软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, [doi: 10.13328/j.cnki.jos.000000]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于运行时验证的无人飞行系统安全威胁检测方法*

杨栋, 史浩, 董威, 刘宗林, 周戈

(国防科技大学计算机学院, 湖南长沙 410073)

通讯作者: 董威, E-mail: wdong@nudt.edu.cn

摘要:无人飞行系统(Unmanned Aerial Systems,UAS)的软、硬件存在缺陷以及遇到外部恶意攻击,会给 UAS 的安全性带来极大威胁.由于 UAS 的运行环境复杂多变,很多因素在开发过程中难以准确预测,因此研究有效的运行时安全保证机制具有重要意义.本文提出一种基于运行时验证的 UAS 安全威胁检测方法.首先对 UAS 可能遇到的多种安全威胁进行分析并采用离散时间时序逻辑进行描述,提出相应的 UAS-DL 语言描述安全监控规约;然后基于交错自动机提出了自动生成安全威胁监控器的算法,并利用参数化方法实现对多 UAS 的安全监控.为了提高检测的准确性,进一步研究了将运行时验证和贝叶斯网络推断结合的方法.采用实际的 UAS 开发仿真平台 Ardupilot 进行了实验,并设计了将监控器独立部署在 FPGA 硬件上的方法,避免对 UAS 计算资源的过多占用.实验结果表明上述方法能够有效检测 UAS 的安全威胁.

关键词:无人飞行系统;安全威胁检测;运行时验证

中图法分类号: TP311

中文引用格式: 杨栋,史浩,董威,刘宗林,周戈.基于运行时验证的无人飞行系统安全威胁检测方法.软件学报.http://www.jos.org.cn/1000-9825/ 0000.htm

英文引用格式: Yang Dong, Shi Hao, Dong Wei, Liu Zonglin, Zhou Ge. Security and Safety Threats Detection for Unmanned Aerial System Based on Runtime Verification

Ruan Jian XueBao/Journal of Software, 2016 (in Chinese).http://www.jos.org.cn/1000-9825/0000.htm

Security and Safety Threats Detection for Unmanned Aerial System Based on Runtime Verification

YANG Dong,SHI Hao,DONG Wei,LIU Zonglin,ZHOU Ge

(College of Computer, National University of Defense Technology, Changsha 410008, China)

Abstract: The defects of the software and hardware in unmanned aerial system and external malicious attack will pose a great threat to the security and safety of UAS. Due to the complex running environment of UAS, many factors are difficult to predict accurately in the development process, therefore, it is of great significance to adopt an effective runtime security and safety guarantee mechanism. This paper proposes a UAS security and safety threat detection method based on runtime verification. Firstly, after analyzing a variety of security and safety threats that UAS may encounter, we describe them in Discrete-Time MTL and present the corresponding UAS-DL language to describe the security and safety monitoring specification. Then we propose an automatic generation algorithm of security and

* 基金项目: 国家重点研发计划(2017YFB1001802),国家 973 项目(2014CB340703),国家自然科学基金(61690203,61532007).
Foundation item: National Key Research and Development Program of China (2017YFB1001802) , National 973 Program of China (2014CB340703), National Natural Science Foundation of China (61690203, 61532007).
收稿时间:0000-00-00; 修改时间: 0000-00-00; 采用时间:0000-00-00; jos 在线出版时间: 0000-00-00

CNKI 在线出版时间: 0000-00-00

safety threats monitor based on the alternating automaton and implement security and safety monitoring of multi-UAS by parameterization method. In order to improve the accuracy of the detection, we further study the method of combining runtime verification with Bayesian network inference. The experiment was carried out with the actual UAS development simulation platform Ardupilot, and monitor are deployed on the Field-Programmable Gate Array(FPGA) hardware independently to avoid excessive usage of UAS computing resources. Experimental results show that the proposed method can effectively detect the security and safety threats of UAS.

Key words: Unmanned aerial systems, security and safety threat detection, runtime verification

无人飞行系统(Unmanned Aerial Systems,UAS)包括无人机、无人浮空器、跨介质无人飞行器等各类飞行平台以及相关任务设备、信息传输设备及地面配套设备,可以自主感知、自主决策、相互协同执行任务,具有“平台无人、系统有人”的属性和自主程度高、非接触、零伤亡、可长时间工作的特点,在军事和民用领域中具有广阔的应用发展前景.软件在无人飞行系统中发挥着核心控制作用,软件中存在的缺陷或软件系统的设计考虑不够全面、健壮,将会给无人系统的保密安全性(Security)和可靠安全性(Safety)带来极大威胁,例如被外部通过各种攻击方式进行诱导、劫持,或者因软硬件缺陷引发异常而导致任务失败甚至坠毁.

尽管无人飞行系统的相关技术不断发展,但其安全威胁检测和防护技术还远不能达到人们的要求.现有无人机系统必须依靠数据通信来完成任务,通信信号易受到外界电磁环境的影响,在最近的几场局部战争中时常出现无人机被干扰坠毁或是诱骗的案例.例如 2011 年,伊朗军方就用 GPS 诱骗的方法成功捕获了美军隐形无人侦察机 RQ-170.不光如此,由无人机驾驶员本身误操作导致无人机坠毁的事件也时有发生.人们一方面加强无人飞行器控制系统中软硬件设计方法研究,另一方面也认识到在系统运行时提供有效的机制从而能及时发现可能的安全威胁非常重要,必须要研究无人飞行系统的运行时安全保证方法.例如美国航空航天局(NASA)已开始针对 UAS 的安全保证,采用运行时验证(Runtime Verification,RV)技术对系统运行过程中出现的安全问题进行检测,并在美军龙眼(Drangon Eye)无人机上进行了试验应用[1].

运行时验证是一种轻量级的软件安全性保障技术,它基于目标系统的运行轨迹来判定给定的规约性质是否得到满足.由于系统运行轨迹的唯一性,它有效克服了当系统过于复杂后,对设计模型或代码进行模型检验所产生的状态空间爆炸问题,和软件测试技术相比也不需要考虑所有路径的覆盖率问题,因此受到广泛研究和关注.与传统的软件可靠性保障技术不同的是,运行时验证技术主要应用在系统部署后,对运行环境和上下文不确定导致的系统问题能很好地监控.

运行时验证技术一般采用时序逻辑(例如线性时序逻辑 LTL)对要监控的性质进行描述.对于无人飞行系统的安全威胁进行检测时,需要监控的事件和状态不仅需要考虑其时序特征,还要考虑它们的实时特征以及参数化特征,这为监控器的生成和运行效率带来了挑战.本文对无人飞行系统的安全性威胁类型进行了分析,研究了针对几种主要类型的安全性质进行规约以及生成相应监控器的方法,并研究了监控器的高效部署和运行机制,通过仿真实验表明了方法的有效性.本文的主要贡献包括:

1) 提出基于离散时间的度量时序逻辑(DT-MTL)对无人飞行系统的安全性质进行规约的方法,并设计了一种对性质和相关监控事件、状态进行描述的规约语言 UAS-DL;
2) 提出了一种基于交错自动机从监控规约自动化生成监控器的算法,以及基于路径切片以支持参数化性质监控的方法;
3) 研究了将运行时验证和基于贝叶斯网络的故障诊断相结合的方法,以提高安全威胁检测的精确性;
4) 为减少资源占用和提高监控效率,采用 FPGA 硬件实现监控器,并在广泛应用的无人机开发和仿真平台 Ardupilot 中进行了实验和分析.

本文后面第 1 节介绍无人飞行系统面临的主要安全威胁以及论文研究的总体框架,第 2 节介绍了运行时验证相关理论以及如何对无人飞行系统安全性质进行描述,第 3 节定义了能够描述性质离散实时特征的 DT-MTL 以及监控规约,第 4 节给出了基于交错自动机生成运行时监控器的方法,第 5 节研究了运行时验证和贝叶斯网络结合提高监控准确性的方法,第 6 节对提出的方法进行实验和有效性评估,最后一节对相关研究进行

介绍并对工作进行了总结和展望.

1 基于运行时验证的安全威胁检测框架

为了在运行时对安全威胁进行检测,本节首先分析无人飞行器面临的安全威胁类型,然后提出基于运行时验证对这些安全威胁进行检测的技术框架,并对相关工作进行了介绍.

1.1 无人飞行系统的安全隐患类型

随着 UAS 的广泛使用,对于无人机的攻击手段也发生了日新月异的变化.下面我们简要介绍无人机所面临的一些主要安全隐患[2].由于无人飞行器自身的设计缺陷和外部的恶意攻击都可以带来安全隐患,其所面临的安全隐患可分为 UAS 行为隐患和外部攻击造成的隐患.

(1) UAS 行为隐患

主要指 UAS 自身软硬件缺陷和操作人员失误所带来的安全隐患.

- 飞行器振荡:无人机中的震荡可能由异常的命令序列或是错误改变控制环路中增益值引发,可能会导致机体损坏甚至飞行器解体并坠毁.
- 偏离飞行路线:一般无人机沿直线从一个点飞行到下一个点,突然偏离航线可能意味着一些意外的飞行机动,这也包括飞行器的突然爬升和下降.
- 传感器的读数:传感器读数的突然变化和朝一个方向的连续漂移可能是受攻击后引发的结果,也可能是由传感器本身失灵导致的.
- 软件行为异常:比如内存泄漏、实时错误、非法的数值等,这些可能是软件本身存在缺陷,也有可能是软件漏洞被外部恶意攻击所引发的.

(2) 外部攻击隐患

主要指外部实体利用破解后的通信信道或伪造的各类信号对 UAS 安全造成的隐患.

- 非法命令:不应该被无人机系统软件执行的命令,这样的命令可能是有指令传输错误或者外部攻击引起的.如果反复接受到这种命令就很有可能是由外部攻击引起的.
- 危险命令:格式上正确的但可能导致严重的问题甚至坠毁的命令.比如,不在飞行状态下重启飞行系统并不会导致严重问题,但在飞行状态下向无人机发送重启飞行系统的命令很有可能导致飞机坠毁,因为这样会导致所有的通信和系统参数都会丢失.
- 无意义和重复的导航命令:尽管允许在无人机飞行的时候向它发送新的导航路径来更新其任务,但是反复发送完全相同或是异常的坐标则可能是一种攻击.
- GPS 信号的突然改变:因为无人机系统导航的质量非常依赖于接收到的 GPS 信号,导航系统搜星数目、GPS 信号强度和噪声比的突然变化都可能是利用 GPS 信号进行攻击.

1.2 技术研究框架

由于软件对于 UAS 的控制至关重要,因此设计出更加安全的软件系统、使软件能够应对各种安全威胁受到人们重视,并且通过采用各种测试、静态分析、形式验证技术来检测部署前软件本身存在的隐患.然而,飞行中的无人机面临的很多安全威胁都是来自外部环境,这在开发阶段难以预测和应对,因此需要一种运行时机制能够实时监测 UAS 的运行状态和环境,及时发现存在的安全威胁,以便采取相应的应对措施.此外,由于很多带来安全威胁的事件都带有时序、实时的特征,并且和物理部件、通信等信息密切相关,事件和状态之间还有先后因果关系,因此仅靠传统的状态监控难以对这些安全威胁进行检测.为此,我们采用运行时验证的方法,以自动化生成运行时监控器的方式对UAS安全威胁进行检测.图1.1给出我们所提出的基于运行时验证的安全威胁检测方法框架.

无人飞行器飞行过程中需要不断接收地面控制台的指令并通过卫星实时获得自身位置数据,导航电脑会记录无人机飞行情况,如CPU温度、内存使用率、传感器的参数信息等,导航通信数据和导航电脑数据通过 UAS

软件系统实时生成日志,以便分析无人机的工作情况.传感器参数信息可以通过读取 UAS 日志获取,也可以通过直接读取传感器数据获取.为了保证传感器信息的实时性,我们在此选择直接读取传感器信息.

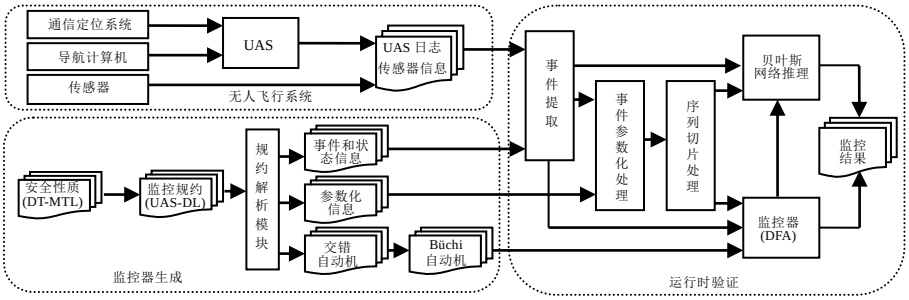

Fig.1.1 Framework of security and safety monitoring for UAS

图 1.1 无人飞行系统安全威胁检测方法框架

在图 1.1 方法框架中,首先定义了对安全威胁相关性质进行描述的时序逻辑 DT-MTL,能够体现威胁检测所需的时序、实时、参数化等特征;为了能够自动生成可运行的软件监控器,进一步设计了能够将时序逻辑公式与系统事件、状态进行映射的监控规约语言 UAS-DL;为了根据监控规约生成相应的安全威胁监控器,基于交错自动机将 DT-MTL 性质转换为自动机形式的监控器,然后生成相应的监控代码.UAS 日志和传感器信息作为运行时验证模块的输入首先传输到事件提取模块进行数据的提取和数据格式的转化, 然后将轨迹送给监控器进行判断并产生监控结论.为了对参数化性质进行监控,采用基于路径切片的方法,结合规约解析生成的事件、状态信息和参数化信息, 将运行轨迹分解并对应到相应的监控器.有些安全威胁可以从时序性质的违背直接推断出来,但有些时序性质只是引发安全威胁的可能原因之一,为此将运行时验证与基于贝叶斯网络的推断方法结合,监控器的判断结果可以输入到贝叶斯网络,根据事件提取和序列切片产生的先验信息进行推理以产生更为精确的监控结果.

1.3 相关研究

运行时验证中应用最为广泛的时序逻辑是线性时序逻辑(Linear Temporal Logic, LTL),并且由于复杂性上的原因[3][4],不带 past 算子的 LTL 子集更为常用(尽管有些时候带 past 的 LTL 直觉上更加有优势).关于各种离散系统和连续系统上的运行时验证的调查、对比、研究有很多[5],学界也提出了很多算法和框架[6][4],但它们在硬件上的具体应用,多不涉及实时性质[7].NASA 曾开展研究使用 DT-MTL 描述离散控制系统中的实时性质,但相关监控器生成算法并不是基于自动机框架 [8].

另一方面,在工具支持上,运行时验证对于软件的插装多基于面向方面的编程(Aspect Oriented Programing, AOP)[9].在 Java 中,有比较成熟的 AOP 机制 AspectJ[10],所以可以较为方便的进行监控器的插装,例如 JavaMOP 使用的就是这种插装方式.而对于在硬件系统中应用更为广泛的 C 语言,AOP 并不能起到很有效的作用.例如,AspectC++是用于 C++的 AOP 的实现,但其生成的代码不能由 C 编译器编译[11].Coady 等人使用 AspectC 对 FreeBSD 操作系统内核中页面错误处理预取的实现进行模块化,取得了显著效果.但是他们只使用了用于 AspectC 的理论设计,仅支持函数调用和控制流的连接点 [12].ACC(AspeCt-oriented C)是当前最先进的 C 语言 AOP 实现,但它目前的维护工作并不完善,最新版本存在大量错误,例如,连接点和切入点有时不能正确匹配,插装代码在语义上可能与其对应的方面不一致等.另外,ACC 的实现没有很好地符合模块化原则,因此对其进行扩充和改进也很困难[13].

近年来,运行时验证作为一种轻量级的软件系统安全性保障技术受到了广泛研究和关注.以往多数研究关注对计算系统的运行时监控,但由于越来越多的安全攸关系统是软硬件及运行时环境的融合,运行过程中的物理环境难以事先准确预测,因此对这类将计算过程和物理环境、人机交互结合起来的系统进行运行时验证越来越受到重视.例如,Stefan Mitsch[14]等人将定理证明合成监控器的方法用于信息物理融合系统(Cyber-Physical System,CPS)的运行时验证;Pierre Fraigniaud[15]等人利用去中心化思想将 RV 运用于分布式系统的监控;André

Matos Pedro[16]等人利用 MTL 语言实现对实时系统的监控.

随着无人系统更加广泛地用于民生和国防各领域,采用运行时验证技术对无人系统进行监控也出现相关研究.例如,Aaron Kane1 等[17]利用独立硬件实现对自主驾驶系统的运行时验证,Anthony M. Aiello[18]等人利用运行时保证技术实现对关键飞行控制系统的监控,但在他们的监控框架中都只关注了可靠安全性,无法对保密安全性进行监控.Johann Schumann 等[1]提出的 UAS 监控框架 R2U2 实现了对无人机常见安全性质的监控,但其缺乏参数化性质的监控方法,无法用于对无人机机群的监控.

2 运行时验证及关注的 UAS 安全性质

2.1 运行时验证的定义

运行时验证是一种轻量级的验证技术.它是传统验证技术,比如模型检测和测试的一个有效补充.它最重要的一个特征是其验证对象为被监控系统的实际运行轨迹,避免了模型检测的状态空间爆炸问题和测试的不完全.文献[19]中给出了一个在运行时验证领域广泛接受的一个定义:

定义 2.1 (软件运行时验证):软件运行时验证是计算机科学中验证被监控程序的一次运行是否满足或违背给定性质的相关技术.

从定义可以看到,运行时验证关注于系统的运行轨迹是否违背监控性质,当性质被违背时,运行时验证技术一般不会对被监控的系统进行调整.但是在实际应用中,运行时验证工具还往往对被监控系统的行为进行调整.现行的运行时验证框架,比如 Tracematches[20],JavaMOP[21],Larva[22]等,均可以在程序运行满足或违背监控性质时(JavaMOP 甚至允许在每个监控事件发生时),执行用户提供的任何代码,这种行为被称为“Runtime Reflection”[19].

在运行时验证中,通常使用一个监控器来检查程序的当前运行是否满足给定性质,监控器一般从监控性质自动产生.文献[19]对监控器定义如下:

定义 2.2 (监控器):监控器是一个输入有穷轨迹并输出某种裁决结论的装置.

现实中,监控器不一定是独立存在的部件,有可能是与目标软件并发的进程,或者作为目标软件系统体系结构的有机组成部分,甚至是监控代码与目标代码交织在一起.根据监控器是否与被监控系统同时运行,可以分为“在线监控”(Online Monitoring)和“离线监控”(Offline Monitoring).离线监控一般用于事后分析,在线监控能尽快发现可能存在的问题,但需要占用系统更多的资源.本文为了及时发现安全威胁,采用在线监控的方式.

2.2 UAS安全性质

根据被监控软件的应用领域以及监控性质的不同,在运行时验证中存在多种时序逻辑用来描述监控性质.最常用的是线性时序逻辑 LTL.但由于很多性质关注实时、参数化等特征,因此又采用很多能力更强的逻辑对性质进行描述.下面我们简要介绍 LTL 以及如何采用 LTL 的扩展 MTL 来描述 UAS 安全性质.

(1) 线性时序逻辑(Linear Temporal Logic, LTL)[23] [24]

LTL 是由 Pnueli 提出,被广泛应用于模型检验和运行时验证领域,用于描述安全性质.LTL 公式主要有以下几个部分构成.

- 系统状态变量:本文中主要包含从 UAS 日志和各个传感器中获取的系统状态数据.
- 命题逻辑操作符:包含标准操作符逻辑与(∧),逻辑或(∨),非(¬),和蕴含(→).
- 时序操作符:这些操作符表达时间之间的时序关系,比如 ALWAYS, EVENTUALLY,NEXTTIME, UNTIL.例如对于布尔变量 p,q,基本的时序逻辑公式含义如下:
  - GLOBAL p,(Gp):p 必须在时间轴一直为真.
  - FUTURE p,(Fp): p 必须在现在或未来的某个时间点为真.
  - NEXTTIME p,(Xp): p 必须在下一个时间点为真.
  - p UNTIL q,(pUq): q 在现在的时间点为真,或者 q 在未来某个时间点为真之前 p 连续为真.

(2)度量时序逻辑(Metric Temporal Logic, MTL)

MTL 是对 LTL 的扩充,它的每一个时序逻辑操作符都有一个代表操作符持续时间的下标,例如$G[\mathrm{i}:\mathrm{j}]\mathrm{p}$, $F[\mathrm{i}:\mathrm{j}]\mathrm{p}$, $\mathrm{p}U[\mathrm{i}:\mathrm{j}]\mathrm{q}$.在标准的 MTL 中,时间区间是定义在实数上的,MTL 的语义也是定义在连续时间轴上的.在本文中,我们通过对 UAS 系统的运行进行离散化,得到系统的执行序列,并将 MTL 的语义简化到整数域上,执行序列的每一个节点代表系统的一个周期,因此[2,6]就代表从现在起第 2 到 6 个系统周期内的时间区间.$G[2:6]\mathrm{p}$表示在第 2 到 6 个系统周期内 p 一直为真.LTL 公式可被认为是不带时间区间的 MTL 公式.下面作为示例,我们对无人机可靠安全性(Safety)和保密安全性(Security)一些典型性质进行描述.因为后续各节会用到其中部分性质作为案例,我们对其进行了编号 V1-V3 以及 R1-R4.

1)可靠安全性质

V1:无人机所能达到的最低的安全飞行高度 $\mathrm{H_{min}}$

$G(\mathrm{H_{min}} \geq 120\mathrm{m})$

V2:无人机正常飞行时机体震荡频率 $\mathrm{F_{osl}}$

$G(\mathrm{F_{osl}} \leq 10\mathrm{Hz})$

V3:无人机控制器在工作飞行过程中温度 $\mathrm{T_{mcu}}$ 必须保持在一定的范围内,并且工作频率 $\mathrm{F_{mcu}}$ 在 100 个时间周期内不能一直大于 800MHZ 以免处理器过热.

$G((-20℃ \leq \mathrm{T_{mcu}} \leq 100℃) \wedge \neg(G[0,99]\mathrm{F_{mcu}} > 800\mathrm{MHz}))$

2)保密安全性质

对于具有相似功能的事件,我们可以定义事件集合来增强表达式的表达力,例如,我们可以定义风险指令集、反常指令集、反常周期指令集如下:

risky_cmds= cmd_reset∨cmd_calibrate_sensor∨cmd_disarm∨ ...

abnormal_cmds = cmd_get_params∨set_params∨get_waypoints∨ ...

abnormal_cmds_periodic = cmd_nav_to∨cmd_mode_change∨nvalid_packet_rcvd∨...

定义事件集合具有很好的可复用性,同时还能使我们更直观的表达复杂的规约,例如:

R1:无人机在飞行过程中不能接受带有风险的指令

$G((\mathrm{CMD} == \text{takeoff}) \rightarrow ((\neg\textbf{risky\_cmds})\ U\ \text{landing\_on}))$

R2:无人机在飞行过程中接受到反常指令并在随后的 0 到 200 时间段内机体发生了大幅振荡

$G((\mathrm{MODE} == \text{takeoff}) \wedge (\textbf{abnormal\_cmd} \wedge F[0,200]\ \mathrm{OSL}))$

R3:无人机在收到飞行模式改为导航模式的指令后 100 个时间周期内收到反常周期指令

$G((\mathrm{CMD} == \mathrm{GUIDED}) \wedge F[0:100]\textbf{abnormal\_cmds\_periodic})$

R4:下面我们以更为复杂的拒绝服务攻击(Denial of Service,DOS)为例,进一步解释安全性质,后文也将会以 DOS 攻击为例,详细介绍安全威胁检测过程.拒绝服务攻击源于互联网上分布式拒绝服务攻击(**D**istributed **D**enial**o**f **S**ervice,DDOS),亦称洪水攻击.顾名思义,即是利用网络上已被攻陷的电脑作为"僵尸",向某一特定的目标电脑发动密集式的"拒绝服务"式攻击,用以把目标电脑的网络资源及系统资源耗尽,使之无法向真正正常请求的用户提供服务.

攻击者可以效仿 DDOS 攻击的方式来攻击无人机[1],首先发送大量试探性的指令来破解通信链路,在破解过程中无人机会在短时间内收到许多坏包;攻击者在破解无人机通信协议之后可能会发送一系列反常指令来获取无人机的飞行数据;攻击者还可能采取更改飞行方式、修改飞行坐标点的方式来劫持无人机到指定的地点,在此过程中可能会发送大量导航命令使地面控制台发送的信号淹没在命令噪声中.为此我们制定如表 2.1 所示规约.

值得一提的是,对于性质 R4 的判断,不是简单的将 R4.1-R4.4 通过合取或析取后进行判断,因为其他非外部攻击操作也可能引起上述情况.为此,我们采用贝叶斯网络进行推断,将上述性质各自的监控结果作为判据输入到贝叶斯网络中,进一步得到关于此类复杂性质的判断结果,具体计算过程在第 5 节有详细介绍.

Table 2.1 Properties description of DOS

表 2.1 拒绝服务攻击的安全性质描述

ID	性质描述	性质公式
R4.1	在 5 个单位周期内接受不超过一个坏包	$G\neg$(bad_packet $\wedge F[1,4]$bad_packet)
R4.2	规定无人机在高度超过 50 后不能接受反常指令	G((ALT >= 50) $\rightarrow$ (($\neg$**abnormal_cmd**) U landing_on))
R4.3	连续反常指令不能超过 10 个周期	$G\neg(G[0,9]$**abnormal_cmd**)
R4.4	连续导航指令不能超过 10 个周期	$G\neg(G[0,9](CMD == NAV))$

3 离散实时性质及监控规约语言

3.1 离散时间上的MTL

从上一节的性质可以看到,UAS 中需要监控的安全性质往往和时间有关.例如,在飞行过程中,由于跑道长度有限,要求无人机在起飞后二十秒内达到某个飞行速度.这条性质不仅牵涉到两个事件(发出起飞指令、达到飞行速度)的先后关系,还对两个事件之间的时间进行了量化约束.这种体现了系统对时间属性具有约束性需求的安全性质被称为实时性质.实时性质不仅限制事件之间的时序关系,更强调与时间相关的约束.因此,实时性质大多不能由 LTL 简洁、有效地表达.

由于实时性质的建模及监控过程与时间特性密切相关,它的运行时验证与传统的运行时验证过程也有所区别.要对实时性质进行运行时验证,首先必须使用一种实时时序逻辑来对其进行描述,然后,基于该实时时序逻辑公式构造相应的监控器来对目标系统进行监控.但从实时性质生成监控器以及运行监控器由于开销非常大而难以在实际中实现,我们研究的目标系统是具有时间特征的离散控制系统,其控制过程具有以时间周期为单位的特征,因此为了提高监控效率,我们引入了离散时间上的 MTL 来描述安全性质.

在离散控制系统中,与时间相关的计算并不会真正转化为实数数值计算,而是通过系统的采样周期抽象成离散的周期数,进而进行计算.如对于上述性质中“二十秒”的概念,具体到一个 CPU 时钟频率为 500Hz 的 FPGA 中,对应的周期数就是一万个时钟周期.因此我们可以借助嵌入式系统的周期性质,将实时性质离散化,用周期数目来代替时间约束.进一步,我们采用离散时间上的 MTL(Discrete-Time MTL,DT-MTL)来描述简化后的实时性质.总体看,与采用 LTL,TLTL 等时序逻辑相比,采用 DT-MTL 可以更好地描述与时间区间相关的无人飞行系统安全性质.NASA 曾在相关工作中采用 DT-MTL 对无人飞行系统的健康管理功能进行描述,表现出良好的效果.

DT-MTL 是将 MTL 通过离散化获得的子集,它的时间区间为离散化的数值,可以转化为整数运算,相比于 MTL,降低了运算复杂度.在定义 DT-MTL 的语法之前,我们先定义时间区间$\mathrm{J} = [\mathrm{t}, \mathrm{t}']$,其中$\mathrm{t}, \mathrm{t}' \in \mathbb{N}^0$.DT-MTL 的语法和 LTL 类似,唯一不同的地方在于 U 算子的定义[25].

定义 3.1(DT-MTL 语法):设AP为原子命题集合,则 DT-MTL 合式公式(也简称为 DT-MTL 公式)φ可以归纳定义如下:

$$\varphi := \mathrm{p} \in \mathrm{AP} \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \mathrm{X}\varphi \mid \varphi_1 \mathrm{U}_\mathrm{J} \varphi_2$$

为了使用方便,往往还会定义如下派生时序算子:

$$F_J\varphi \triangleq trueU_J\varphi \qquad G_J\varphi \triangleq \neg F_J\neg\varphi \qquad \varphi_1 R_J \varphi_2 \triangleq \neg(\neg\varphi_1 U_J \neg\varphi_2)$$

在此基础上,我们给出 DT-MTL 的语义定义.

定义 3.2(DT-MTL 语义):给定线性结构π, π是在 AP 上的原子命题集合构成的序列,其中$\pi(i)$为在π上第i个位置的命题集合,给定 DT-MTL 公式φ,可以归纳定义满足关系$\vDash$如下:若$\varphi = \mathrm{p} \in \mathrm{AP}$,则$\pi, i \vDash \varphi$当且仅当$\mathrm{p} \in \pi(i)$;

- 若$\varphi = \neg\psi$,则$\pi, i \vDash \varphi$当且仅当$\pi, i \nvDash \psi$;
- 若$\varphi = \varphi_1 \wedge \varphi_2$,则$\pi, i \vDash \varphi$当且仅当$\pi, i \vDash \varphi_1$且$\pi, i \vDash \varphi_2$;
- 若$\varphi = X\psi$,则$\pi, i \vDash \varphi$当且仅当$\pi, i+1 \vDash \psi$;

- 若$\varphi = \varphi_1 U_J \varphi_2$,则$\pi, i \vDash \varphi$当且仅当存在$k \geq i$并且$k - i \in$ J,使得$\pi, k \vDash \varphi_2$并且对于任意的$i \leq j < k$,都有$\pi, j \vDash \varphi_1$;

特别地,当i = 0时,直接将$\pi, i \vDash \varphi$记作$\pi \vDash \varphi$.

3.2 性质规约语言UAS-DL

时序逻辑公式所描述的性质比较抽象,在生成监控器并对系统进行监控时,要和相应的事件、状态进行映射,这需要把逻辑公式扩展成为相应的监控脚本,给出计算机能识别的监控规约表达形式.因此,需要一种脚本式的监控规约语言能够简洁、高效的表达安全性质.在 DT-MTL 的基础上,我们定义了面向 UAS 的性质规约语言 UAS-DL(UAS Dscrition Language)来对安全性质进行形式化表达.下面我们采用了 BNF 来描述表达监控性质的规约语言.

```
<Specification>     ::= <ID> "{"
                          { <Event> }
                          { <Property>
                            { "@violation"   "{" <Predefined Operation> "}" }
                          }
                        "}"
<Event>       ::= "event" <ID><Event Dscription>
<Property>  ::="DT-MTL:" <DT-MTL>
<Event Dscription>::= "MODE:" <MODE>|"CMD:" <CMD> | "PARM:" <PARM><OP><Integer>
<MODE>::="Auto"| "Guided"| "RTL"| ...
<CMD>::=<Normal Cmd> | <Risky Cmd> | <Abnormal Cmd> | <Invalid Cmd>| ...
<PARM>::="GPS" | "Baro" | "IMU" |"ALT" | ...
<OP>::=">=" | "=" | "<="
```

Fig3.1 Specification syntax of UAS-DL represented in BNF.

图 3.1 UAS-DL 规约语言的 BNF 描述

如图 3.1 所示, Specification 为一个规约,由一个特定的 ID、多个事件 Event 和多条性质 Property 组成.其中,事件 Event 作为性质中的原子命题,用于触发监控器的状态改变,可以是飞行模式 MODE 的状态,收到指令 CMD 的类型及参数信息 PARM 的值.Property 为 DT-MTL 逻辑公式,可以通过相应算法生成自动机形式的监控器.另外,规约中还定义了当监控器判断性质被违反时应当执行的行为 Predefined Operation,该行为可以是报警、记录或者对 UAS 的控制干预行为.其中 DT-MTL 是描述性质的时序逻辑公式,其语法的 BNF 描述如图 3.2 所示,包括了基于定义 3.1 中基本算子进行派生后的一些逻辑和时序算子.

```
<DT-MTL>                ::="true" | "false"
                        | <Event Name>
                        | "not" <DT-MTL>
                        | <DT-MTL> "and" <DT-MTL>
                        | <DT-MTL> "or" <DT-MTL>
                        | <DT-MTL> "=>" <DT-MTL>
                        | <DT-MTL> "<=>" <DT-MTL>
                        | <DT-MTL> "xor" <DT-MTL>
                        | "o" <DT-MTL>
                        | "[]" <Interval><DT-MTL>
                        | "<>" <Interval><DT-MTL>
                        |<DT-MTL> "U" <Interval><DT-MTL>
                        |<DT-MTL> "R" <Interval><DT-MTL>
<Event Name>      ::= </* <ID> in <Event> */>
<Interval>   ::= "[" <Integer> "," <Integer> "]"
```

Fig3.2 DT-MTL syntax represented in BNF

图 3.2 DT-MTL 的 BNF 描述

下面我们以 2.2 节中保密安全性质里的 R4 为例,给出 DOS 攻击的具体规约应用示例:

R4.1{
event C1 CMD: Bad_packet
DT-MTL: [] not (C1 and <>[1,4] C1)
@violation { Alarm }
}
	R4.2{
	event C2 PARM: ATL >= 50
	event C3 CMD: Abnormal
	event C4 CMD: landingon
	DT-MTL: [] (C2 =>(not C2 U C4))
	@violation { Alarm }
	}

R4.3{
event C5 CMD: Abnormal
DT-MTL: [] not [][0,9] C5
@violation { Alarm }
}
	R4.4{
	event C6 CMD: nav
	DT-MTL: [] not [][0,9] C6
	@violation { Alarm }
	}

Fig3.3 An example of DOS specification

图 3.3 DOS 攻击的规约示例

如图 3.3 所示,R4.1-R4.4 分别对应 2.2 节中的四条性质,其中 C1-C6 表示事件定义,每条规约包含一条 DT-MTL 公式,该性质限制了事件之间的时序关系;规约最后一部分表示违反性质后需要执行的操作,该例子中为输出警告信息.

4 安全威胁监控器的生成

为了能够在运行时对时序逻辑公式进行在线监控,一般把性质规约生成为自动机形式的监控器,在软件运行过程中监控器根据输入的状态和事件进行状态迁移,并在抵达非法状态时发出警告.对此,我们需要针对上一节中的监控规约给出监控器自动生成算法.对于离散时间的 DT-MTL,其表达能力实际与 LTL 是等价的,因此可以借鉴从 LTL 生成自动机的经典算法,给出基于 DT-MTL 的自动机生成算法.

监控器的生成过程如图 4.1 所示,该生成算法先将 DT-MTL 公式转换生成交错自动机(Alternating Automaton,AA),这一步算法是本节的重点,将在 4.1 小节中详述.从交错自动机生成 Büchi 自动机(Büchi Automaton,BA)步骤使用了[26]中描述的算法,该算法先将交错自动机转换得到广义 Büchi 自动机(General Büchi Automaton,GBA),再转换得到 Büchi 自动机,其中每一个环节的中间产物都需要先进行简化,再开始下一步转化,以减少复杂度.得到 Büchi 自动机后,将其通过经典算法转换得到非确定的有穷自动机(Non-deterministic Finite Automaton,NFA),再将其进行确定化得到确定有穷自动机(Deterministic Finite Automaton,DFA),这里不再赘述.

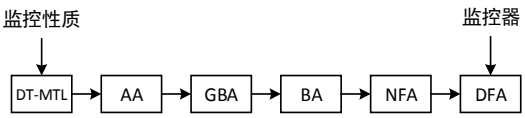

Fig.4.1 Monitor generation algorithm based on DT-MTL

图 4.1 基于 DT-MTL 的监控器生成算法

4.1 监控器生成算法

交错自动机由 Muller 和 Schupp 在[27]中进行了介绍,在[28]中 Rohde 定义了带偏序的交错自动机,这里我们用到的带偏序的交错自动机定义如下:

定义 4.1(偏序交错自动机):AP上的一个偏序交错有穷状态机是一个五元组$A =< Q, \Sigma, \delta, I, F >$,其中

- Q是有穷状态集合,记Q'是由Q中元素组成的所有合取式的集合;
- Σ是有穷非空字母表,记$\Sigma' = 2^{\Sigma}$;
- $\delta: Q \to 2^{\Sigma' \times Q'}$是迁移函数;

- $I \subseteq Q'$是初始状态集合;
- $F \subseteq Q$是接收状态集合.
- 状态集合Q中存在一个偏序,即对于$\forall q \in Q$,和所有出现在$\delta(q)$中的状态p,都有$p < q$.

如果一个运行在交错自动机中的路径存在分支仅有限次地经过接收状态,则称该运行是被接收的.

在给出从 DT-MTL 到交错自动机的转换算法之前,我们先定义$2^{\Sigma' \times Q'}$上的析取运算$\otimes$和 DT-MTL 公式φ上的集合化运算$\overline{\varphi}$:

定义 4.2:对于$K_1, K_2 \in 2^{\Sigma' \times Q'}$,

$$K_1 \otimes K_2 = \{(\alpha_1 \cap \alpha_2, e_1 \wedge e_2) | (\alpha_1, e_1) \in K_1 \ and \ (\alpha_2, e_2) \in K_2\};$$

对于一个 DT-MTL 公式φ,定义$\overline{\varphi}$:若φ是时序公式,则$\overline{\varphi} = \{\varphi\}$,否则,$\overline{\varphi_1 \wedge \varphi_2} = \{e_1 \wedge e_2 | e_1 \in \overline{\varphi}_1 且 e_2 \in \overline{\varphi}_2\}, \overline{\varphi_1 \vee \varphi_2} = \overline{\varphi}_1 \cup \overline{\varphi}_2$.

在此基础上,我们进一步给出从 DT-MTL 到交错自动机的转换算法:

定义 4.3:对于一个 DT-MTL 公式φ,可以定义相应的交错自动机$A_\varphi = < Q, \Sigma, \delta, I, F >$如下:

Q为φ的所有时序子公式集合,Σ为2^{AP},I为$\overline{\varphi}$,F为空,δ定义如下(Δ将δ扩展到所有子公式上):

$$\delta(tt) = \{(\Sigma, tt)\}$$

$$\delta(p) = \{(\Sigma_p, tt)\}, \quad where \ \Sigma_p = \{a \subseteq \Sigma | p \in a\}$$

$$\delta(\neg p) = \{(\Sigma_{\neg p}, tt)\}, \quad where \ \Sigma_{\neg p} = \Sigma \backslash \Sigma_p$$

$$\delta(X\psi) = \{(\Sigma, e) | e \in \bar{\psi}\}$$

$$\delta(\psi_1 U_J \psi_2) = \begin{cases} \Delta(\psi_1) \otimes \{\Sigma, \psi_1 U_{J'} \psi_2\}, \\ \qquad if \ N_1 > 0, where \ J' = [N_1 - 1, N_2 - 1] \\ \Delta(\psi_2) \cup (\Delta(\psi_1) \otimes \{\Sigma, \psi_1 U_{J''} \psi_2\}), \\ \qquad if \ N_1 = 0 \ and \ N_2 > 0, where \ J'' = [0, N_2 - 1] \\ \Delta(\psi_2), \quad if \ N_2 = 0 \end{cases}$$

$$\delta(\psi_1 R_J \psi_2) = \begin{cases} \Delta(\psi_1) \cup \{\Sigma, \psi_1 R_{J'} \psi_2\}, \\ \qquad if \ N_1 > 0, where \ J' = [N_1 - 1, N_2 - 1] \\ \Delta(\psi_2) \otimes (\Delta(\psi_1) \cup \{\Sigma, \psi_1 R_{J''} \psi_2\}), \\ \qquad if \ N_1 = 0 \ and \ N_2 > 0, where \ J'' = [0, N_2 - 1] \\ \Delta(\psi_2), \quad if \ N_2 = 0 \end{cases}$$

$$\Delta(\psi) = \delta(\psi) \ if \ \psi \ is \ a \ temporal \ formula$$

$$\Delta(\psi_1 \vee \psi_2) = \Delta(\psi_1) \cup \Delta(\psi_2)$$

$$\Delta(\psi_1 \wedge \psi_2) = \Delta(\psi_1) \otimes \Delta(\psi_2)$$

上述定义中tt表示永真式,结合上述定义可知$\delta(tt) = \{(\Sigma, tt)\}$表示:在状态 tt 下,接收任意字母都会使其跳转到自身.其余定义同理.通过上述转换算法,我们可以得到与 DT-MTL 公式φ语义相同的交错自动机A_φ.该转换算法的核心部分在于对迁移函数δ(及Δ)的定义.对于公式中的永真式、原子命题以及$\neg$、X、$\vee$、$\wedge$等算子,其转换算法均和 LTL 中相应算子的定义与计算相同,对于这些算子的语义一致性在论文[26]中均给出了证明.而对于U_J和R_J算子,可以将其先转化为 LTL 公式,并进一步得到以上的结论.通过对每一个算子的运算使用归纳法可以证明,该交错自动机A_φ和 DT-MTL 公式φ在语义是一致的.

例如,对于性质"无人机在发出提升指令后需要在 3 到 5 个周期之间高度提高 5 米,并且在高度达标前持续发出提升指令",可以定义事件p为"无人机发出提升指令",事件q为"无人机高度提高 5 米",则该性质可以用 DT-MTL 公式$pU[3,5]q$来表示,并且该公式可以通过上述转换算法经过转换得到如图 4.2 所示的交错自动机.类似地,DT-MTL 公式$pR[3,5]q$可以转换得到如图 4.3 所示的交错自动机.

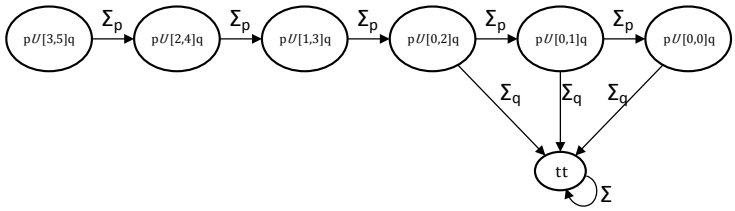

Fig.4.2 Alternating automaton translated form $pU[3,5]q$

图 4.2 由$pU[3,5]q$得到的交错自动机

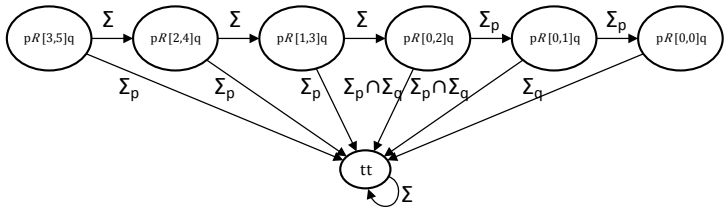

Fig.4.3 Alternating automaton translated form $pR[3,5]q$

图 4.3 由$pR[3,5]q$得到的交错自动机

得到交错自动机后,再按照如图 4.1 所示过程综合生成监控器.该监控器以自定义的中间表达形式表示,可以通过进一步的处理模块生成用于综合生成实际电路的 HDL 语言.

图 4.4 为图 3.3 示例规约中的 R4.1 综合生成的自动机,左边为监控器在代码中的表现形式.如规约中所示,C1 表示事件“接收到坏包”,相应的,nC1 表示事件“接收到正常数据包”,它们在代码中的表现形式是两个数组,数组中的第 N 位上的整数值表示在第 N 个状态遇到该事件时,应该跳转到的状态编号.右图为左边数组所表示的自动机,其中状态 0 为初始状态,状态 5 为违反状态,状态 6 位陷阱状态.通过分析该自动机可以看出,一旦无人机在五个时间单位内接收到超过一个坏包,即跳转到违反状态.

```
int Prop_1_transition_C1[] = {2,5,5,5,5,6,6};
int Prop_1_transition_nC1[] = {0,4,1,0,3,6,6};
```

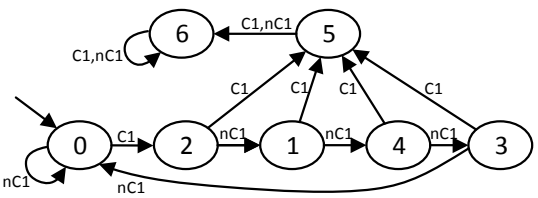

Fig.4.4 An example of monitor

图 4.4 监控器示例

4.2 参数化性质的监控

在离散控制软件中,普遍存在一种监控需求,要求一个类的不同对象或者一个结构体的不同实例均满足某个性质.例如在 UAS 中,往往要求多个无人机在飞行过程中飞行高度均不小于某个值.对于这类性质,当然可以对每个对象或实例均编写规约,以达到监控的目的,但更常用的做法是引入参数化的命题,或利用参数化的性质规约来对其进行描述,并使用形式化的方法生成监控器,以完成对这类性质的监控.

对于参数化性质监控,与不带参数的性质最大的不同即对参数的处理,尤其是对于多个对象进行集中式监控时需要进行特殊处理.运行时验证中典型的处理方法有两种思路.第一种是路径切片[29],该方法首先根据被监控性质生成一个监控器,再根据参数将程序的执行路径切分为多个切片,将多个路径切片分别作为输入送到监控器中,以对每个参数分别给出对应的结论;另一种方法是使用参数化的逻辑语言[30][31](如一阶线性时序逻辑)对安全性质进行描述,并综合生成带参数的自动机,并将含有参数的执行路径直接输入到自动机中,以得出性质

是否被违反的结论.

以上两种方法都可以在一定程度上解决对参数化性质进行运行时验证的问题,但它们的在参数化性质的表达能力上存在着微妙的区别[32].由于对参数化路径进行切片的思想更加直观,且易于实现,所以在目前已有的运行时验证工具中更倾向使用这种方法.我们在对参数化性质的监控框架设计上,也采用了这种方法.

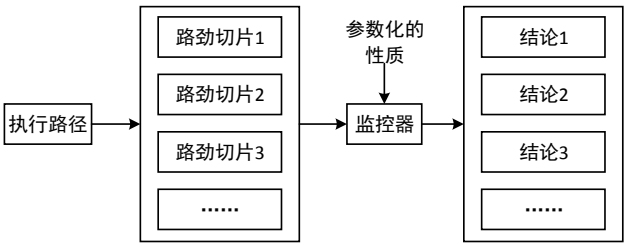

Fig.4.5 Framework of monioring parametric properties

图 4.5 对参数化性质进行监控的框架

在我们的运行时验证框架中,对参数化性质进行监控的思想如图 4.5 所示.在具体实施中,如图 1.1 中参数化部分所示,UAS 日志及传感器信息通过事件提取模块生成事件后,与参数化信息一同输入事件参数化处理模块.该模块对事件进行参数化处理,甄别每个事件所属的参数(不同飞行器或者飞行器上的部件),以确定该事件可以触发的状态变量.然后将事件序列依据不同参数进行切片化处理,将事件序列中与同一参数相关的事件进行标记提取,组成与该参数对应的路径切片,以此得到多条参数化的路径切片.最后,将各个切片输入到监控器后,监控器对不同参数的状态变量进行修改,以达到使用相同监控器、不同状态变量来处理不同路径切片的目的.

5 运行时验证和贝叶斯网络的交互

由于在 UAS 中,某些安全性质的违背可能是多种原因引发的, 单纯依靠监控器难以精确判断真正原因是否是由于外部攻击.为此,我们引入贝叶斯网络模型,来提高整体安全威胁检测的准确性.

贝叶斯网络又称为信念网络[33],其定义为〈X,A, Θ〉.其中: 〈X,A〉表示一个有向无环图(Directed Acyclic Graph, DAG) 的结构 G,如图 5.1 所示; X 是网络中节点的集合,$X_i \in$ X 表示一个限制定义域的随机变量; A 是网络中有向边的集合,$\alpha_{ij} \in$ A表示节点之间的直接依赖关系,即X_i与X_j之间的有向连接$X_i \leftarrow X_j$; Θ是网络参数,如图 5.1 中节点右侧的概率取值,$\theta_i \in \Theta$表示与节点X_i相关的条件概率分布函数,如图 5.1 左侧的列表.贝叶斯网络蕴涵了条件独立性假设,即给定一个节点的父节点集,该节点独立于它的所有非后代节点.因此,贝叶斯网络所表示的所有节点的联合概率就可以表示为各节点条件概率的乘积,如式(1)所示.

$$P(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} P(X_i | X_1, X_2, \ldots X_{i-1}) = \prod_{i=1}^{n} P(X_i | \pi(X_i)) \qquad (1)$$

其中: i = 1,2,⋯,n,$\pi(X_i)$表示X_i的父节点集.当给出了网络结构 G,节点间的相关关系也就随之确定.在这个前提下,结合网络参数Θ,一个贝叶斯网络就可以唯一地确定节点 X 的联合概率分布,得到推理结果.由于节点间存在条件独立的性质,贝叶斯网络的计算效率比其他计算联合概率的方法高很多.

首先,由于观测数据可能是完备也可能不是完备的,因此我们利用观测数据集合结合专家经验(先验知识)等相关方法建立监控对象的贝叶斯网络模型,然后将监控器的输出和其他相关信息作为贝叶斯网络的已知信息输入网络中,就可以得到引发问题的原因的精确概率值.通过观测数据的累积,我们还可以利用贝叶斯网络学习算法来更新贝叶斯网络,以得到更精确的网络模型.贝叶斯网络的学习包含了两大部分:结构学习和参数学习.观测数据又分为完备数据和不完备数据两种情况,它们所对应的学习算法有所不同.这里我们不详细描述贝叶斯网络的构造方法,具体内容可参考文献[34][35].

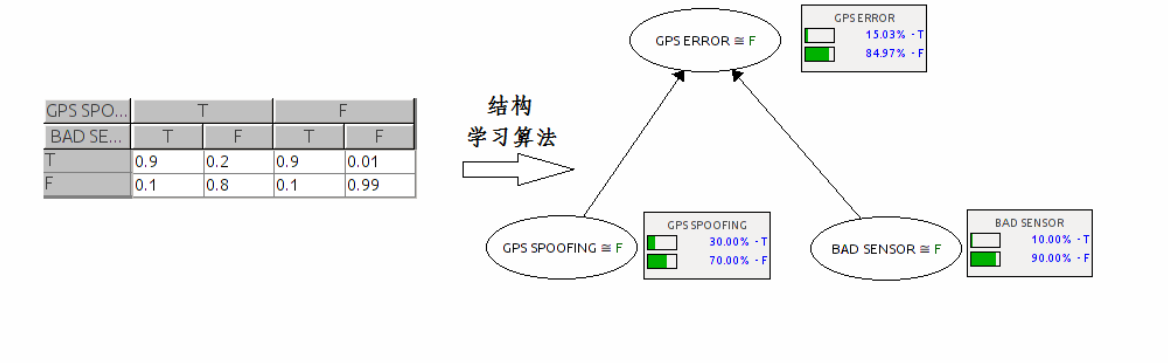

Fig.5.1 An example of Bayesian network

图 5.1 贝叶斯网络示例

如图 5.1,高度测量值的突然变化(GPS ERROR)可能是由于气压高度计、激光高度计和 GPS 接收器的故障(BAD SENSOR)引发的,也有可能是遭受到了 GPS 诱骗攻击(GPS SPOOFING) ,通过结构学习算法我们将联合概率表(左)转化为贝叶斯网络(右)的形式以便观察概率值的变化.为了确定最有可能的原因,我们可以利用最近接受的命令、GPS 接收器的信号强度来帮助我们判断.因此,如果是 GPS 信号强度突然变化而其他传感器比如气压高度计、激光高度计都正常时,可能确实遭受到了攻击.反之,如果机载高度传感器数值差异巨大时就有可能是传感器本身的故障引发的.随着像激光高度计比气压高度计或 GPS 接收器更容易失灵、GPS 信号在信号弱的地方很容易发生突变等先验知识加入到贝叶斯网络,就可以得到更精确的判断结果.本文中贝叶斯网络及其推理基于开源软件 Samiam 3.0 所提供的库进行设计实现[36].

6 实验

为了验证上述方法的有效性,我们首先在目前非常流行的开源无人机开发和仿真平台 ArduPilot[37]上进行软件模拟仿真.由于在实际部署的无人机控制系统中,如果监控器作为软件的组成部分会影响飞控软件的完整性和系统的运行效率,我们进一步研究了如何通过硬件 FPGA 来实现自动生成的监控器.

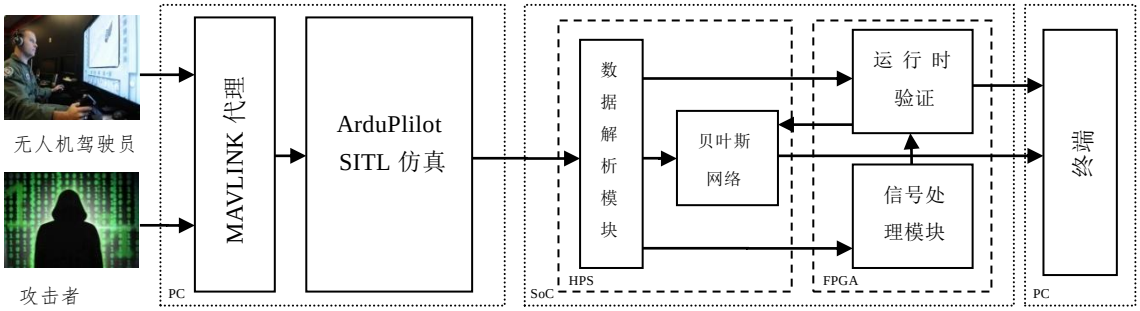

Fig.6.1 Platform of experiments

图 6.1 实验平台框架

ArduPilot(也称为 ArduPilotMega-APM)是一个开源的无人飞行器开发仿真平台,被广泛应用在多旋翼机、固定翼飞机,直升机,地面探测器等多种无人载具上.它提供了一套完善的开发、仿真、实验和应用平台,并能进行软件在环(System in the loop, SITL)和硬件在环 (Hardware in the loop, HITL)仿真,大大加快了开发进度.为了

实现硬件安全威胁检测监控器,我们采用了 Altera 的 CycloneV SoC(System On Chip)开发板.它在 FPGA 架构中集成很多系统级硬核功能,包括双核 ARMCortex-A9 硬核处理器系统(Hard Processor System, HPS)、嵌入式外设、多端口存储器控制器、串行收发器等.CycloneV SoC 被广泛应用在工业电机控制驱动器、协议桥接、视频转换器和采集卡,以及手持式设备上.在 UAS 运行时验证平台中,HPS 运行 Ubuntu Linux 12.04 操作系统,实现端口配置、信号预处理、评估运算电路和图形界面等功能,FPGA 实现信号处理、监控器等功能,HPS 和 FPGA 通过 AXI bridge 进行通信.

ArduPlilot 仿真环境和监控器的 FPGA 硬件实现框架如图 6.1,我们实验运行在 ArduPlilot 软件在环模拟(SITL)环境中,通过无人机通信协议 Mavlink 代理来模拟无人机驾驶员和攻击者的操作,产生的飞行数据通过 NFS 协议传送到 Altera 的 CycloneV SoC 开发板中,SOC 开发板分为 HPS 和 FPGA 两个部分,在 HPS 中对模拟产生的数据进行预处理,处理过的数据作为运行时验证模块和贝叶斯网络的输入,对于有些复杂的数字信号需要对其进行快速傅里叶变换(Fast Fourier Transformation, FFT)、扩展卡尔曼滤波(Extented Kalman Filter, EKF)后才能输入到运行验证模块[38].运行时验证模块中的逻辑电路根据输入产生相应的输出,对于由可能多个原因诱发的结果,可以将输出的结果输入到贝叶斯网络进行分析,以得到更加精确的结果.

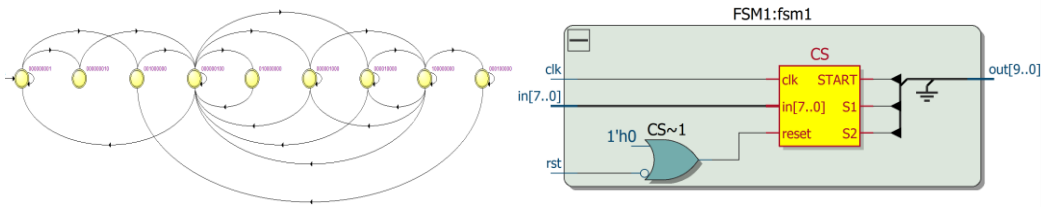

Fig.6.2 FSM view of FSM1(left) and RTL view of FSM1(right)

图 6.2 FSM1 的内部转换图(左)和 RTL 图(右)

图 6.2 显示了利用 Altera 的 Quratus II 将左图监控器自动机 FSM1 编译综合为右图的硬件 RTL 结果,实验中 FPGA 使用了 12815/32070(40%)的 ALMs、24901 个寄存器、389/457(81%)的引脚、1,975,936 / 4,065,280 (49 %)内存和 35 / 87 (40 %)的 DSP 单元,最大工作频率为 50MHz.HPS 模块和 FPGA 采用 AXI 总线进行通信,HPS 模块与模拟环境 ArduPilot 通过 NFS 服务器进行通信.对监控器代码综合成为 FPGA 代码主要是硬件实现过程,本文中不再具体进行阐述.

6.1 反常指令实验

无人机在飞行过程中由于无人机飞行员误操作导致的无人机坠毁事件时有发生,而且攻击者也可能利用无人机的漏洞远程发出反常指令来进行攻击,所以本实验通过判断无人机飞行过程中收到反常指令后是否产生大幅震荡来进行攻击检测,具体规约见 2.2 节的性质 R2:

$$G((\text{MODE} == \text{takeoff}) \wedge (\textbf{abnormal\_cmd} \wedge F_{[0,200]}\,\text{OSL}))$$

图 6.4 分别展示了正常飞行(上)和受到攻击后无人机(下)俯角的变化情况,由于振荡可以通过无人机俯角的周期性变化表现出来,因此,我们可以通过观察无人机俯角的变化来检测是否发生了振荡.对于振荡信号的处理,我们利用 Altera 提供的 FFT IP 模块进行频率分析,FFT 模块采样点设为 1024 个.俯仰角(Pitch)信号的采样周期为 40ms(频率 25Hz),由图 6.3 攻击后的振荡频域图(右下)可以看到在第 32 个周期(箭头处)较正常飞行的振荡频域图(右上)有明显频率分布,根据频率计算公式 $f=mf_s/N$(m: 目标频率的时钟周期,f_s:采样频率,N:采样点个数)可知该频率为 0.78125(32/1024*25)Hz, 与实际振荡频率基本相同.在图 6.4(下)中箭头处可以看到监控器检测到了攻击.

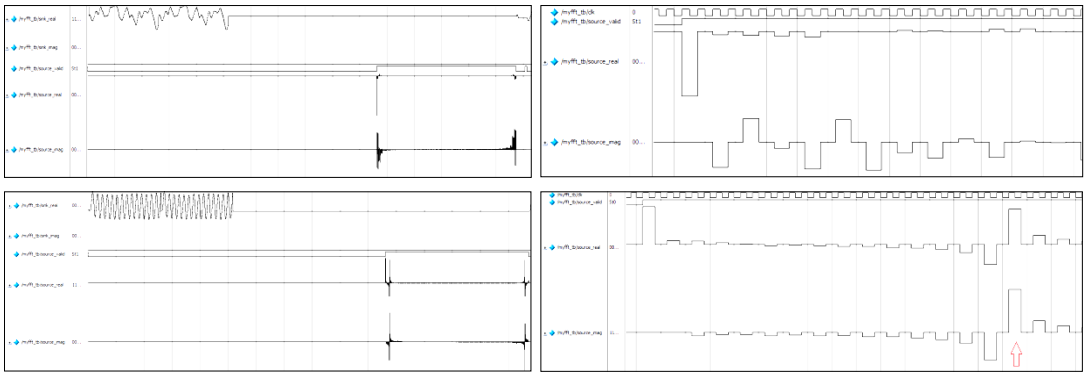

Fig.6.3 Simulation of pitch variations of normal flight(upper left) and attacked flight(lower left) in Fig.6.4 and oscillation spectrum of normal flight(upper right) and attacked flight(lower right)from the FFT module.

图 6.3 FFT 模块对图 6.4 的正常飞行(左上)和受攻击后(左下)俯仰角变化仿真图和正常飞行(右上)和受攻击后(右下)无人机振荡频域图

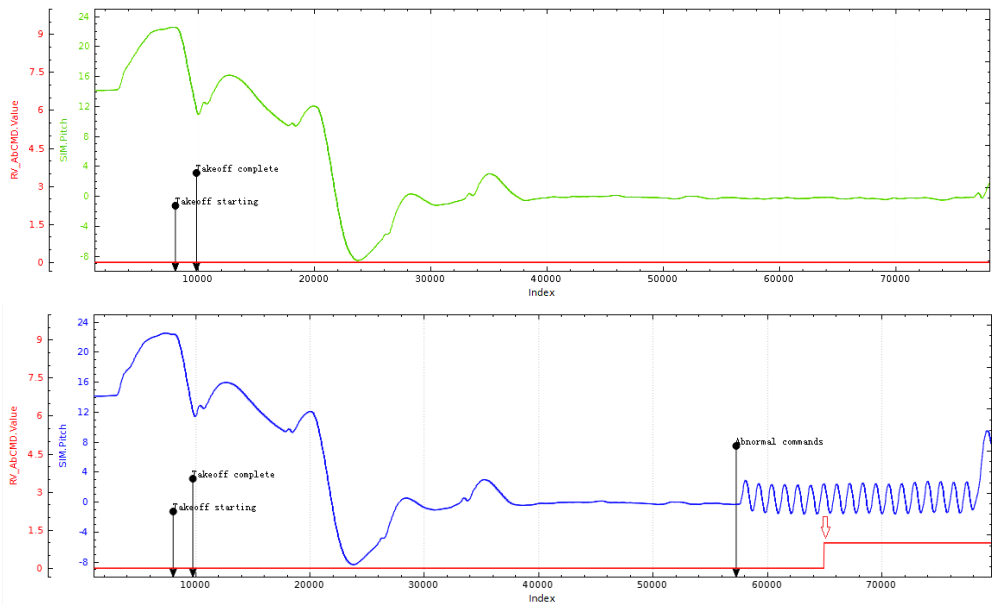

Fig.6.4 Pitch variations of normal flight(up) and attack flight(down)

图 6.4 正常飞行(上)和受攻击后(下)无人机俯仰角变换对比图

6.2 DOS劫持实验

2.2 节 R4 对 DOS 劫持的基本原理[39]和规约进行了详细介绍,然而单凭这些规约并不能断定无人机一定受到了 DOS 攻击,因为 GCS 的操作也可能导致类似的现象发生,因此为了更精确的计算出 DOS 攻击的可能性,我们利用先验知识对 DOS 攻击建立贝叶斯网络模型(如图 6.5)来确定引发无人机偏离指定航向的可能性最大的原因.

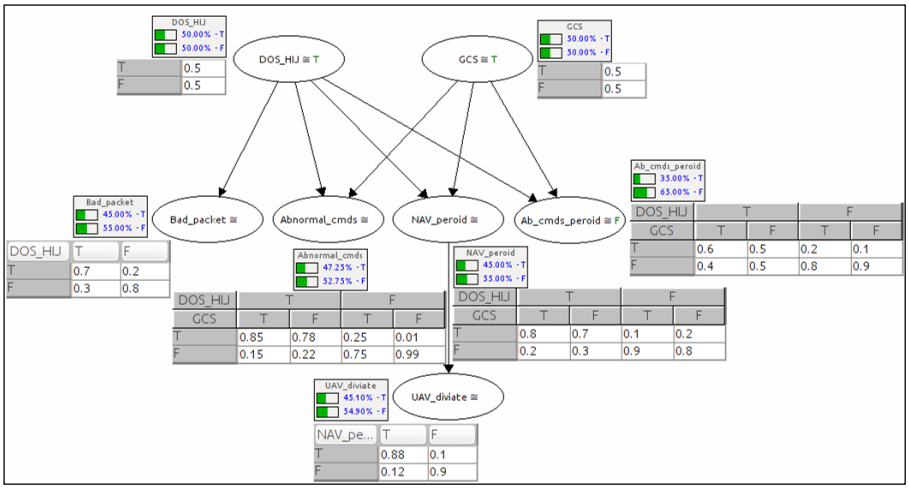

Fig.6.5 BN of the DOS attack

图 6.5 DOS 攻击诊断贝叶斯网络

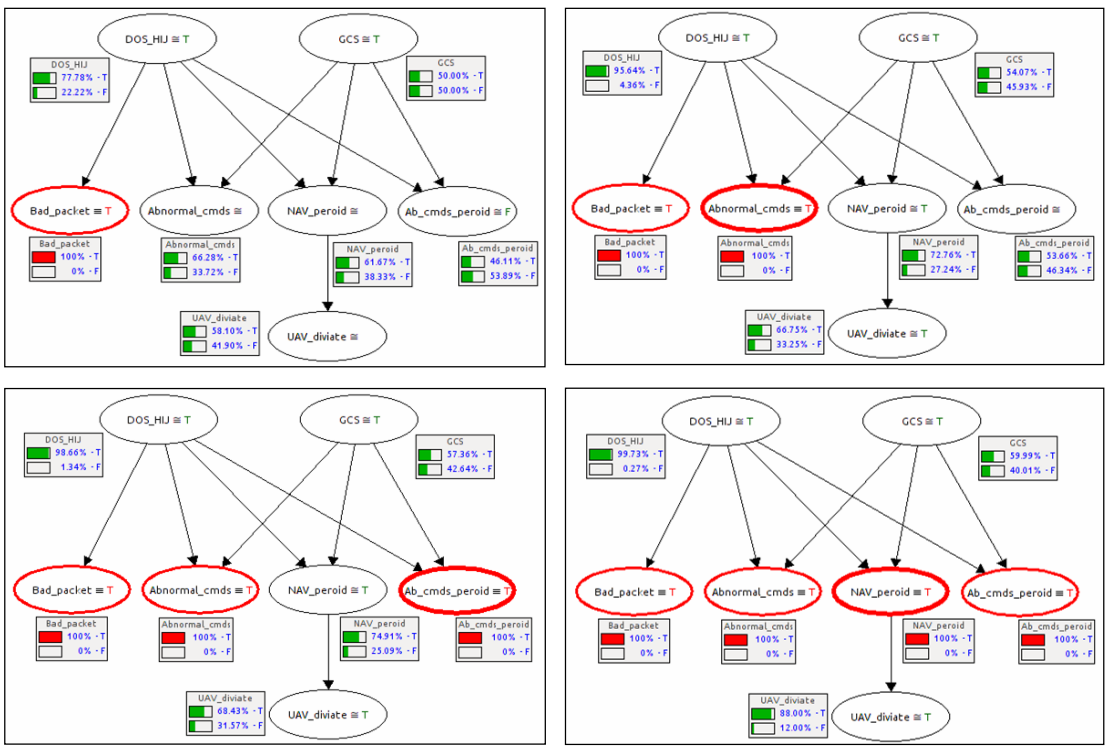

Fig.6.6 Bayesian network change diagram after input the ouput of monitors in R4.1、R4.2、R4.3、R4.4.

图 6.6 输入 R4.1、R4.2、R4.3、R4.4 监控器的监控结果后贝叶斯网络变化图

如图 6.6,当规约 R4.1、R4.2、R4.3、R4.4 的监控器判断结果作为已知条件输入贝叶斯网络后时,DOS 攻击的可能性不断增加,图 6.7 展示了无人机接受普通指令后 DOS 攻击监控器的值(RV_DOS.Value)发生阶梯式上升与贝叶斯网络诊断结果一致,当收到返航指令(RTL)后监控器重置为默认值(50%).

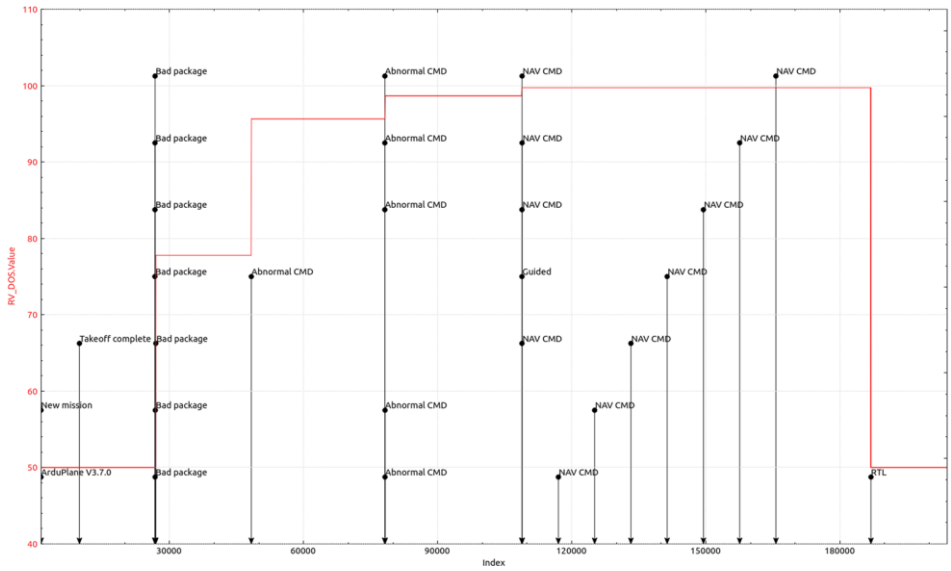

Fig.6.7 DOS attack experiment

图 6.7 DOS 攻击实验图

当 GCS 发现无人机受到劫持后由于劫持者比它发送指令的速度快得多,GCS 并不能有效控制无人机,因此无人机一般都有一个完全自主的飞行模式,在发现自己受攻击失控之后可以返回起飞点,避免继续受到攻击.

6.3 多无人机监控参数化实验

对于多架无人机需要遵守的共同性质的规约,我们可以采用参数化的方法,通过相同的规约生成一个带有不同参数的 FSM 来进行监控,以达到节约硬件资源,简化监控器生成工作量.例如,为了避免碰撞到建筑物我们制定规约:

V4:任意无人机收到起飞指令后 500 至 10000 个时间周期内,飞行高度要提升到大于 120 米

$$\forall x: G((\mathrm{CMD} == \mathrm{takeoff}) \wedge F[500{:}10000]\mathrm{ALT}(x) > 120m))$$

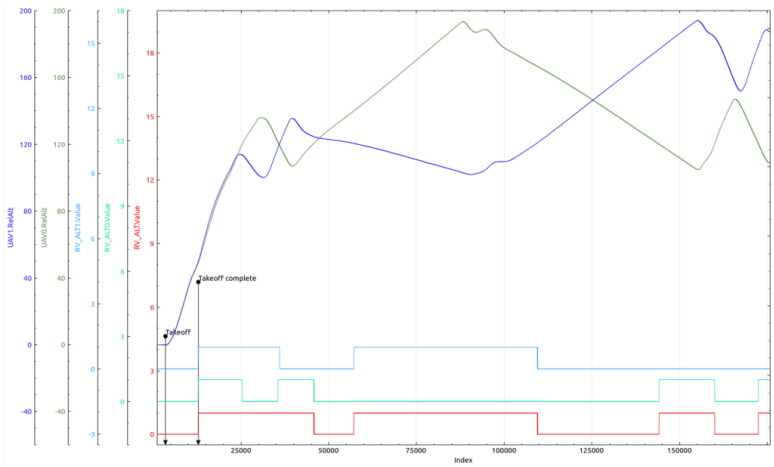

Fig.6.8 Parameterization experiment of multi-UAVs monitoring

图 6.8 多无人机监控参数化实验图

如图 6.8,本实验中一个时钟周期为 20ms,一般无人机会在收到起飞指令 500 个时钟周期内爬升到 50 米高度,两架无人机在完成起飞动作后飞行高度值(UAV0_ReAlt、UAV1_ReAlt)不断改变,当飞行低于 120 米时 UAV0

监控器值(RV_ALT0.Value)和 UAV1 监控器值(RV_ALT1.Value)发生改变,参数化监控器值(RV_ALT.Value)根据机群飞行高度变化而产生变化,只要有一架无人机低于指定高度监控器就发生改变,并产生报警.

7 总结与展望

本文提出一种基于运行时验证的无人飞行系统安全威胁检测方法,利用 DT-MTL 描述可能遇到的可靠安全性和保密安全性威胁,提出 UAS-DL 语言描述系统的监控规约.然后提出了基于交错自动机自动生成监控器的方法,并利用参数化的方法实现对多对象系统的监控.为了提高检测的准确性,研究了将运行时验证和贝叶斯网络结合的方法.采用实际的无人机开发和仿真平台 Ardupilot 进行了仿真实验,并设计了将监控器部署独立 FPGA 硬件上、避免对 UAS 计算资源过多占用的方法.实验表明上述方法能够有效检测 UAS 的安全威胁,并具有较好的运行效率.

在未来工作中,我们计划进一步研究更复杂的 UAS 攻击方式的监控方法,扩展时序逻辑语言和监控规约以增强其表达能力,并对多无人系统的协同安全监控进行更深入研究.

References:

[1] Johann Schumann, Patrick Moosbrugger, and Kristin Y. Rozier, Monitoring and Diagnosis of Security Threats for Unmanned Aerial Systems, Runtime Verification,2015,233-249.

[2] Aiello, A.M., Berryman, J.F., Grohs, J.R., Schierman, J.D.: Run-time assurance for advanced flight-critical control systems. In: AIAA Guidance, Nav. and Control Conf. AIAA (2010).

[3] Basin D, Klaedtke F, Zălinescu E. Algorithms for monitoring real-time properties[C]//International Conference on Runtime Verification. Springer Berlin Heidelberg, 2011: 260-275.

[4] Divakaran S, D'Souza D. Conflict-tolerant real-time specifications in metric temporal logic[C]//Temporal Representation and Reasoning (TIME), 2010 17th International Symposium on. IEEE, 2010: 35-42.

[5] Maler O, Nickovic D, Pnueli A. Checking temporal properties of discrete, timed and continuous behaviors[M]//Pillars of computer science. Springer Berlin Heidelberg, 2008: 475-505.

[6] Basin D, Klaedtke F, Müller S, et al. Runtime monitoring of metric first-order temporal properties[C]//LIPIcs-Leibniz International Proceedings in Informatics. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2008, 2.

[7] Backasch R, Hochberger C, Weiss A, et al. Runtime verification for multicore SoC with high-quality trace data[J]. ACM Transactions on Design Automation of Electronic Systems (TODAES), 2013, 18(2): 18.

[8] Reinbacher T, Rozier K Y, Schumann J. Temporal-logic based runtime observer pairs for system health management of real-time systems[C]//International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Springer Berlin Heidelberg, 2014: 357-372.

[9] Kiczales G, Lamping J, Mendhekar A, et al. Aspect-oriented programming[J]. ECOOP'97—Object-oriented programming, 1997: 220-242.

[10] Kiczales G, Hilsdale E, Hugunin J, et al. An overview of AspectJ[C]//European Conference on Object-Oriented Programming. Springer Berlin Heidelberg, 2001: 327-354.

[11] Spinczyk O, Gal A, Schröder-Preikschat W. AspectC++: an aspect-oriented extension to the C++ programming language[C]//Proceedings of the Fortieth International Conference on Tools Pacific: Objects for internet, mobile and embedded applications. Australian Computer Society, Inc., 2002: 53-60.

[12] Coady Y, Kiczales G, Feeley M, et al. Using AspectC to improve the modularity of path-specific customization in operating system code[C]//ACM SIGSOFT Software Engineering Notes. ACM, 2001, 26(5): 88-98.

[13] Gong W M, Jacobsen H A. Aspect-oriented c specification. Working technical draft, University of Toronto, 2008.

[14] Stefan Mitsch and André Platzer, ModelPlex: Verified Runtime Validation of Verified Cyber-Physical System Models. Runtime Verification, 2014, 199-214.

[15] Pierre Fraigniaud, Sergio Rajsbaum, and Corentin Travers. On the Number of Opinions Needed for Fault-Tolerant Run-Time Monitoring in Distributed Systems, Runtime Verification, 2014, 92-107.

[16] André Matos Pedro, David Pereira1, Luis Miguel Pinho, and Jorge Sousa Pinto.Monitoring for a Decidable Fragment of MTL-∫. Runtime Verification, 2015, 169-184.

[17] Aaron Kane1, Omar Chowdhury, Anupam Datta,and Philip Koopman. A Case Study on Runtime Monitoring of an Autonomous Research Vehicle (ARV) System. Runtime Verification, 2015, 102-117.

[18] Kim, A., Wampler, B., Goppert, J., Hwang, I., Aldridge, H.: Cyber attack vulnerabilities analysis for unmanned aerial vehicles. Infotech@Aerospace (2012).

[19] Leucker M, Schallhart C. A Brief Account of Runtime Verification [J]. Journal of Logic and Algebraic Programming. 2009, 78 (5): 293-303.

[20] Allan C, Avgustinov P, Christensen A S, et al. Adding Trace Matching with Free Variables to Aspect J [C]. In OOPSLA '05: Proceedings of the 20th annual ACM SIGPLAN Conference on Object Oriented Programming Systems, Languages, and Applications. New York, NY, USA, 2005: 345-364.

[21] Chen F, Jin D, Meredith P, et al. Monitoring Oriented Programming - A Project Overview [C]. In Proceedings of the Fourth International Conference on Intelligent Computing and Information Systems (ICICIS'09). 2009: 72-77.

[22] Colombo C, Pace G J, Schneider G. LARVA — Safer Monitoring of Real-Time Java Programs (Tool Paper) [C]. In Seventh IEEE International Conference on Software Engineering and Formal Methods (SEFM). November 2009: 33–37.

[23] Pnueli A. The Temporal Logic of Programs [C]. In 18th IEEE Symposium on Foundation of Computer Science (FOCS'77). Washington, D.C., USA, 1977:46–57.Subramanyam R, Krishnan MS. Empirical analysis of CK metrics for object-oriented design complexity: Implications for software defects. IEEE Trans. on Software Engineering, 2003,29(4):297 310.

[24] Gabbay D, Pnueli A, Shelah S, et al. On the Temporal Analysis of Fairness [C]. In 7th ACM Symp. on Principles of Programming Languages (POPL'80). Washington, D.C., USA, 1980: 163–173.

[25] Reinbacher T, Rozier K Y, Schumann J. Temporal-logic based runtime observer pairs for system health management of real-time systems[C]//International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Springer Berlin Heidelberg, 2014: 357-372.

[26] Gastin P, Oddoux D. Fast LTL to Büchi automata translation[C]//International Conference on Computer Aided Verification. Springer Berlin Heidelberg, 2001: 53-65.

[27] Muller D, Schupp P E. Alternating automata on infinite objects, determinacy and Rabin's theorem[C]//LITP Spring School on Theoretical Computer Science. Springer Berlin Heidelberg, 1984: 99-107.

[28] Rohde G S. Alternating automata and the temporal logic of ordinals[D]. University of Illinois at Urbana-Champaign, 1997.

[29] Chen F, Rou G. Parametric trace slicing and monitoring[C]//International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Springer Berlin Heidelberg, 2009: 246-261.

[30] Bauer A, Küster J C, Vegliach G. The ins and outs of first-order runtime verification[J]. Formal Methods in System Design, 2015, 46(3): 286-316.

[31] Medhat R, Joshi Y, Bonakdarpour B, et al. Parallelized runtime verification of first-order LTL specifications[R]. Technical Report CS-2014-11, University of Waterloo, 2014.

[32] Reger G, Rydeheard D. From first-order temporal logic to parametric trace slicing[C]//Runtime Verification. Springer International Publishing, 2015: 216-232.

[33] PEARL J. Fusion，propagation，and structuring in belief networks[J]. Artificial intelligence, 1986，29(3): 241-288.

[34] SPIRTES P, GLYMOU R C, SCHEINES R. Causality from probability[R]. 1989.

[35] XIE Xian-chao, GENG Zhi，ZHAO Qiang.Decomposition of structural learning about directed acyclic graphs［J]. Artificial Intelligence, 2006, 170(4): 422-439.

[36] http://reasoning.cs.ucla.edu/samiam/index.php.

[37] http://ardupilot.org/.

[38] Geist, J., Rozier, K.Y., Schumann, J.: Runtime observer pairs and Bayesian network reasoners on-board FPGAs: flight-certifiable system health management for embedded systems. In: Proceedings of the RV 2014, pp. 215–230 (2014).

[39] Mirkovic J., Reiher P.A Taxonomy of DDoS attack and DDoS defense mechanisms [J] .ACM SIGCOMM Computer Communications Review , 2004,34(2):39-53.