 10000 次,并求解时间开销、错加率和遗漏率等的平均值.

实验结果如表 2 所示,其中 IMSPE、IMSNE、MIMSPE 和 MIMSNE 分别为独立模块错加率、独立模块遗漏率、最小独立模块错加率和最小独立模块遗漏率.

表 2 既含重复事件又含相互依赖基本事件的故障树求解性能对比表

	识别时间/s	IMSPE	IMSNE	MIMSPE	MIMSNE
IIMKDR	11.993	0	0	0	0
Sun	10.873	0.371	0	0.366	0.343
Huang	10.363	0.682	0	0.713	0.707

若一个故障树中既含重复事件又含相互依赖基本事件,则 IIMKDR 方法能够正确识别出独立模块和最小独立模块;Sun 方法和 Huang 方法在独立模块的识别过程中,虽然其遗漏率为 0,但是其错加率大于 0,也就是说会将一些不是独立模块的节点识别为独立模块.由于 Sun 方法能够处理相互依赖的基本事件,故而其错加率要明显低于 Huang 方法.从表 2 也可看出,IIMKDR 方法的识别时间要略高于 Sun 方法和 Huang 方法.

9 结 论

独立模块的识别在动态故障树研究中具有极其重要的意义,它是灵敏性分析、可靠性的定性和定量分析的基础.本文提出了一种基于亲戚依赖关系的独立模块识别方法 IIMKDR,并将之与其它两种方法进行了对比,理论分析及实验结果均表明:它可以对具有相互依赖基本事件和重复事件的动态故障树进行处理,而其它两种方法都不具备这样的功能;对于不含相互依赖基本事件和重复事件的故障树,处理开销略高于 Sun 和 Huang.

下一步我们将把 IIMKDR 方法应用到动态故障树的重要度分析研究中,使其能显著降低问题求解的代价.

参 考 文 献

[1] Dugan J B, Bavuso S J, Boyd M A. Dynamic fault-tree models for fault-tolerant computer systems. IEEE Transactions on Reliability, 1992, 41(3): 363-377

[2] Zhu Zheng-Fu, Li Chang-Fu, He En-Shan et al. Dynamic fault tree analysis method based on Markov chain. Binggong Xuebao/Acta Armamentarii, 2008, 29(9): 1104-1107 (in Chinese)
(朱正福,李长福,何恩山等. 基于马尔卡夫链的动态故障树分析方法. 兵工学报, 2008, 29(9): 1104-1107)

[3] Amari S, Dill G, Howald E. A new approach to solve dynamic fault trees//Proceedings of the Annual Reliability and Maintainability Symposium. Tampa, Florida, USA, 2003: 374-379

[4] Bobbio A, Codetta-Raiteri D, De Pierro M et al. Efficient analysis algorithms for parametric fault trees//Proceedings of the Workshop on Techniques, Mehodologies and Tools for Performance Evaluation of Complex Systems. Torino, Italy, 2005: 91-105

[5] Marquez D, Neil M, Fenton N. Solving dynamic fault trees using a new hybrid bayesian network inference algorithm//Proceedings of the Mediterranean Conference on Control Automation. Ajaccio, France, 2008: 1526-1531

[6] Yuge T, Yanagi S. Quantitative analysis of a fault tree with priority AND gates. Reliability Engineering & System Safety, 2008, 93(11): 1577-1583

[7] Zhou Zhong-Bao, Ma Chao-Qun, Zhou Jing-Lun, Dong Dou-Dou. Dynamic fault tree analysis based on dynamic Bayesian networks. System Engineering Theory and Practice, 2008, 28(2): 35-42

[8] Gulati R, Dugan J B. A modular approach for analyzing static and dynamic fault trees//Proceedings of the Annual Reliability and Maintainability Symposium. Philadelphia, PA, USA, 1997: 57-63

[9] Dutuit Y, Rauzy A. A linear-time algorithm to find modules of fault trees. IEEE Transactions on Reliability, 1996, 45(3): 422-425

[10] Sun H, Andrews J D. Identification of independent modules in fault trees which contain dependent basic events. Reliability Engineering & System Safety, 2004, 86(3): 285-296

[11] Huang Chin-Yu, Chang Yung-Ruei. An improved decomposition scheme for assessing the reliability of embedded systems by using dynamic fault trees. Reliability Engineering & System Safety, 2007, 92(10): 1403-1412

[12] Lo H K, Huang C Y, Chang Y R et al. Reliability and sensitivity analysis, of embedded systems with modular dynamic fault trees//Proceedings of the IEEE Region 10 Conference. Melbourne, Australia, 2005. New York: IEEE, 2006: 1863-1868

[13] Shi Ding-Hua, Wang Song-Rui. Methods and Theories of Fault Tree Analysis. Beijing: Beijing Normal University Press, 1993(in Chinese)
(史定华，王松瑞. 故障树分析技术方法和理论. 北京：北京师范大学出版社，1993)

ZHANG Hong-Lin, born in 1977, Ph. D. candidate. His current research interests include reliability analysis and fault-tolerance.

ZHANG Chun-Yuan, born in 1964, Ph. D., professor, Ph. D. supervisor. His major research interests include computer architecture and high-performance.

LIU Dong, born in 1981, Ph. D., lecturer. His current research interests include reliability analysis of computer system and fault-tolerance.

Background

Fault tree (FT) analysis is widely used for reliability analysis of complex and critical computer-based system. Dynamic Fault Tree (DFT) extends traditional FT by defining dynamic gates to model dynamic behavior. The research of dynamic fault tree analysis has been investigated greatly for these ten years, and the research center lies in University of Virginia, Duke University and Loughborough University. Markov model is always applied to solve DFT, which has its own drawback. The number of system states involved in a Markov process will increase exponentially with the number of basic events in the dynamic fault tree. On the other hand, usually only a small number of basic events are dependent and require a Markov model. DFT can be modularized into independent static subtrees and independent dynamic subtrees. The results of each modular analysis can be then combined to obtain the final system measures. So it is one of the key questions in the research of dynamic fault tree how to partition and identify the independent module.

As we know, the interdependent basic events and repeated events complicate the identification in the dynamic fault tree. Some previous researches gave some methods on how to identify independent module of fault trees which have interdependent basic events or dynamic gates. But to my best knowledge, there is no method to be applied to the dynamic fault trees with interdependent basic events and repeated events. Based on the kinship dependence relation, this paper proposes an identification method of independent module, called IIMKDR. IIMKDR converts the fault tree to dependent tree according the dependence relations among the nodes. We construct the object of dependent tree and the node, which consists of the dependent set property and the ancestor set property. Through the analysis of these properties, we can obtain all the independent modules and minimum independent modules. This method has no relation with the existence of interdependent basic events, repeated events and dynamic logic gates, so it can be used to identify the independent module of the dynamic fault trees which have interdependent basic events or repeated events.

This research is partially supported by the National Natural Science Foundation of China (60904082) and National Defence Pre-Research of China (51320010201) and has been applied to the reliability analysis of our projects. The aim of the research is to establish and develop a systematic method on fault tolerance design technology and reliability analysis technology in critical computer-based system, such as space information systems and aircraft systems. Until now, the research team has published more than 20 papers about the fault tolerant design and the reliability analysis of space information systems. The purpose of this paper is to solve one of the problems in the reliability quantitative analysis of the complex system.