基于故障配置的故障树生成

黄鸣宇 魏 欧 胡 军
(南京航空航天大学计算机科学与技术学院 南京 211100)

摘 要 故障树分析是提高系统安全性和可靠性的有效方法。传统的人工故障树生成方式难以解决当前系统的庞大规模与复杂性的问题,且容易出错。为此,提出基于故障配置的故障树生成方法,引入软件产品线的可变性管理,用于系统故障建模与形式化分析。首先,定义故障特征图模型用于刻画系统故障间的约束关系,基于Kripke结构定义故障标记迁移系统来描述系统的行为;然后,基于模型的语义建立通过模型检测生成故障树的过程;最后,通过时序逻辑描述系统安全属性,利用模型检测工具SNIP验证安全属性进而生成故障树。案例研究验证了该方法的有效性。

关键词 故障树,故障配置,模型检测
中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.02.029

Fault Tree Generation Based on Fault Configuration

HUANG Ming-yu WEI Ou HU Jun
(Department of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 211100, China)

Abstract Fault tree analysis is an effective method to improve system safety and reliability. However, traditional manual fault tree generation is difficult to solve the problem of large scale and complexity of system and error-prone. In order to systematically support system faults modeling and formal analysis, a fault tree generation method based on fault configuration was proposed in this paper by introducing variability management form software product line into system faults modeling. Firstly, we defined fault feature diagram for describing the constraints among faults and proposed fault labeled transition system based on Kripke structure to describe system behavior. Secondly, a model checking procedure of generating fault tree was established based on the model semantics. Finally, using model checker SNIP, the safety properties specified with temporal logic were verified and fault tree was generated based on the result. Case study shows the effectiveness of the proposed approach.

Keywords Fault tree, Fault configuration, Model checking

1 引言

系统安全性分析[1]是系统安全工程的核心内容,其主要内容是研究当部分系统组件由于故障处于非正常工作状态时的系统行为。系统安全性分析是安全评估的基础,其主要目的是了解、查明系统存在的危险,以确保系统满足规定的安全需求。

故障树分析(Fault Tree Analysis,FTA)[2]是最常见的安全性分析技术之一;它基于自顶向下的演绎方法对系统失效进行定性和定量分析。故障树作为分析的基础,以树状逻辑关系图的形式,用规定的事件、逻辑门等符号刻画导致系统失效的基本事件之间的逻辑关系。在传统的故障树生成过程中,通常需要针对给定的系统失效事件,由安全工程师对失效事件进行分析,通过迭代的方式获取所有可能导致系统失效的原因,结果表示为故障树。因此故障树的质量取决于安全工程师的个人技巧与经验,这也使得故障树生成过程容易出错,同时耗费大量时间[3]。

随着系统规模和复杂程度的不断增加,传统安全性分析方法面对着巨大的挑战[4]。如何改善分析过程、降低成本、提高分析质量,成为了国内外的一个研究热点。近些年,基于模型的安全性分析方法逐渐得到了工业界和学术界的关注[5-7]。法国达索飞机公司的猎鹰7x型飞机(falcon 7x)的飞行控制系统在使用数据流语言AltaRica描述的系统模型基础上通过认证[6]。尚未发布的民用机载系统和设备进行安全性评估过程的准则和方法ARP4761修订版可能加入使用基于模型的安全性分析的条款[8]。以统一的形式化系统和故障模型为

到稿日期:2016-01-07 返修日期:2016-07-03 本文受国家自然科学基金项目(61170043),国家重点基础研究发展计划(973)项目(2014CB744904),航空科学基金项目(20155552047),江苏省研究生培养创新工程(SJLX15_0139),南京航空航天大学研究生创新基地(实验室)开放基金(kfjj20161602),中央高校基本科研业务费专项资金资助。

黄鸣宇(1994—),男,硕士生,CCF会员,主要研究方向为系统安全性分析、模型检测,E-mail:hmy189@nuaa.edu.cn;魏 欧(1974—),男,博士,副教授,主要研究方向为形式化方法、软件自动验证,E-mail:owei@nuaa.edu.cn;胡 军(1973—),男,博士,副教授,主要研究方向为系统安全性分析、软件验证,E-mail:hujun.nju@139.com。

基础，系统开发和安全分析故障紧密联系在一起。通过自动化地生成故障树等传统安全分析产物，基于模型的安全性分析在减少人工的同时提高了安全性分析的准确性、完整性和重用性[4,8]。基于上述思想，本文针对现有故障树生成中的问题，提出一种采用基于软件产品线模型检测自动生成故障树的方法。

模型检测[9]是一种自动形式化的验证技术，用于判断计算机系统的正确行为属性，其基本思想是用状态迁移系统(S)表示系统的行为，用模态逻辑公式或时序逻辑公式(φ)描述系统的性质。这样，“系统是否具有所期望的性质”就转化为数学问题“状态迁移系统 S 是否是满足公式 φ 的一个模型”，用公式表示为 $S\vdash\varphi$。对于有限状态系统，这个问题是可判定的，即可以用计算机程序在有限时间内通过对模型状态空间的穷举搜索来判断 φ 是否在 S 上被满足。如果 φ 在 S 上满足，则系统的正确性得以证实；否则，系统中存在错误，系统的正确性被证伪。

软件产品线是一组在公共核心资源的基础上按照规定的方式开发的软件密集系统的集合。这些系统共享一组公共的、可管理的、能够满足特定的市场或者任务需求的功能集合[10]。在实际中软件产品线可用特征(Feature)来描述，一条产品线可看作是一个有层次关系的特征的集合。所谓特征，是指软件系统或系统中用户可见的、显著的或与众不同的方面、品质或特点[11]。软件产品线上的产品之间既存在相同和相似的部分，也存在着差异。可变性则用于描述产品线上产品之间的差异，这是软件产品线工程的重要特征[12]。软件产品线中的产品可以看作是特征的集合，而软件产品线的模型检测是指基于产品线模型针对产品需求找到满足产品需求的产品[13]。

本文将软件产品线的可变性建模引入安全分析过程，利用特征模型作为系统故障的结构模型来刻画故障的层次与约束关系。同时通过对状态迁移系统的拓展提出一种故障标记迁移系统(Fault Labeled Transition System，FLTS)作为系统故障行为模型。然后基于故障标记迁移系统的语义定义了利用模型检测生成故障树的过程。最后，利用现有的软件产品线模型检测工具 SNIP[14-15] 实现了利用系统模型生成故障树的过程。

本文的主要贡献在于：

1)将软件产品线中可变性管理引入安全分析过程，将系统中的故障看作系统中可以进行配置的特征，从而为安全分析中刻画系统的结构和行为特性提供了一种直观、易理解的视角；

2)定义了一种基于状态迁移自动机的系统故障配置建模方法，可显式地描述和刻画系统故障间的约束关系以及不同系统故障配置下的系统行为；

3)基于故障标记迁移系统的语义，定义了基于系统模型针对特定安全属性生成故障树的模型检测过程；

4)利用现有的软件产品线模型检测器 SNIP 建立了根据系统模型针对特定安全属性生成故障树的过程，并通过实验验证了该方法的可行性。

本文第 2 节概述相关预备知识，包括故障树、线性时序逻辑(Linear-time Temporal Logic，LTL)[16]；第 3 节展示故障标记迁移系统；第 4 节介绍基于 FLTS 的语义，利用模型检测生成故障树的过程；第 5 节说明如何使用 SNIP 模型检测器的建模语言对故障行为进行描述；第 6 节结合具体实例展示实验的结果与分析；第 7 节讨论基于模型的安全分析的相关工作；最后总结全文并展望下一步的研究方向。

2　预备知识

本节介绍相关的预备知识，主要包括故障树的基本概念和时序逻辑。

2.1　故障树

故障树是一种特定的树状逻辑关系图，用故障事件、逻辑门等符号描述系统中各种事件之间的逻辑关系，刻画系统失效的原因。故障事件包括系统硬件、软件的故障和人为错误等事件；逻辑门描述了故障事件间的逻辑关系。故障树具体由以下几部分组成。

1)顶事件(Top Level Event，TLE)：代表系统的某种失效，表示为故障树的根节点；

2)底事件(基本事件)：代表系统构件的基本故障，表示为故障树的叶节点；

3)中间事件：代表未完全分解的故障事件，表示为故障树的中间节点；

4)逻辑门：描述事件之间因果关系的逻辑符号，例如，“与门”、“或门”、“表决门”等。故障树常见符号及其含义可参考文献[2]。

数据采集器系统(Acquisition)是基于模型的安全性领域中具有代表性的例子之一[17-18]。如图 1 所示，数据采集器系统包括监控模块(Monitor)、传感器模块(Sensors)以及过滤器模块(Filters)。其中传感器模块和过滤器均包含两个工作单元，分别对应优先模式(Primary Mode)和备用模式(Backup Mode)。数据采集器系统的工作流程如下：传感器模块采集数据，并将数据分别发送给过滤器和监控单元；过滤器模块接收传感器模块的数据，对该数据进行处理后将其发送给监控单元，同时将其作为数据采集器系统的输出；监控单元接收传感器模块和过滤器模块的数据，通过数据判断传感器模块和过滤器模块是否出现故障；如果某个模块出现故障则将该模块从优先模式切换到备用模式，同时发送警报。

图 2 所示为数据采集器系统输出错误的一个故障树示例。其中 *OutputError* 为顶层事件，表示系统的输出错误；*SensorError* 和 *FilterError* 为两个基本事件，分别表示传感器模块错误和过滤器模块错误。二者通过或门连接到顶层事件，表示二者都可以独立导致顶层事件发生。

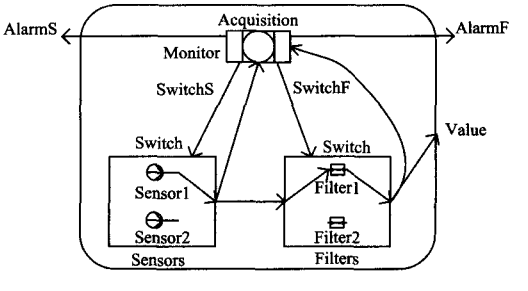


图 1　数据采集器系统

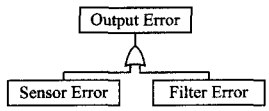


图 2 数据采集器系统故障树示例

割集(Cut Set,CS)是故障树中一些基本事件的集合;当这些基本事件同时发生时,顶层事件发生。对于某个割集,若其中任一底事件不发生时顶事件也不发生,则这样的割集称为最小割集(Minimal Cut Set,MCS)。因此最小割集表示了导致系统故障发生的最根本的基本故障事件组合。任意一棵故障树可以用一种简化的两层结构表示,底层表示所有最小割集,通过或门连接到顶层故障事件。以图 2 中故障树为例,割集为{*SensorError*},{*FilterError*}和{*SensorError*,*FilterError*}。最小割集为{*SensorError*}和{*FilterError*}。

2.2 线性时序逻辑

本文研究基于模型检测的故障树生成,采用线性时序逻辑描述的安全属性。

定义 1 线性时序逻辑(LTL)有下列用 Bakus Naur 范式给出的语法:

$$\varphi ::= \top \mid \bot \mid p \mid (\neg\varphi) \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi) \mid (\varphi \rightarrow \varphi) \mid (X\varphi) \mid (F\varphi) \mid (G\varphi) \mid (\varphi U\varphi) \mid (\varphi W\varphi) \mid (\varphi R\varphi)$$

其中,p 表示任意原子命题。

符号$\top$和$\bot$是 LTL 公式,它们都是原子命题,分别表示真(True)和假(False)。连接词 X,F,G,U,R 和 W 称为时序连接词(Temporal Connectives)。X 意为"下一个状态"(neXt),F 意为"某未来状态"(Future),G 意为"所有未来状态"(Globally)。接下来的 3 个连接词 U,R 和 W 分别表示"直到"(Until),"释放"(Release)和"弱-直到"(Weak-Until)。

LTL 公式可以转化为 Büchi 自动机(Büchi Automaton,BA)[19]。

定义 2 一个 *BA* 是一个元组$(\mathcal{Q},\Sigma,\delta,\mathcal{Q}_0,\mathcal{F})$。其中 $\mathcal{Q}$ 是一个状态集合,Σ 是符号表,$\delta \subseteq \mathcal{Q}\times\Sigma\times\mathcal{Q}$ 表示迁移关系,$\mathcal{Q}_0 \subseteq \mathcal{Q}$ 代表初始状态集合,$\mathcal{F} \subseteq \mathcal{Q}$ 表示接受状态集合。*BA* 接受可以无限次访问接受状态的字符串(Infinite Words)。

3 系统故障模型

本文研究基于模型的故障树生成方法,提出新的描述故障约束关系的模型——故障特征图(Fault Feature Diagram,FFD),以及描述包含系统的正常和故障行为的模型——故障标记迁移系统(Fault Labeled Transition System,FLTS)。下面对这两种模型进行介绍。

3.1 故障特征图

在复杂系统中存在多种故障,故障间存在如依赖、互斥等约束关系[20]。比如一个开关可能发生固定型故障(Stuck-At Fault),可能永久保持在闭合或断开状态无法切换,但是这两种故障状态不会同时出现在系统中。

为了保证故障树分析的准确性,须考虑故障之间的约束关系。为此,本文基于软件产品线的特征建模,提出故障特征图作为描述故障约束关系的模型。具体定义如下。

定义 3 故障特征图(Fault Feature Diagram,FFD)是一种以图的形式表示的基于故障特征集合 $\mathcal{F}$ 的树状特征模型。其中,树中每条边代表一个特征约束。特征约束分为或(or),可选(optional)、多选一(alternative)以及必选(mandatory)4 种,即:

$$root(f) \equiv f$$

$$mandatory(p,f) \equiv f \Leftrightarrow p$$

$$alternative(p,f_1,\cdots,f_n) \equiv ((f_1 \vee \cdots \vee f_n) \Leftrightarrow) \wedge \bigwedge_{i<j} \neg(f_i \wedge f_j) \text{ or } (p,\{f_1,\cdots,f_n\}) \equiv (f_1 \vee \cdots \vee f_n) \Leftrightarrow p$$

其中,p,f 和 f_i 为特征集合 $\mathcal{F}$ 中的元素。

下面以图 1 中的数据采集器系统对故障特征图进行说明。本文根据需要对原数据采集器系统进行了一些拓展,单个组件可能发生多种错误。而对于现有的故障树生成研究,往往在一次分析过程中单个组件只能注入一种故障。数据采集器系统各组成单元可能发生的故障如下:

1)传感器单元可能会发生数据变化从而偏离正常变化规律的故障(Drift)。例如正常情况下,传感器单元采集数据的初值为 1,每次数据采集过程中,数据值增加 1;当数据值达到 5 后重置为 1;如果发生偏离故障,则数据值每次增加 2,且不再进行重置。该故障为瞬时性故障(Transient Fault),即组件发生故障后可能会恢复正常。

2)传感器单元也可能发生固定型故障。当传感器发生瘫痪后,采集到的数据值会保持为一个固定的值不再发生变化。该故障一旦发生,其影响是永久性的(Permanent Fault),组件无法再恢复正常。

3)过滤器单元可能发生与传感器单元类似的固定型故障,过滤器单元的数据保持为固定数值而不再变化,该故障属于永久性故障。

数据采集器系统的故障特征图如图 3 所示。故障特征图是一种树状图,图中的每一个节点表示一个故障特征,其中根节点表示数据采集器系统的故障;叶节点表示系统元件的基本故障;中间节点表示系统模块的故障。故障特征图中的边刻画了故障特征间的约束关系,图 3 示例表示的是与关系、或关系等。图 3 中,特征 *S1* 和 *S2* 以与关系连接到特征 *SS*,表示传感器模块的故障包括优先和备份传感器单元的故障;特征 *S1DT* 和 *S1SP* 表示优先传感器可能发生的故障,二者均标记为可选且以与关系连接,表示在系统运行过程中二者互不影响,从而可能出现 3 种情况,即二者都发生、其中之一发生或都不发生;特征 *S2DT* 和 *S2SP* 也为可选故障,但与前面不同的是二者以异或关系连接到特征 *S2*,表示在系统的一次运行过程中,备份传感器单元的这两种故障不会同时出现。

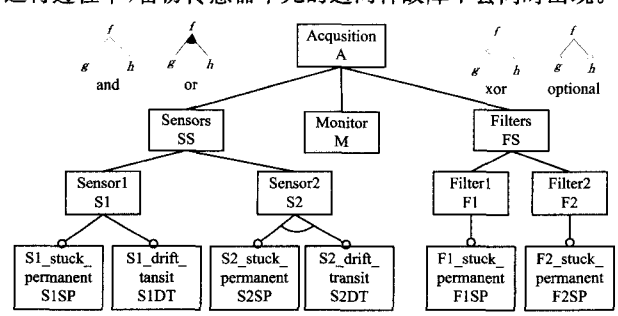


图 3 数据采集器系统故障特征图

现有的安全性分析技术在对故障建模时通常只是描述其起源、传播或具体行为,而没有对故障间的约束关系进行刻

画。通过图中节点与边的含义,故障特征图描述了故障特征的约束关系。一个满足故障特征图约束关系的故障特征集合称为系统的一个故障配置(Fault Configuration)。

定义 4　设 ffd 是一个系统的故障特征模型,$\mathcal{F}$ 是全部故障特征的集合。若存在特征集合 fc 满足 $fc\subseteq\mathcal{F}$ 且 fc 满足 ffd 中特征间的约束关系,则称 fc 是系统的一种故障配置(Fault Configuration)。故障特征图的语义即所有合法的系统故障配置的集合记作$[[ffd]]_{FFD}$,且$[[ffd]]_{FFD}\subseteq\{2^{\mathcal{F}}\}$。

以数据采集器系统为例,如图 3 所示,根据故障特征图的语法,数据采集器系统的故障特征集合为 $Fault=\{S1SP, S1DT, S2SP, S2DT, F1SP, F2SP\}$,而系统所有的故障配置集合为$\{2^{Fault}\}$。

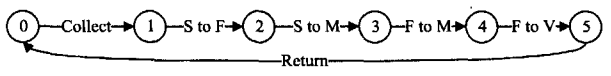
3.2　故障标记迁移系统

传统的模型检测过程通常采用状态迁移系统对系统行为进行建模。Kripke 结构[21]是一种常用的系统模型,包含系统状态和系统状态迁移关系。为了对包含故障特征的系统行为进行分析,本文对 Kripke 结构进行拓展并提出故障标记迁移系统(Fault Labeled Transition System, FLTS),通过在迁移关系上添加特征标记来区分迁移关系描述的行为。故障标记迁移系统的形式化定义如下。

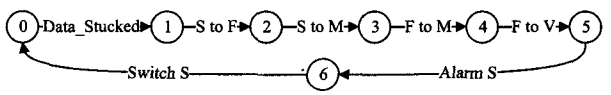
定义 5　一个 FLTS 是一个元组 $flts=(\mathcal{S},\mathcal{I},\mathcal{P},\mathcal{R},\mathcal{L},ffd,\gamma)$,其中:$\mathcal{S}$是一个有限状态集合;$\mathcal{I}\subseteq\mathcal{S}$是初始状态集合;$\mathcal{P}$是一个原子命题集合;$\mathcal{R}\subseteq\mathcal{S}\times\mathcal{S}$表示迁移关系;$\mathcal{L}:\mathcal{S}\to 2^{\mathcal{P}}$是一个标记函数;$ffd$ 是一个故障特征模型;$\gamma:\mathcal{R}\to(\{\bot,\top\}^{|N|}\to\{\bot,\top\})$是一个满函数,将每条迁移标记上一个故障特征表达式。

迁移关系 $\mathcal{R}$ 是满射,即对于每一个状态 s,都存在一个后继状态 s',记作 $\mathcal{R}(s,s')$。标记函数 $\mathcal{L}$在每个状态上标记出所有在该状态上为真的原子命题。FLTS 包含着一个故障特征图 ffd 用来描述故障间的约束关系并规定合法的系统故障配置集合。故障标记函数 γ 为每条迁移$(s,s')\in\mathcal{R}$标记一个故障特征表达式 f,f 刻画的系统故障组合发生时系统发生从当前状态 s 到达后继状态 s'的迁移关系。如果 $f=a\wedge b$,则表示需要故障 a,b 同时发生;若 $f=a\vee b$,则表示任意故障 a,b 可使系统发生该迁移关系;$f=\neg a\wedge b$ 表示当故障 a 不发生且故障 b 发生时使系统触发该迁移关系。

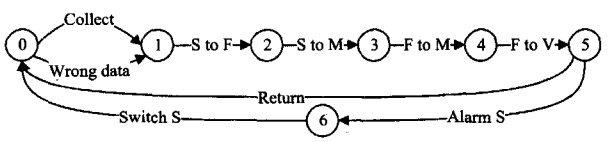
不同的故障会导致不同的系统行为。以图 1 所示的数据采集器系统为例,图 4 中的 3 个 Kripke 结构分别刻画了该系统的正常行为和两种故障行为。为方便理解,图中迁移上加注了标记以表明该迁移的含义,如 collect 表示传感器模块采集数据,S to F 表示由传感器模块向过滤器模块发送数据,Kripke 结构本身并不包含这些内容。图 4(a)表示系统的正常行为。状态 0 代表系统的初始状态,从状态 0 到状态 1 的迁移关系描述传感器模块采集数据的过程;由状态 1 到达状态 2 表示传感器模块将数据发送到过滤器模块;由状态 2 到达状态 3 表示传感器模块发送数据到监控模块;由状态 3 到达状态 4 表示过滤器模块将处理后的数据发送到监控模块;由状态 4 到达状态 5 表示将传感器模块处理后的数据作为系统输出;最后,系统返回状态 0 重新采集数据。图 4(b)描述了优先传感器单元发生永久固定型故障时的系统行为。当传感器模块采集数据时,由于优先传感器单元发生固定型故障,因此传感器模块采集到错误的数据。随后的系统行为与正常系统一致,但是当监控模块发现传感器模块数据错误时,系统不再直接返回初始状态来重新采集数据,而是首先发送警报,随后监控模块向传感器模块发送切换指令使传感器模块切换到备份单元,然后系统才重新采集数据。这个过程由图 4(b)中状态 5 到状态 6、状态 6 到状态 0 的迁移关系进行描述。图 4(c)描述了优先传感器单元发生瞬时性偏离故障时的行为。首先传感器模块在采集数据时可能采集正常数据也可能采集到发生偏离的数据,所以在状态 0 和状态 1 之间存在两条迁移关系分别表示这两种行为。当传感器模块采集到的数据为正常数据或数据值偏离没有超出数据值正常范围时,系统行为与正常系统一致。如果传感器模块采集到的数据值超出了正常范围,则后续系统行为与图 4(b)所示的系统行为相同。



(a)基本系统行为



(b)含故障特征 S1SP 的系统行为



(c)含故障特征 S1DT 的系统行为

图 4　数据采集器不同的系统行为

图 5 使用 Kripke 结构描述同时包含这两种故障的系统行为,但是从图中无法看出其中的迁移关系表示正常系统行为或是故障系统行为。在系统模型中添加特征标记能有效刻画故障对系统行为的影响,FLTS 在一张图中直观地描述了所有特征对系统行为的影响。图 6 所示为包含优先传感器单元两种故障特征的数据采集器系统的故障标记迁移系统。图中没有故障特征标记的迁移表示正常的系统行为,如状态 2 到状态 3 的迁移关系。而状态 5 到状态 6 的迁移特征标记表达式为 $S1SP\vee S1DT$,表示传感器单元的两种故障均可以导致该迁移发生,该迁移关系属于故障行为。

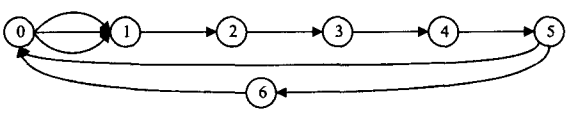


图 5　包含两种故障特征的 Kripke 结构

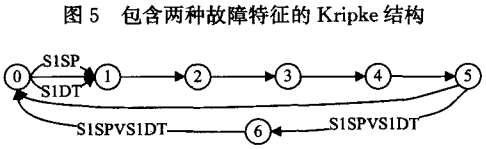


图 6　数据采集器系统的 FLTS 示例

在含义上,故障标记迁移系统可以看作是描述不同故障配置下系统行为的 Kripke 结构的聚合。作为一种参数化的模型,根据不同的故障配置可实例化为不同的行为模型,描述特定故障配置下的系统行为。

从 FLTS 实例化为特定故障配置的行为模型的过程可以用 FLTS 的投影(Projection)操作来定义。为了获得特定故

障配置的系统行为，FLTS 的一个投影需要关联到某个故障特征集合，记作 $fc\subseteq[[ffd]]_{FFD}$。投影操作从语法上看就是移除那些故障标记不属于 fc 的迁移。投影操作的结果是一个普通的 Kripke 结构。

定义 6 对于一个 FLTS 模型 $flts=(\mathcal{S},\mathcal{I},\mathcal{P},\mathcal{R},\mathcal{L},ffd,\gamma)$，以及一个故障配置 $fc\in[[ffd]]_{FFD}$，$flts$ 对 fc 的投影是一个 Kripke 结构 $\mathcal{M}=(S,\mathcal{I},\mathcal{P},\mathcal{R}',\mathcal{L})$，其中 $\mathcal{R}'=\{t\in\mathcal{R}\mid fc\in\gamma(t)\}$，记该投影为 $flts_{|fc}$。

如图 6 中的 FLTS，如果需要得到故障配置 $fc=\{S1DT\}$ 的投影，则需要删除故障特征 $S1SP$ 及其相关迁移。首先找到特征标记包含 $S1SP$ 的迁移，这个过程在图中表示为 $0\xrightarrow{S1SP}1$，5→6 和 6→0。$0\xrightarrow{S1SP}1$ 迁移如果删除故障特征 $S1SP$，则不满足迁移的激发条件，因此删除该迁移。而迁移 5→6 和 6→0 的故障标记为 $S1SP\vee S1DT$，在删除故障特征 $S1SP$ 后仍然满足激发条件，所以保留迁移 $S1DT$ 关系。最后去除故障特征后得到的投影与图 4(c)表示的 Kripke 结构一致。从实例中可以看出，一个故障标记迁移系统描述了其所有投影的行为，因此 FLTS 的语义是它所有投影的组合。

定义 7 $[[flts]]_{FLTS}=\bigcup_{fc\in[[ffd]]_{FFD}}[[flts_{|fc}]]_{\mathcal{M}}$

4 FLTS 的故障树生成

第 3 节定义了故障特征图和故障标记迁移系统，并结合实例对其进行了说明，随后定义了故障标记迁移系统的投影操作。本节在此基础上将给出针对线性时序逻辑描述的安全属性通过模型检测生成故障树的过程。

故障树生成本质上是发现所有能够导致顶层的故障组合，而软件产品线的模型检测会找到所有不满足产品需求的产品。两者的思想是一致的，所以本文采用 LTL 描述系统的安全属性，利用软件产品线模型检测方法对 FLTS 进行分析，进而针对特定安全属性生成故障树。模型检测过程的输入是一个故障标记迁移系统和一个用于描述系统安全的属性。本文利用模型检测针对 FLTS 分析生成故障树的本质是：根据 FLTS 针对特定安全属性计算全部割集，然后通过最小化得到最小割集。

对于 LTL 公式和 FLTS 模型，可以使用基于自动机的方式进行模型检测。假设 $flts$ 是一个系统模型，φ 是一个 LTL 公式表示的安全属性。首先对安全属性 φ 取反，然后将其转化为 Büchi 自动机 $BA(\neg\varphi)$。最后，将 $flts$ 和 $BA(\neg\varphi)$ 通过组合操作(Composition)得到一个拓展的 FLTS。

定义 8 对于一个故障标记迁移系统 $flts=(\mathcal{S},\mathcal{I},\mathcal{R},\mathcal{L},fd,\gamma)$ 和一个 $BA\ a=(\mathcal{Q},2^{\mathcal{P}},\delta,\mathcal{Q}_0,\mathcal{F})$，其组合操作 $flts\otimes a$ 的结果仍然是一个 FLTS，$flts'=(\mathcal{S}\times\mathcal{Q},\mathcal{I}',\mathcal{P}',\mathcal{L}',\mathrm{ffd},\gamma')$，其中：

1)$\mathcal{P}'=\mathcal{Q}$ 且 $\mathcal{L}'(s,q)=q$，$flts'$ 中的状态的标记是 BA 中的状态；

2)$(s,q)\rightarrow'(t,p)$，当且仅当 $s\rightarrow t$ 且 $q\rightarrow p$；

3)$\mathcal{I}'=\{(s_0,q)\mid s_0\in\mathcal{I}\wedge\exists q_0\in\mathcal{Q}_0\cdot(q_0,\mathcal{L}(s_0),q\in\delta)$。

Büchi 自动机 a 中 $\mathcal{F}$ 表示接受状态，即违背了安全属性的状态[22]。$flts'$ 中状态上的标记是 a 中的状态，所以 $flts'$ 中的接受状态是标记有 $\mathcal{F}$ 中接受状态的那些状态。$flts'$ 的接受状态表示顶层事件对应的系统故障状态集合，即 $TLE=\{s\in S\times\mathcal{Q}\mid\mathcal{L}'(s)\in\mathcal{F}\}$。因为 $flts'$ 仍然是一个 FLTS，所以它的语义仍然是它表示的所有路径 $[[flts']]_{FLTS}$。这样，验证 $flts$ 是否违背安全属性 φ 的问题就转化为在拓展的 FLTS $flts'$ 上是否存在一条可接受的执行路径。进一步，就是一个“不安全”的状态 s 在 $flts'$ 上是否可以被无限次地访问，其中 $s\in TLE$。如果不存在这样的路径，则表示系统满足安全属性；否则，系统不满足安全属性，而该路径上的所有故障特征的集合就是一个割集。以数据采集器系统为例，设顶层事件为“传感器模块发生失效”，LTL 公式表示为 $\langle\rangle value\geqslant 15$。如图 6 所示，状态 6 为顶层事件对应的故障状态。从初始状态出发，首先经过迁移 $0\xrightarrow{S1SP}1$，数据采集器模块采集了错误的数据，随后经过迁移 $1\rightarrow\cdots\rightarrow5$ 使得系统输出发生错误。然后系统发送警报，经迁移 $5\xrightarrow{S1SP}6$ 到达状态 6。因此在系统中存在一条到达该故障状态的路径 $0\xrightarrow{S1SP}1\rightarrow\cdots\rightarrow5\xrightarrow{S1SP}6$，这条路径上包含的故障特征为 $S1SP$，所以 $\{S1SP\}$ 是导致顶层事件发生的一个割集。基于上述观察，FLTS 中割集的形式化定义如下。

定义 9 $flts=(\mathcal{S},\mathcal{I},\mathcal{P},\mathcal{R},\mathcal{L},ffd,\gamma)$ 是一个系统，其中 ffd 是一个故障特征模型，$\mathcal{F}$ 表示所有故障特征的集合。$fc\subseteq\mathcal{F}$ 是一个故障配置。如果 $flts_{|fc}$ 中存在一条路径 $\pi=s_0s_1,\cdots,s_k$，其中 $s_k\in TLE$，则将 fc 称为 TLE 的一个割集，记作 $cs(FC,TLE)$。

系统故障树的一个割集代表了该系统发生故障的一种可能性，即一种失效模式。由于最小割集发生时顶事件必然发生，因此一个故障树的全部最小割集就代表了顶事件发生的所有可能性，即系统的全部故障模式。最小割集显示了处于故障状态的系统所必须修复的基本故障，展示出的是系统的最薄弱环节。通过在系统设计中隔离某个割集的故障，避免这些故障同时发生，可以提高系统的安全性。最小割集的形式化定义如下。

定义 10 $\mathcal{F}$ 表示所有故障特征的集合。令 $FC=2^{\mathcal{F}}$ 表示所有故障配置的集合，TLE 表示系统故障状态集合。割集和最小割集的表示如下：

$$CS(TLE)=\{fc\in FC\mid cs(fc,TLE)\}$$

$$MCS(TLE)=\{cs\in CS(TLE)\mid\forall cs'\in CS(TLE)(cs'\subseteq cs\Rightarrow cs'=cs\}$$

基于不同的“最小化”的定义，最小割集的概念可以被拓展为意义更广泛的质蕴涵集[23]。后者表示的集合不止包括发生的故障，也包含不发生的故障。

基于上述过程，给定一个 FLTS $flts$ 和一个表示安全属性 LTL 公式 φ，利用模型检测生成故障树的过程可归纳为以下步骤：

1)得到拓展的 FLTS $flts'=flts\otimes BA(\neg\varphi)$，其中 $TLE=\{s\in\mathcal{S}\times\mathcal{Q}\mid\mathcal{L}'(s)\in\mathcal{F}\}$；

2)找到所有从初始状态到达 TLC 的路径，进而得到顶层事件的全部割集 $CS(TLE)$；

3)通过对 $CS(TLE)$ 的最小化处理，得到全部最小割集 $MCS(TLE)$，过程参照文献[24]；

4)按照故障树的符号定义将最小割集组织成树状结构。

故障树的本质是对所有导致顶层事件发生的故障组合的刻画，而软件产品线模型检测可以找到所有产品需求的产品。本文利用了二者的这种特质，将软件产品线模型检测应用到故障树的生成。本节基于 *FLTS* 的定义，利用 *LTL* 描述系统安全属性，给出了割集和最小割集的形式化定义，随后归纳了利用基于自动机的模型检测求解割集与最小割集的过程。第 5 节将介绍现有的模型检测工具 SNIP[14-15] 并介绍如何利用 SNIP 对系统故障行为进行建模。

5　系统建模

本文在 SNIP 的基础上实现了第 4 节介绍的故障树分析过程，本节简要介绍在 SNIP 中如何对 FLTS 进行描述。

FLTS 和特征图是语义模型，为了对实际系统进行建模，需要高级建模语言。SNIP 是由 A. Classen 等开发的一款针对软件产品线的模型检测工具。传统的模型检测器只能检测特定系统故障配置下的系统行为，而 SNIP 可以同时对多种故障配置下的系统行为进行分析。SNIP 实现了软件产品线的模型检测过程。在给定产品线模型和产品需求的情况下，如果所有产品满足产品需求，检测结果为 True，否则 SNIP 可以找到所有不满足产品需求的产品。SNIP 同样采用特征模型对软件产品线进行建模，所以 SNIP 的检测结果为产品的特征集合。SNIP 在建立系统模型时依赖于两种高级建模语言：TVL(Textual Variability Language)[25] 和 fPromela[14]，其中 TVL 用于描述可变性配置，即故障特征模型；fPromela 用于描述系统正常行为与故障行为。

5.1　故障特征模型

TVL 是一种文本形式的特征模型描述语言，能够有效地描述故障特征的约束关系，同时具有良好的易读性。此外 TVL 具有严格的语义，支持进行形式化的分析。使用 TVL 语言描述第 2 节中图 3 表示的故障特征模型，如表 1 所列。

表 1　数据采集器故障特征模型

```
Fault Feature Model of Acquisition System
1. root Acquisition group allof{
2.     Monitor,
3.     Sensors group [*..*]{
4.         Sensor1 group [0..2]{
5.             opt S1_stuck_permanent,
6.             opt S1_drift_transient
7.         }
8.         Sensor2 group oneof{
9.             opt S2_stuck_permanent,
10. opt S2_drift_transient
11.         }
12.     }
13.     Filters gruop allof{
14.         Filter1 group [1..1]{
15.             opt F1_stuck_permanent
16.         }
17.         Filter2 group [0..*]{
18.             F2_stuck_permanent
19.         }
20.     }
21. }
```

在表 1 中，故障特征模型从根特征的声明开始(第 1 行)，根特征 Acquisition 被分解为 3 个子特征：Monitor，Sensors 和 Filters，表示数据采集器系统包含这 3 个模块。特征的分解通过关键词“group”引导，随后是分解的类型。故障特征模型中的与(and)、或(or)和异或(xor)分解类型用“allof”，“someof”和“oneof”表示。这些关键词也可通过基数形式表示，比如第 3 行的“[*..*]”、第 14 行的“[1..1]”与“allof”等价。大括号中为分解后的一系列子特征，通过逗号分隔开来。如果一个特征是可选的，可通过“opt”关键字表示。

5.2　故障行为模型

fPromela 语言是一种用于对有限状态系统进行建模的形式化描述语言，用于描述系统行为。fPromela 的核心数据结构是进程，在系统设计过程中，进程往往用来对系统的某个组件进行建模。fPromela 语言的语法与广泛使用的模型检测器 SPIN[26] 使用的 Promela 语言十分相似，允许动态创建并行的进程，并可以在进程之间通过消息通道进行同步和异步通信。与 Promela 不同的是，fPromela 加入了特征(feature)元素作为标记以表明特征对系统行为的影响。fPromela 代码在语义上对应于特征迁移系统(Feature Transition System，FTS)[15]。本文提出 FLTS 将故障视作可配置的特征，从产品线配置的角度分析故障对系统行为的影响，所以 FLTS 和特征迁移系统在语义上是等价的。因此本文利用 fPromela 语言对 FLTS 进行建模，使用 SNIP 进行模型检测。

一般来说，安全分析将故障分为永久性故障(Permanent Fault)和瞬时性故障(Transient Fault)。前者表示组件一旦发生故障就无法恢复，而后者则表示组件发生故障后可恢复且可能再次发生故障[27]。比如电力系统中由于大风引起的短时碰线导致的短路故障为瞬时性故障，而一些由于组件结构上的损坏导致设备无法运行则属于永久性故障。

故障是组件的属性，但是故障的发生是组件的动态行为。故障在一定触发条件下被激活，对系统行为和状态产生影响。因此，对故障进行建模需要描述故障的触发条件和故障的影响。相应地，为了在系统模型中描述故障行为，对于每一个故障，增加一个故障控制变量，用于控制故障行为是否发生。同时在组件中添加一个状态变量，用于表示组件当前处于正常状态还是某个故障状态。描述两种故障类型的 fPromela 代码片段如表 2 和表 3 所列，这两种描述故障行为的方式可以看作描述两种常见故障分类的模式。这种模式有助于降低故障行为描述的难度，并为自动化的建模过程提供帮助。

表 2 中第 1 行首先实例化了一个特征变量。第 2 行通过 proctype 关键字定义一个进程用于描述某个组件行为。fault_permanent_control变量用于控制故障行为的发生，component_state变量用于表明组件当前的状态。第 5－9 行表示当组件正常时随时可能发生故障。第 10－13 行表示如果系统具有故障特征，系统组件变为故障状态。第 15－16 行表示这是一个永久性故障，组件一旦处于故障状态就无法恢复。component_state 变量是一个整型或枚举变量，根据不同故障赋值不同。第 19－23 行表示组件处于不同状态下执行不同的行为，因此通过这种方式可以对同一组件同时定义多种故障。

表 3 中瞬时型故障与永久性故障的描述过程整体相同，只是组件处于故障状态后仍然有可能恢复到正常状态(第 15－20 行)。

表 2 永久型故障 fPromela 代码

```
Permanent Fault
1. Features f;
2. active proctype component{
3.     bool fault_permanent_control=false;
4.     int component_state;
5.     if
6.         ::component_state==0;
7.           if
8.           ::fault_permanent_control=false;
9.           ::fault_permanent_control=true;
10.              if::f. Fault;
11.                  component_state=1;
12.              ::else->skip;
13.              fi;
14.          fi;
15.    ::component_state==1;
16.      skip;
17.    fi;
18.    if
19.    ::component_state==0;
20.      Nomal Component Behavior;
21.    ::component_state==1;
22.        Fault Behavior;
23.    fi;
24. }
```

表 3 瞬时型故障 fPromela 代码

```
Transient Fault
1. eatures f;
2. active proctype component{
3.     bool fault_permanent_control=false;
4.     int component_state;
5.     if
6.     ::component_state==0;
7.       if
8.       ::fault_permanent_control=false;
9.       ::fault_permanent_control=true;
10.         if :: f. Fault;
11.             component_state=1;
12.           :: else->skip;
13.         fi;
14.       fi;
15.    ::component_state==1;
16.  if
17.  ::fault_permanent_control=false;
18.    component_state=0;
19.    ::fault_permanent_control=true;
20.    fi;
21.    fi;
22.    if
23.    ::component_state==0;
24.      Nomal Component Behavior;
25.    ::component_state==1;
26.      Fault Behavior;
27.    fi;
28. }
```

6 实验分析

本节主要以第 2.1 节提到的数据采集器系统为例来展示所提的基于故障配置的故障树生成方法的可行性。

6.1 实验环境

本文使用 SNIP 模型检测器对 2.1 节中的数据采集器系统进行分析，进而针对特定安全属性计算最小割集并生成故障树。运行平台为 Intel i5 M480 2.67GHz＋4G RAM＋deepin 2014.3。

6.2 实验过程及分析

SNIP 模型检测器的输入包括 3 个部分，分别为故障特征图模型、故障行为模型以及系统安全属性。其中数据采集器系统的故障特征图模型如表 1 所列。

以数据采集器中的优先传感单元为例，采用第 5.2 节介绍的故障描述模式同时对两种故障行为进行描述的 fPromela 模型如表 4 所列。第 1 行的 sensor_state 变量用于表示优先传感器当前状态，1 表示正常，2 表示 drift 故障状态，3 表示 stuck 状态。第 2，3 行中的 drift_control 和 stuck_contorl 变量分别用于控制两种故障的发生。第 4 行的 mode 变量表示传感器模块处于优先或备份模式。第 5 行中的 CMode 为 channel 变量，监控模块通过该信道向传感器单元发送模式信息，然后判断当前模式是否为优先模式，如果不是，则后续行为不会执行。第 6－17 行表示如果系统包含 S1_drift_transient故障特征，当系统没有发生永久 stuck 故障时，系统可能发生 drift 故障，也可能恢复；如果已经发生 stuck 故障，则永久处于 stuck 故障状态。第 19－30 行描述了 S1_stuck_permanent 故障特征对系统行为的影响，当 stuck 故障没有发生时，不影响系统的行为，系统仍然可能正常运行或发生 drift 故障，但当 stuck 故障发生后，则会覆盖 drift 故障，系统永久处于 stuck 故障状态。第 33－40 行表示传感器单元在不同状态时执行不同的系统行为。

表 4 优先传感器单元 fPromela 代码

```
Primary Sensor fPromela Code
1. sensor_state=1;
2. bool stuck_control=false;
3. bool drift_control=false;
4. mtype mode=primary;
5. do::{CMode?mode;&&. mode==primary;}
6.    if
7.    ::f. S1_drift_transient;
8.      if
9.      ::sensor_state!=3;
10.       if
11.       ::drift_control=false;
12.         sensor_state=1;
13.       ::drift_control=true;
14.         sensor_state=2;
15.       fi;
16.     ::sensor_state==3;
17.       skip;
18.     fi;
19.   ::f. S1_stuck_permanent;
20.     if
21.     ::sensor_state !=3;
22.       if
23.       ::stuck_control=false;
24.         skip;
25.       ::stuck_control=true;
26.         sensor_state=3;
27.       fi;
28.   ::sensor_state==3;
29.     skip;
30.   fi;
31.   ::else-> sensor_state=1;
32.   fi;
33.   if
34.   ::sensor_state==1;
35.     Normal Behavior;
36.   ::sensor_state==2;
37.     Drift Fault Behavior;
38.   ::sensor_state==3;
39.     Stuck Fault Behavior;
40.     fi;
```

系统安全属性以 LTL 公式的方式输入，本文对数据采集

器系统验证以下 4 个安全属性。

(#1)$G(0<value\leqslant 5)$：该安全属性表示系统采集到的数据永远处于正常范围。

(#2)$G(value\leqslant 5)$：表示传感器模块不会发生故障。

(#3)$!F(sensor.mode==backup\&\&value\geqslant 15)$：表示当传感器模块处于备份模式时，备份传感器单元不会出现故障。简言之，就是优先传感器单元和备份传奇单元不会都发生故障。

(#4)$!F(filter.mode==backup\&\&value==0)$：与属性(#2)类似，即过滤器模块不会完全失效。

使用 SNIP 验证数据采集器模型并针对上述安全属性求解最小割集，结果如表 5 所列。

表 5　安全属性实例及对应最小割集

公式	最小割集
(#1)	{S1DT},{S1SP}
(#2)	{F1DP},{F2DP}{S1DT},{S1SP}
(#3)	{S1DT,S2DT}{S1DT,S2SP}{S1SP,S2DT}{S1SP,S2SP}
(#4)	{F1DP,F2DP}

表 5 中，公式(#2)被证伪对应的顶层事件为“优先传感器单元发生失效”。图 6 为包含了优先传感器单元两种故障特征的 FLTS，其中状态 6 为式(1)表示的顶层事件对应的故障状态。从图 4 中可以找到从初始状态 0 到达状态 6 的路径 $0\xrightarrow{S1SP}1\rightarrow\cdots\rightarrow5\xrightarrow{S1SP\vee S1DT}6$ 和 $0\xrightarrow{S1DT}1\rightarrow\cdots\rightarrow5\xrightarrow{S1SP\vee S1DT}6$，所以顶层事件对应的割集为 $\{S1DT\}$，$\{S1SP\}$ 和 $\{S1SP, S1DT\}$，而最小割集是 $\{S1DT\}$ 和 $\{S1SP\}$。图 7 为式(1)表示的顶层事件的故障树。

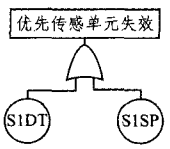


图 7　传感器失效的故障树

公式(#3)被证伪表示的顶层事件为“传感器模块失效两次”，即优先传感器单元失效后备份传感器单元再次失效。备份传感器的 FLTS 和优先传感器单元是相似的，只是故障特征对应地从 $S1SP$ 和 $S1DT$ 变为 $S2SP$ 和 $S2DT$。因此顶层事件可以理解为优先传感器单元失效使系统从初始状态到达状态 6，随后系统将传感器模块切换到备份单元后再次到达状态 6。系统从初始状态第一次到达状态 6 的过程与式(1)表示的顶层事件相同，系统再次到达状态 6 的路径为 $0\xrightarrow{S2SP}1\rightarrow\cdots\rightarrow5\xrightarrow{S2SP\vee S2DT}6$ 和 $0\xrightarrow{S2DT}1\rightarrow\cdots\rightarrow5\xrightarrow{S2SP\vee S2DT}6$，所以最小割集如表 4 所列，故障树如图 8 所示。

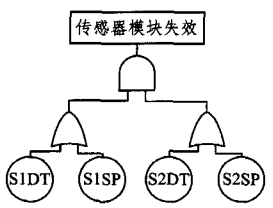


图 8　传感器失效两次的故障树

从实验结果可以看出，本文将故障视为可配置的特征，利用故障特征图刻画故障特征约束关系，使用 FLTS 描述系统和故障行为，通过软件产品线模型检测求解最小割集生成故障树是可行的。基于模型的准确语义和模型检测的自动化状态空间搜索，故障树生成的准确性得到了提高，同时减少了人工的工作量。

6.3　时间性能分析

为了分析故障注入及不同故障类型对系统规模和故障树生成过程性能的影响，本文在数据采集器系统原型的基础上通过改变故障类型得到了不同的变种系统，如表 6 所列。各系统状态空间及状态空间搜索过程使用的时间和内存情况如表 7 所列。

表 6　不同的数据采集器系统变种

系统编号	系统说明
1	正常系统，不包含任何故障特征
2	所有故障特征均为永久性故障
3	6.2 节所用系统
4	所有故障特征全部为瞬时性故障

表 7　状态空间及性能分析

系统编号	系统状态	搜索状态	时间/s	内存/kB
1	131	131	0.64	31208
2	29553	87848	1.27	38144
3	40455	103646	1.34	41132
4	48009	128975	1.46	44238

表 7 中系统状态数目表示系统状态空间大小，搜索状态数为搜索过程中实际访问系统的状态数目，因为在模型检测过程中需搜索所有可以导致顶层事件发生的路径，部分状态会被多次访问。从表中的数据可以看出，在系统中注入故障特征会对系统状态空间造成显著影响，进而增加了搜索所有执行路径需要的时间和内存，而且瞬时性故障的影响更大。

7　相关工作

故障树是安全分析的重要内容。近些年对故障树相关的研究工作主要集中于两个方向，分别为故障树的分析和故障树的生成。在故障树分析方面，文献[28]提出以二叉判定图对故障树进行存储，并且通过二叉判定图的操作计算故障树的最小割集与质蕴含集。利用这种方式，大幅缩减了故障树的存储空间，提高了故障树分析的效率。但是由于故障树底事件的排序问题，这种方法的效率仍然有待提高。Joanne Bechta Dugan 等人[29]针对传统故障树无法描述故障事件时序关系的缺陷对故障树进行了拓展，提出了动态故障树，随后又开发了动态故障树建模与分析工具 Galileo[30]。其主要特点是加入了动态逻辑门，引入了动态逻辑，能更准确地描述故障事件的优先关系与时序关系。但是由于传统的人工故障树生成过程很难生成正确的动态故障树和其符号表示带来的高复杂性，动态故障树在实际工程领域并没有得到十分广泛的应用。Kwang Yong Koh 和 Poong Hyun Seong 在文献[7]中提出利用模型检测化简并验证某个安全属性的故障树。利用 SMV 语言建立系统模型，通过时序逻辑描述安全属性，根据模型检测的结果删除故障树中冗余节点和错误节点从而得到正确的故障树。但是这种方法仍然十分繁琐，缺乏人性化的建模方式，很难在传统安全分析领域进行推广。文献[31]提出了一种利用逆向 Petri 网(Revrse Petri Nets)生成故障树最小割集的方法，有效提高了求解效率，但是构造 Petri 网的过程相对复杂。

本文主要关注故障树的生成。在这方面，日本的 Aref Majdara 和 Toshio Wakabayashi 在文献[32-33]中提出了一种基于组件模型的故障树自动生成过程。该方法将系统分解为组件集合和组件间的数据流集合，每个组件通过功能表和状态迁移表进行描述，然后通过一种回溯的搜索过程自顶层故障事件搜索生成故障树。这种方法简单、清晰、易于理解。组件模型可以存储为组件库以提高复用性。但是这种方法对系统的抽象过于简单，忽略了安全分析中系统的很多重要行为，同时也不适合对大规模复杂系统进行分析。

作为欧盟研究项目 ESACS[34]（Enhanced Safety Assessment for Complex Systems）和 ISAAC[6]（Improvement of Safety Activities on Aeronautical Complex Systems）中的一部分，意大利 Fondazione Bruno Kessler 研究中心的 Alessandro Cimatti 和 Marco Bozzano 领导的研究小组开发了采用符号化模型检测进行故障树生成的 FSAP/NuSMV-SA 平台[35-36]。FSAP 采用 NuSMV 模型检测器进行故障树生成，对于给定的顶层事件，通过扩展模型所描述的系统状态空间的穷举搜索，提取由能够导致失效的基本事件组成所有的最小割集。相对于手工故障树分析而言，这样的方法能够给出更加准确和完备的故障树。FSAP 的工作内容与本文十分相似，但是本文是从故障配置的角度对系统进行建模。FSAP 方法所存在的问题是只提供了对简单故障的建模，不支持灵活构件外部故障行为的描述。

英国郝尔大学的 Yiannis Papadopoulos 团队开发的 HiP-HOPS（Hierarchically Performed Hazard Origin and Propagation Studies）工具[37-38]通过对构件失效及其传播方式的分析实现了对复杂系统从系统功能级到底层构件级的安全性评估。HiP-HOPS 工具的优点在于系统的层次结构在故障树中有明显的体现，便于理解。但其主要问题是在故障树的生成过程中并没有考虑构件本身的行为，缺乏故障间的时序依赖关系和故障路径等动态信息。

近些年，国内也进行了一些利用形式化方法对故障树进行分析和构建的工作。文献[39]运用线性时序逻辑对故障树进行形式化规约，从中抽取软件安全属性并用时序逻辑进行描述，用以支持对安全关键软件的模型检测。文献[40]利用本体描述语言构建故障树领域本体，在不影响系统故障定位的情况下利用推理机解决故障树的重复构建问题。

与上述基于模型的安全分析和故障树生成工作相比，本文的基于故障配置的故障树方法将故障视作系统可管理、可配置的特征，并规定了严格的语法与语义。在对系统故障及其行为进行了建模时，可有效描述故障间的约束关系。另外，本文的方法从配置的角度对系统故障进行了建模，可同时对组件的多个故障进行建模与分析。

结束语 传统安全分析在应对安全关键系统的庞大规模和高复杂度时缺乏有效的解决方案。本文在将产品线中可变性管理概念引入安全分析的基础上，提出了一种对系统故障和系统行为进行建模的方法，并进一步给出了通过模型检测针对安全属性求解最小割集的算法。这种方法不同于传统安全分析手段，利用了现有的模型检测工具，提高了安全分析的效率。

在未来的工作中，主要对以下几个方向进行研究：1）文中提到目前的算法在求解非单调故障树时存在一些缺陷，因此需要对故障树生成算法进行改进；2）目前的系统模型只能描述离散的系统行为，将来会对连续时间的系统行为进行研究；3）目前的故障模型是布尔逻辑的，将来可以拓展到多值逻辑下的故障树分析；4）本文基于模型检测生成故障树，而状态空间爆炸是模型检测对大规模系统进行验证时遇到的重要问题，因此未来还将针对故障树的生成研究模型抽象的方法，提高分析效率[41]。除此之外，我们会进一步将安全分析从单个系统拓展到可配置的系统产品线上，也会研究将故障树拓展为动态故障树，同时对自动化的建模过程进行一定的探索。

参 考 文 献

[1] JAHANIAN F, MOK A K. Safety analysis of timing properties in real-time systems[J]. IEEE Transactions on Software Engineering, 1986, 12(9): 890-904.

[2] VESELY W E, GOLDBERG F F, ROBERTS N H, et al. Fault tree handbook[R]. DTIC Document, 1981.

[3] LIGGESMEYER P, ROTHFELDER M. Improving system reliability with automatic fault tree generation[C]// The Twenty-Eighth Annual International Symposium on Fault-Tolerant Computing (FTCS 28). Piscataway, N. J.: IEEE Computer Society, 1998: 90-98.

[4] JOSHI A, MILLER S P, WHALEN M, et al. A proposal for model-based safety analysis[C]// Proceedings of Digital Avionics Systems Conference 2005 (DASC 2005). Piscataway, N. J.: IEEE, 2005.

[5] BeERNARD R, AUBERT J J, BIEBER P, et al. Experiments in Model Based Safety Analysis: Flight ontrols[J]. Dependable Control of Discrete Systems, 2007, 1(1): 43-48.

[6] AKERLUND O, BIEBER P, BOEDE E, et al. ISAAC, a framework for integrated safety analysis of functional, geometrical and human aspects[C]// Proceedings of the European Congress on Embedded Real Time Software (ERTS 2006). Berlin Heidelberg: Springer, 2006: 1-11.

[7] KOH K Y, SEONG P H. SMV model-based safety analysis of software requirements[J]. Reliability Engineering & System Safety, 2009, 94(2): 320-331.

[8] LISAGOR O, KELLY T, NIU R. Model-based safety assessment: Review of the discipline and its challenges[C]// Proceedings of 9th International Conference on Reliability, Maintainability and Safety (ICRMS 2011). Piscataway, N. J.: IEEE, 2011: 625-632.

[9] CLARKE E M, GRUMBERG O, PELED D. Model checking[M]. Boston, USA: MIT press, 1999.

[10] NORTHROP L M. SEI's software product line tenets[J]. IEEE Software, 2002, 19(4): 32-40.

[11] KANG K, COHEN S G, HESS J A, et al. Feature Oriented Domain Analysis (FODA)-Feasibility Study[J]. Feature-Oriented Domain Analysis (FODA) Feasibility Study, 1990, 4(4): 206-207.

[12] NIE K M, ZHANG Li, FAN Z Q. Systematic Literarture Review of Software Product Line Variability Modeling Techniques[J]. Journal of Software, 2013, 24(9): 2001-2019. (in Chinese)

聂坤明，张莉，樊志强. 软件产品线可变性建模技术系统综述[J]. 软件学报，2013，24(9)：2001-2019.

[13] CORDY M，SCHOBBENS P Y，HEYMANS P，et al. Towards an incremental automata-based approach for software product-line model checking[C]// Proceedings of 16th International Software Product Line Conference (SPLC 12). Los Angeles，CA，USA：ACM，2012：74-81.

[14] CLASSEN A，CORDY M，HEYMANS P，et al. Model checking software product lines with SNIP[J]. International Journal on Software Tools for Technology Transfer，2012，14(5)：589-612.

[15] CLASSEN A，HEYMANS P，SCHOBBENS P，et al. Model checking lots of systems：efficient verification of temporal properties in software product lines[C]// Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE 2010). Los Angeles，CA，USA：ACM，2010：335-344.

[16] PNUELI A. The Temporal Logic of Programs[C]// Proceedings of 18th Annual Symposium on Foundations of Computer Science. Piscataway，N. J.：IEEE，1977：46-57.

[17] BOZZANO M，CIMATTI A，KATOEN J P，et al. Safety，Dependability and Performance Analysis of Extended AADL Models[J]. Computer Journal，2011，54(5)：754-775.

[18] Nguyen V Y. Trustworthy Spacecraft Design Using Formal Methods[R]. RWTH Aachen，2013.

[19] SISTLA A P，VARDI M Y，WOLPER P. The complementation problem for Büchi automata with applications to temporal logic[J]. Theoretical Computer Science，1987，49(2)：217-237.

[20] VAURIO J K. Treatment of general dependencies in system fault-tree and risk analysis[J]. IEEE Transactions on Reliability，2002，51(3)：278-287.

[21] BROWNE M C，CLARKE E M，GRÜMBERG O. Characterizing finite Kripke structures in propositional temporal logic[J]. Theoretical Computer Science，1988，59(1)：115-131.

[22] GASTIN P，ODDOUX D. Fast LTL to Büchi Automata Translation[J]. Lecture Notes in Computerence，2002，2102：53-65.

[23] KEAN A，TSIKNIS G. An incremental method for generating prime implicants/implicates[J]. Journal of Symbolic Computation，1990，9(2)：185-206.

[24] RAUZY A，DUTUIT Y. Exact and truncated computations of prime implicants of coherent and non-coherent fault trees within Aralia[J]. Reliability Engineering & System Safety，1997，58(2)：127-144.

[25] CLASSEN A，BOUCHER Q，HEYMANS P. A text-based approach to feature modelling：Syntax and semantics of TVL[J]. Science of Computer Programming，2011，76(12)：1130-1143.

[26] HOLZMANN G J. The SPIN model checker：Primer and reference manual[M]. Boston，USA：Addison-Wesley Reading，2004.

[27] D'ANGELO S，METRA C，SECHI G. Transient and permanent fault diagnosis for FPGA-based TMR systems[C]// Proceedings of 14th International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT 99). Piscataway，N. J.：IEEE，1999：330-338.

[28] RAUZY A. New algorithms for fault tree analysis[J]. Reliability Engineering & System Safety，1993，40(3)：203-211.

[29] DUGAN J B，BAVUSO S J，BOYD M A. Dynamic fault-tree models for fault-tolerant computer systems[J]. IEEE Transactions on Reliability，1992，41(3)：363-377.

[30] DUGAN J B，SULLIVAN K J，COPPIT D. Developing a low-cost high-quality software tool for dynamic fault-tree analysis[J]. IEEE Transactions on Reliability，2000，49(1)：49-59

[31] MAKAJI Ć NIKOLIĆD，VUJOŠUJOŠEVIC M，NIKOLIČ N. Minimal cut sets of a coherent fault tree generation using reverse Petri nets[J]. Optimization，2013，62(8)：1069-1087.

[32] MAJDARA A，WAKABAYASHI T. A new approach for computer-aided fault tree generation[C]// Proceedings of 3rd Annual IEEE Systems Conference. Piscataway，N. J.：IEEE，2009：308-312.

[33] MAJDARA A，WAKABAYASHI T. Component-based modeling of systems for automated fault tree generation[J]. Reliability Engineering & System Safety，2009，94(6)：1076-1086.

[34] BOZZANO M，VILLAFIORITA A，ÅKERLUND O，et al. ESACS：an integrated methodology for design and safety analysis of complex systems[C]// Proceedings of the 13th Annual European Safety and Reliability Conference (ESREL 2003). Boca Raton，Florida：CRC Press，2003：237-245.

[35] BOZZANO M，CAVALLO A，CIFALDI M，et al. Improving Safety Assessment of Complex Systems：An Industrial Case Study[C]// Proceedings of International Symposium of Formal Methods Europe 2003 (FME 2003). Berlin Heidelberg：Springer，2003：208-222.

[36] BOZZANO M，VILLAFIORITA A. Improving System Reliability via Model Checking：The FSAP/NuSMV-SA Safety Analysis Platform[C]// Proceedings of Computer Safety，Reliability，and Security，22nd International Conference (SAFECOMP 2003). Berlin Heidelberg：Springer，2003：49-62.

[37] PAPADOPOULOS Y，MCDERMID J，SASSE R，et al. Analysis and synthesis of the behaviour of complex programmable electronic systems in conditions of failure[J]. Reliability Engineering & System Safety，2001，71(3)：229-247.

[38] PAPADOPOULOS Y，WALKER M，PARKER D，et al. Engineering failure analysis and design optimisation with HiP-HOPS[J]. Engineering Failure Analysis，2011，18(2)：590-608.

[39] WANG F，SHEN G H，HUANG Z Q，et al. Method Combining Linear Temporal Logic and Fault Tree for Software Safety Verification[J]. Computer Science，2015，42(12)：71-75. (in Chinese)
王飞，沈国华，黄志球，等. 一种结合线性时序逻辑和故障树的软件安全验证方法[J]. 计算机科学，2015，42(12)：71-75.

[40] ZHOU L，HUANG Z Q，HUANG C L. Construction Method for Fault Tree Domain Ontology Supporting SWRL Rules[J]. Computer Science，2015，42(8)：198-202. (in Chinese)
周亮，黄志球，黄传林. 故障树领域本体及 SWRL 规则的构建方法研究[J]. 计算机科学，2015，42(8)：198-202.

[41] WEI O，SHI Y F，XU B F，et al. Abstract Modeling Formalisms in Software Model Checking[J]. Journal of Computer Research and Development，2015，52(7)：1580-1603. (in Chinese)
魏欧，石玉峰，徐丙凤，等. 软件模型检测中的抽象模型研究综述[J]. 计算机研究与发展，2015，52(7)：1580-1603.