XXX工程

项目名称

形式化逻辑任务分析与分解算法研究报告

北京大学

2022年6月

项目名称

|  |  |
| --- | --- |
| 拟 制： |  |
| 审 核： |  |
| 批 准： |  |
| 会 签： |  |

**目录**

[2 形式化方法 1](#_Toc137903099)

[2.1 形式化方法研发背景 1](#_Toc137903100)

[2.2 形式化方法 2](#_Toc137903101)

[2.3 形式化语言 3](#_Toc137903102)

[2.4 其他相关技术 4](#_Toc137903103)

[2.4.1 同行评审 4](#_Toc137903104)

[2.4.2 软件测试 4](#_Toc137903105)

[2.4.3 软件错误捕捉 5](#_Toc137903106)

[2.4.4 硬件验证 6](#_Toc137903107)

[3 基于形式化语言的任务规划 7](#_Toc137903108)

[3.1 线性时序逻辑 7](#_Toc137903109)

[3.2 运动规划算法 10](#_Toc137903110)

[3.3 离散时序逻辑规划 12](#_Toc137903111)

[4 基于战场形式化逻辑任务分析与分解 16](#_Toc137903112)

[4.1 基于线性时序逻辑的任务定义 16](#_Toc137903113)

[4.1.1 行为类 16](#_Toc137903114)

[4.1.2 环境类 17](#_Toc137903115)

[4.1.3 约束类 17](#_Toc137903116)

[4.2 常见先验条例的形式化表达与任务的融合 18](#_Toc137903117)

[4.3 任务自动机的预处理算法设计 21](#_Toc137903118)

[4.3.1 自动机生成 21](#_Toc137903119)

[4.3.2 自动机减枝 23](#_Toc137903120)

[4.3.2.1 消除不可行的过渡 23](#_Toc137903121)

[4.3.2.2 消除不可达的转移边 23](#_Toc137903122)

[4.3.2.3 删除可分解的转移边 23](#_Toc137903123)

[4.3.3 偏序分析 24](#_Toc137903124)

# 形式化方法

## 形式化方法研发背景

随着电气技术的发展，各类实用系统正变得越来越复杂，并通过互联网和各种嵌入式系统，如智能卡、掌上电脑、移动电话、高端电视机等，大规模地接入日常生活。据估计，在1995年，我们每天要面对大约25个信息和通信技术设备。电子银行和电话购物等服务已经成为现实。每天通过互联网产生的现金流约为10.12亿美元。现代交通工具，如汽车、高速列车和飞机，大约20%的产品开发成本用于信息处理系统。我们对嵌入式系统的依赖使得它们的可靠运行具有很大的社会重要性。除了在响应时间和处理能力方面提供良好的性能外，没有恶性的错误也是主要的质量指标之一。

对于通讯系统而言，这些软件和硬件错误不会威胁到我们的生命，但可能对制造商造成重大的经济后果。正确的信息通信技术系统对公司的生存至关重要。上世纪90年代初，英特尔奔腾II浮点除法单元的漏洞造成了约4.75亿美元的损失，用于更换有缺陷的处理器，严重损害了英特尔作为可靠芯片制造商的声誉。行李处理系统的软件错误将丹佛机场的开放推迟了9个月，每天损失110万美元。一家大型飞机公司的全球在线机票预订系统将因订单丢失二十四小时而破产。

与安全有关的系统错误可能是灾难性的。阿丽亚娜-5型导弹、火星探路者号和空客家族飞机控制软件的致命缺陷曾登上世界各地报纸的头条。类似的软件用于安全关键系统的过程控制，如化工厂、核电站、交通控制和警报系统，以及风暴潮屏障。显然，这类软件中的错误会带来灾难性的后果。例如，放射治疗机Therac-25控制部分的一个软件缺陷在1985年至1987年期间导致6名癌症患者死亡，因为他们暴露在过量的辐射下。

关键应用对信息处理的日益依赖使业界提出了一个重要的需求:规划系统的可靠性是系统设计过程中的一个关键问题。规划系统的规模及其复杂性正在迅速增长，且不再是独立的，而是通常嵌入在一个更大的环境中，与其他几个组件和系统连接并相互作用。因此，它们变得更容易出错——缺陷的数量随着交互系统组件的数量呈指数增长。特别是，对交互系统建模至关重要的并发性和不确定性等现象很难用标准技术来处理。它们日益增长的复杂性，加上大幅度减少系统开发时间的压力，使得交付低缺陷的规划系统成为一项极具挑战性和复杂的活动。

形式化方法正是在这种背景下提出的，希望以数学的方式更可靠的应用于规划系统的设计。简而言之，系统验证用于确定所考虑的设计或产品具有某些特性。要验证的属性可以是非常基本的，例如，系统永远不应该达到无法进行任何进展的情况(死锁场景)，并且主要是从系统的规范中获得的。该规范规定了系统必须做什么和不做什么，因此构成了任何验证活动的基础。一旦系统没有满足规范的一个属性，缺陷就会被发现。只要系统满足从其规范中获得的所有属性，就认为它是“正确的”。

## 形式化方法

形式化方法提供了在设计过程中获得早期验证集成的巨大潜力，提供了更有效的验证技术，并减少了验证时间。让我们首先简要讨论形式方法的作用。简而言之，形式化方法可以被认为是“建模和分析ICT系统的应用数学”。他们的目标是用数学的严谨性来建立系统的正确性。它们的巨大潜力导致工程师越来越多地使用正式方法来验证复杂的软件和硬件系统。此外，根据IEC(国际电工委员会)的最佳实践标准和ESA(欧洲航天局)的标准，形式化方法是安全关键系统软件开发的“强烈推荐”验证技术之一。美国联邦航空局(FAA)和美国国家航空航天局(NASA)关于使用正式方法的调查结果报告得出结论在过去的二十年中，对形式化方法的研究导致了一些非常有前途的验证技术的发展，这些技术促进了缺陷的早期检测。这些技术伴随着强大的软件工具，可以用来自动化各种验证步骤。

软件形式化方法最早可追溯到20世纪50年代后期对于程序设计语言编译技术的研究，即J.Backus提出BNF描述Algo语言的语法[1]，出现了各种语法分析程序自动生成器以及语法制导的编译方法，使得编译系统的开发从“手工艺制作方式”发展成具有牢固理论基础的系统方法。形式化方法的研究高潮始于 20世纪60年代后期，针对当时所谓“软件危机”[2],人们提出种种解决方法,归纳起来有两类：一是采用工程方法来组织、管理软件的开发过程；二是深入探讨程 序和程序开发过程的规律，建立严密的理论，以其用来指导软件开发实践。前者导致“软件工程”的出现和发展，后者则推动了形式化方法的深入研究。经过30多 年的研究和应用，如今人们在形式化方法这一领域取得了大量、重要的成果，从早期最简单的形式化方法——一阶谓词演算方法到现在的应用于不同领域、不同阶段的基于逻辑、状态机、网络、进程代数、代数等众多形式化方法。形式化方法的发展趋势逐渐融入软件开发过程的各个阶段,从需求分析、功能描述(规约)、(体系结构/算法)设计、编程、测试直至维护。

在 70 年代后期，Pnueli将形式化语言应用于验证复杂计算机系统[3]。自90 年代初，形式化语言被应用于智能体控制领域，用于描述复杂的智能体任务。这些研究一部分关注于在单个复杂系统中保证一些不易建模的安全性约束，比如在交通规则约束下的自动驾驶控制、工厂环境的机械臂控制等；另外一部分研究则关注于大规模的多智能体系统的复杂功能执行，比如在火灾等危险环境下的消防员集群规划，未知环境中考虑能源补给，通讯保持的环境探索规划等等。

## 形式化语言

数学、逻辑和计算机科学中，形式语言（英语：Formal language）是用精确的数学或机器可处理的公式定义的语言。如语言学中语言一样，形式语言一般有两个方面: 语法和语义。专门研究语言的语法的数学和计算机科学分支叫做形式语言理论，它只研究语言的语法而不致力于它的语义。在形式语言理论中，形式语言是一个字母表上的某些有限长字符串的集合。一个形式语言可以包含无限多个字符串。

形式化方法的一个重要研究内容是形式规约(Formal Specification,也称形式规范或形式化描述),它是对程序“做什么”(what to do)的数学描述,是用具有精确语义的形式语言书写的程序功能描述,它是设计和编制程序的出发点,也是验证程序是否正确的依据。对形式规约通常要讨论其一致性(自身无矛盾)和完备性(是否完全、无遗漏地刻画所要描述的对象)等性质。形式规约的方法主要可分为两类:一类是面向模型的方法也称为系统建模,该方 法通过构造系统的计算模型来刻画系统的不同行为特征；另一类是面向性质的方法也称为性质描述,该方法通过定义系统必须满足的一些性质来描述一个系统。不同的形式规约方法要求不同的形式规约语言,即用于书写形式规约的语言(也称形式化描述语言),如代数语言OBJ[4]、Clear、ASL[5]、ACT One/Two[6]等;进程代数语言CSP[7]、CCS、π演算等；时序逻辑语言PLTL[8]、CTL、XYZ/E、UNITY、TLA[8]等；这些规约语言由于基于不同的数学理论及规约方法，因而也千差万别，但它们有一个共同的特点,即每种规约语言均由基本成分和构造成分两部分构成。前者用来描述基本(原子)规约,后者把基本部分组合成大规约。构造成分是形式规约研究和设计的重点，也是衡量规约语言优劣的主要依据。

## 其他相关技术

### 同行评审

软件验证同行评审和测试是实践中使用的主要软件验证技术[9, 10]。同行评审相当于由一组软件工程师执行的软件检查，这些工程师最好没有参与被评审软件的开发。未编译的代码不执行，而是完全静态地进行分析。经验研究表明，同行评审提供了一种有效的技术，可以捕获31%到93%的缺陷，中间值约为60%。虽然大多以一种相当特别的方式应用，但更专门的同行评审程序类型，例如，那些专注于特定错误检测目标的程序，甚至更有效。尽管它几乎完全是手工的，但同行评审是一种相当有用的技术。因此，在几乎80%的软件工程项目中使用某种形式的同行评审就不足为奇了。由于其静态性质，经验表明，使用同行评审很难发现诸如并发性和算法缺陷之类的细微错误。

### 软件测试

软件测试是任何软件工程项目的重要组成部分[11, 12]。软件项目总成本的30%到50%用于测试。与同行评审相反，同行评审静态地分析代码而不执行它，测试是一种实际运行软件的动态技术。测试对正在考虑的软件进行测试，并为其编译后的代码提供输入，称为测试。因此，通过强制软件遍历一组执行路径，即代表软件运行的代码语句序列来确定正确性。基于测试执行期间的观察，将软件的实际输出与系统规范中记录的输出进行比较。虽然测试生成和测试执行可以部分自动化，但是比较通常是由人来完成的。测试的主要优点是它可以应用于所有类型的软件，从应用软件(例如，电子商务软件)到编译器和操作系统。由于对所有执行路径进行详尽的测试实际上是不可行的;在实践中，只处理了这些路径的一小部分。因此，测试永远不可能完成。也就是说，测试只能显示错误的存在，而不能显示错误的不存在。测试的另一个问题是确定何时停止。实际上，要指出达到某个缺陷密度的测试强度是很困难的，而且几乎是不可能的——每数目的未注释代码行的缺陷的比例。

研究提供的证据表明，同行评审和测试在开发周期的不同阶段捕获了不同类型的缺陷。因此，它们经常一起使用。为了增加软件的可靠性，这些软件验证方法与软件过程改进技术、结构化设计和规格说明方法(如统一建模语言)以及版本和配置管理控制系统的使用相结合。大约10%到15%的软件项目以这样或那样的形式使用了正式技术。

### 软件错误捕捉

软件bug的定位是非常重要的[13, 14]。在维护期间修复软件缺陷的成本大约是早期设计阶段修复的500倍。因此，系统验证应该在设计过程的早期阶段进行。大约50%的缺陷是在编程期间引入的，这是实际编码发生的阶段。虽然只有15%的错误在最初的设计阶段被发现，但大多数错误是在测试期间被发现的。在单元测试的开始阶段，它的目标是发现组成系统的单个软件模块中的缺陷，典型的缺陷密度是每1000行(未注释的)代码中大约有20个缺陷。在系统测试开始时，这已经减少到每1000行代码中大约有6个缺陷，在这里，这些模块的集合构成了一个真正的产品被测试。在启动一个新的软件版本时，典型的可接受的软件缺陷密度大约是每1000行代码中有一个缺陷。

错误通常集中在几个软件模块中——大约一半的模块是没有缺陷的，大约80%的缺陷出现在一小部分模块中(大约20%)——并且经常发生在连接模块时。在测试之前检测到的错误的修复可以相当经济地完成。修复成本从单元测试中的大约1000美元(每个错误修复)显著增加到仅在系统运行期间证明缺陷时的最高约12,500美元。在软件设计过程中尽早找到缺陷的技术是至关重要的:修复它们的成本大大降低，并且它们对其余设计的影响较小。

### 硬件验证

硬件验证在硬件设计中防止错误是至关重要的[15, 16]。硬件制造成本高;在交付给客户后修复缺陷是困难的，而且质量期望很高。而软件缺陷可以通过提供有补丁或更新的用户——现在用户甚至倾向于预期并接受这种硬件错误，在交付给客户后修复非常困难，并且大多数需要重新组装和重新分发。这带来了巨大的经济后果。更换有问题的奔腾II处理器给英特尔造成了约4.75亿美元的损失。摩尔定律——电路中逻辑门的数量每18个月翻一番——在实践中被证明是正确的，也是生产正确硬件的主要障碍。实证研究表明，超过50%的ASIC(专用集成电路)在初始设计和制造后不能正常工作。芯片制造商在设计上投入了大量资金，这并不奇怪。硬件验证是设计过程中一个完善的部分。在典型的硬件设计中，设计工作只占芯片总时间的27%;其余部分用于错误检测和预防。

# 基于形式化语言的任务规划

## 线性时序逻辑

线性时序逻辑(Linear temporal logic, 简称LTL)由阿米尔·伯努利在1977年提出[3]，它将时间轴定义为一个线性序列，用于表示模型的动态语义，应用时序运算符描述从一个给定的状态开始的某一条路径上的事件。在LTL中，可以对关于路径上未来某时刻的公式进行编码，例如，一个事件最终将为真，或一个事件将为真直到另一个事件变为真，等等。

形式化语言描述的对象是一个转移系统。[17]转移系统是描述在输入作用下系统状态转移的图模型，具体定义如下。

**定义1**（转移系统，Transition System） 一个转移系统可以表示为元组

各个符号含义如下：

是状态集合，表示初始状态集合；

是输入集合，可以为一组控制或者动作；

表示一组转移关系；

是“原子命题”集合，可以表示系统在某一状态的观测性质;

是标签函数，将系统的状态与原子命题建立对应映射关系。

当系统状态集合和输入集合仅包含有限数量的元素时，则构成有限转移系统（Finite Transition System，FTS）。此外，一个转移系统也可以有无限多个状态，用来表示复杂混合的系统，如电网系统、计算机网络通讯系统等。系统的动态行为可以用一串状态转移序列来描述。

**定义 2** （状态转移， *Run*）给定一组输入的序列, 作用到转移系统中，生成一组状态转移序列, 使得

那么就称为一组状态转移。符号表示状态转移序列的长度。此外，针对这样的一组*Run*，在转移系统中可对应生成一组观测迁移。

**定义 3**（观测迁移， *Trace*）给定一组Run， , 在转移系统中根据标签函数的映射关系，对应生成一组。样的一组实际上就翻译了系统状态转移北航蕴含的观测转移过程。我们将所有原子命题序列组成的集合称为系统的语言, 即*。*

**定义 4**（LTL 语法，Syntax）给定一个原子命题集合, 时序逻辑公式在集合下可用如下基本算符表示：

这里，是一个原子命题，表示LTL公式。各算符含义为

表示“永真”状态，“永假”对应可定义为：

表示两个LTL公式的“与”，表示“非”。

“或”可以通过这两个基本算符表示

进一步，“推得”算符可以表示为：

（接下来）和（直到）为基本的时序逻辑运算符。

表示在下一步为真；表示直到为真，一直保持为真。根据基本时序逻辑运算符，可进一步定义（最终）和（总是），分别表示为

通过结合不同的时序逻辑运算符，可以得到更丰富的表达。比如，“”组合在一起表示最终永远存在；“”表示总是会有出现。这大大拓宽了传统命题逻辑表述的内容，增加了时间上的优先级顺序。

LTL公式蕴含的丰富语义可以解释一组无限长观测性质所表示的含义，其语义可以用数学语言规范化地表述。

**定义5**（LTL 语义，Symmetric） 给定一个基于原子命题的LTL公式，同时选取一组观测迁移，表示这组在第*k*个及以后的命题序列满足，具体语义解释如下

·, 解释为

·, 解释为

·， 解释为

·, 解释为 并且

·, 解释为

·， 解释为 , s. t. ,

,

如果一组完全满足LTL公式，可以表示为。

采用上面介绍的线性时序逻辑LTL，我们可以很好地刻画一系列运动规划任务。

**1）避障行为**

刻画了基本的避障任务，机器人最终要求到达区域，在前往区域的过程中，需要避免进入障碍区域。

**2）遍历行为**

表示机器人需要经过所有的可行区域，到达没有先后顺序，可以先到达再到达，也可以先到达再到达，需要保证不遗漏任何一个区域。

**3）顺序行为**

描述了按一定顺序到达的行为，机器人需要先到达，再到达，最后到达，顺序不可更改。

**4）巡逻行为**

表示机器人需要对目标区域进行不断巡逻，这样的巡逻行为可以是顺序的也可以是无序的。这里给出的例子描述的是机器人无序巡逻，要求机器人在每次巡逻中遍历所有的区域，但不要求每次巡逻的顺序。相应地，我们可以定义有序巡逻

表示按照的顺序不断循环往复地巡逻。

上面介绍了四种基本的运动规划问题，针对更复杂的时序运动规划问题，可以巧妙使用LTL逻辑算符组合简洁地表示出来。

## 运动规划算法

在给出了转移系统（Transition System，TS）和线性时序逻辑（Linear Temporal Logic，LTL）的基本概念后，本章选取了文献[18]中的单机器人运动规划问题作为例子，介绍基于时序逻辑的机器人运动规划算法流程，研究如何驱使机器人在给定的工作环境中实现由LTL公式描述的时序任务。

考虑一个机器人在二维平面中运动，它的运动学模型定义为

（3-1）

其中，表示机器人的位置，表示控制输入，可以驱动机器人执行运动任务。令表示一组原子命题集合，对于系统（3-1），我们可以定义一个观测映射

（3-2）

将机器人连续状态映射到有限的状态集中，原子命题代表的区域可以表示为机器人环境中的一个凸多边形，即可以表示为，对于。

之后，线性时序逻辑运动规划问题可以描述为如下一般形式。

**1）问题描述**

给定机器人模型（3-1），初始状态，观测映射（3-2）和LTL公式，构造一个控制输入，使得生成的机器人轨迹满足公式的任务要求。

接下来研究一个具体的任务。考虑一个机器人，运动学建模为式（3-1），运动在如图3-1所示正方形区域中，区域中拥有四个标签区域，满足映射关系（3-2），机器人初始位置在。期望的机器人任务为：先到达，再到达，然后到达，最后再回到，过程中避免碰到和。这个任务用线性时序逻辑公式可以表示为

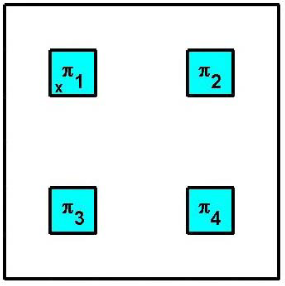


图3-1 环境中包含四个标签区域

**2）问题求解框架**

基于时序逻辑的运动规划问题，其核心求解思路是利用模型检测方法，即验证有限转移系统是否满足时序逻辑公式或规约（Specification）。Fainekos等人[18]提出的时序逻辑运动规划算法的基本框架可以精炼为三个步骤：

·连续状态离散化：将连续环境离散剖分为有限区域，构造可以描述机器人连续运动的离散状态转移模型。

·离散时序逻辑规划：采用模型检测方法和自动机理论，综合离散模型和线性时序逻辑公式，生成满足任务要求的离散运动轨迹。

·离散轨迹连续化：在保持任务满足的前提下，考虑机器人运动学或动力学约束，采用适当方法将离散轨迹连续化，并设计对应的连续控制器。

下面针对图3-1中的问题，采用上述步骤进行求解。

**3）连续状态离散化**

为了能够利用时序逻辑来处理连续系统，首先对机器人运动的环境进行离散剖分，将其剖分为有限个单元格。对平面区域进行离散剖分有多种方法，可以剖分成三角形、正方形、多边形等形状[19]。这里选择了三角形离散剖分法，一是因为针对复杂环境有很多高效的三角形剖分方法[20]，二是便于后续设计符合三角特点的连续控制器。

对连续区域作三角形剖分后得到一组三角形区域，满足映射关系，可以将机器人运动过程抽象为一个离散有限转移系统

（3-3）

其中，表示有限状态集合，表示初始状态，对应于连续域中的；是状态转移关系，表示为当且仅当单元格和相邻；反映的观测映射；为原子命题集。值得注意的是，为了保证是完满的，属于同一单元格的状态应该映射到相同的观测上，这对于离散时序逻辑规划及后续的连续化是十分重要的。



图3-2 三角剖分后得到的无向图有34个状态和49个转移过程

对连续区域采用三角剖分算法，划分成对偶的无向图，如图3-2，离散状态有34个区间，存在49个转移过程。

## 离散时序逻辑规划

给定机器人运动的有限转移系统和时序逻辑任务公式，可以用模型检测（Model Checking）的方法求解满足任务的离散轨迹。模型检测的概念最早在计算机领域提出，本质上是一种用于验证有限状态转移系统是否满足给定任务公式的算法过程[21]，其中任务公式通常用时序逻辑语言进行表述。模型检测方法的核心是自动机理论，这里首先给出一类Büchi自动机的定义。

**定义6**（Büchi自动机，Büchi Automata）[17] Büchi自动机可以表示成一个元组

其中，表示有限状态集，表示初始状态；表示所有原子命题的组合构成的集；表示状态间转移关系；为可接受状态集。

将一个基于原子命题集的命题序列输入到Büchi自动机中，可以生成一条包含无限个状态的序列，满足，这样的一组序列就是Büchi自动机生成的一条路径。若存在一个状态在路径中循环出现，则称该路径为的可接受路径，对应的命题序列称为可接受序列。将所有Büchi自动机可接受的命题序列的集合称为的语言。

给定LTL公式，可以构造对应的Büchi自动机，再结合离散转移模型构成乘积自动机，便可用模型检测的方法求解满足公式的可接受路径。

**定义7**（乘积自动机，Product Automata） 给定一个有限转移系统和LTL公式对应的自动机，则乘积自动机可以写成一个元组

其中，为有限状态集，为初始状态集；表示状态转移关系；为可接受状态集；为原子命题集；为标签函数。类似地，将所有乘积自动机可接受的命题序列的集合成为的语言。

在自动机理论中，寻找满足LTL公式的有限状态转移序列，可以转化为在系统的语言中寻找与Büchi自动机语言有交集的部分，即，如图3-3所示。

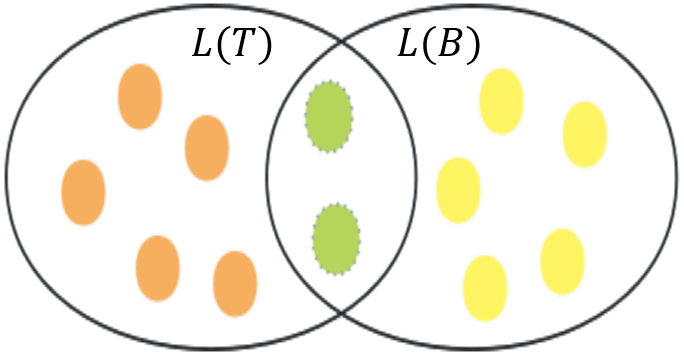


图3-3 离散系统语言和Büchi自动机语言存在交集

对于有限转移系统和公式对应的自动机结合成的乘积自动机，若其语言，即存在一条路径，满足在路径中循环出现，则称乘积自动机能够生成可接受路径。将可接受路径映射到离散有限转移系统中，则可以得到对应的离散路径，从而实现离散路径规划。若，则该乘积自动机不存在可接受路径，即机器人在当前环境下无法完成时序逻辑任务。

因此针对时序规划问题，可以抽象出形式化的求解方法：建立机器人离散模型，并将运动任务规范化地表述为LTL公式，采用模型检测方法将二者结合起来构造乘积自动机，利用乘积自动机的特性求解出可接受路径，或得到任务无解的结论。这种模型检测的方法经过不断发展，目前已经有很多封装好的检测工具（Model Checker），将输入离散模型和LTL公式作为输入，采用内嵌的算法过程进行求解，输出路径规划结果。

这里便采用了NUSMV[22]模型检测工具对上述离散时序逻辑规划问题进行求解，得到一组离散路径

状态转移过程见图3-4，可以看出求解得到的路径满足时序逻辑任务要求。

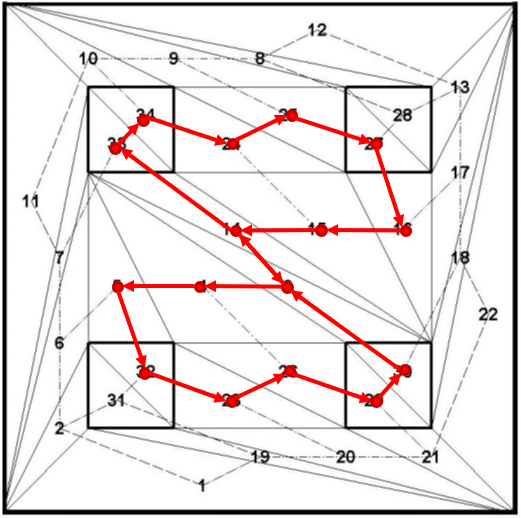


图3-4 通过模型检测方法求得的离散路径

经过上述两个步骤，我们得到了一条满足时序任务要求的离散路径，接下来的任务是根据这条离散路径构造连续的控制器，使得在该控制器作用下生成的连续路径仍能满足时序任务公式。

我们首先需要将离散路径拟合为连续的路径，使得对于离散系统中的转移过程，在连续系统中有对应的，其中有

（3-4）

符号为连续系统中的状态转移关系。

对于连续系统，补充定义，反映连续状态到原子命题集的观测映射关系。为了能够将任意的离散路径拟合为满足任务要求的连续路径，这里要求第一步的三角剖分满足如下的互模拟关系。

**定义8**（互模拟，Bisimulation） 给定一个剖分映射：，称其为互模拟的，当且仅当对，满足下面两条性质

·可观性：若，那么；

·可达性：若，当，对应有，， 满足，。

直观来讲，可观性表示同一单元格状态被映射到相同的观测命题上；可达性说的是，如果一个状态可以转移到相邻的单元格中的状态，那么与在同一单元格中的状态也必须转移到与在同一单元格中的状态。满足这样两个性质的连续系统和离散系统，就称为是互模拟的。显然，根据图3-4可以看出，第一步进行的三角剖分是一种互模拟关系，因而可以直接给出下面的命题。

**命题**：给定LTL公式和离散映射，若离散轨迹，那么对任意初始状态，总存在一个路径满足。



图3-5 满足时序任务要求的连续路径

考虑在三角形环境中的计算性能，采用文献[23]提出的控制框架设计满足互仿真特性的控制器。其基本思想是在每个三角形单元中创建一个仿射向量场，驱动机器人到达邻近的三角形单元，同时考虑了机器人运动的最大速度限制。拟合后的连续路径如图3-5所示，可以看出，得到的连续路径能够鲁棒地满足时序逻辑公式，因为在任意状态都可以被同一单元格中的状态替代。

# 基于战场形式化逻辑任务分析与分解

## 基于线性时序逻辑的任务定义

作为战场系统的特征，我们定义了以下几个种类的原子命题：

### 行为类

作为任务的核心，行为类原子命题直接用于描述智能体的具体的任务行为。根据战场需求，此类原子命题，由4部分组成，分别为主体，动作，区域，目标。一个单个的原子命题直接对应一个具体的任务，表示我方主体智能体以某种动作对区域内的某种目标执行某事，内部每个元素通过下划线进行连接。通过这样的形式，我们能限制约束形式化语言所涉及的内容，并规范化对任务的读取。

**主体：**主体表示我方能控制的作战单位，即选择某个类型或者某些类型的单位进行执行某事。更一般地，这里类型可以选择为各类泛化的组合。这里选择的是，以类型来区分，将主体分为：所有单位，侦察无人机，察打一体无人机，巡飞弹等等。同时也可以采用其他的分了方式，以小规模的战术集群，或者按区域分割的地形集群等等。

不过值得注意的是，想定中的每一个智能体都对应一些特有的功能。主体的选择与动作的选择会发生一定的耦合，这使得过于精细的主体划分会导致动作无法执行，比如说要求侦查无人机去进攻敌方，而侦查无人机却没有进攻的功能。

**动作：**这里动作根据类型核心分为三类，进攻，侦查，支持。进攻表示攻击目标区域内的敌人，侦查表示搜索目标区，支持是指到达目标区域后维持该区域的军事存在，打击来犯的敌人。这里动作会跟之后的博弈成进行策略上的指导。

**地点：**地点则是事先画分好的区域。常见的区域划分方法有，三角形法，安全走廊法，热点图法，网格法等。

**目标：**目标表示敌方所控制的战斗单位。和主体类似，目标主要由不同种类的智能体单元构成，包括指挥所，炮兵集群，步兵集群等等。结合主体和目标，能生成一下精细化的任务，比如说利用巡飞弹，对敌方的指挥所进行斩首。利用无人机，侦察敌方的炮兵位置等等。

### 环境类

环境类原子命题用于描述执行任务所在环境的特点，作为描述任务的辅助。根据战场需求，我们设置了以下几个种类的环境类原子命题作为约束：

此类原子命题，由2部分组成，分别为区域特征和类型约束。单个的原子命题对应一组特定的类型的约束。表示我方该类型的智能体需要绕开具有相关区域特征的所有区域，内部每个元素通过下划线进行连接。通过这样的形式，我们能限制约束形式化语言所涉及的内容，并规范化对任务的读取。

**区域特征：**可分为禁飞区，我方占区，敌占区，高价值区域等等。可基于战场环境确定，环境的特征分类和息息相关。

**类型约束**：包括我方侦察无人机，通讯无人机，打击无人机，也可以选择我方所有的智能体等等。

通过环境类组合，能实现在执行任务过程中，对敌方高价值区域优先进攻或者避免我方高价值的单位进入到危险区域等。

### 约束类

约束类原子命题用于描述执行任务时的特征，作为描述任务的细节。根据战场需求，我们设置了以下几个种类的约束行为作为约束类的原子命题：

此类原子命题，由4部分组成，分别为动作类型和执行特征。单个的原子命题对应一组特定的类型的约束，表示我方执行该动作的智能体需要以该种形式执行，内部每个元素通过下划线进行连接。通过这样的形式，我们能限制约束形式化语言所涉及的内容，并规范化对任务的读取。

动作类型：动作类型对应之前提及的进攻，侦察，支持。

执行约束：执行约束表示执行该动作的行为特征。包括不开火，有限弹药，无限制，生存优先，任务优先等约束。

## 常见先验条例的形式化表达与任务的融合

由于使用的用户通常并非专业的形式化语言研究人员，通常无法快速学会使用形式化语言公式描述。且在面向战场环境中，编写合适形式化语言公式的费时费力，因此我们提出面相用户的形式化语言这个概念。这里设计一类决策方法，使得用户不负责具体的形式化语言编辑，而是只选择意图，通过事先确定好的语言结构，保留对内容更改的灵活度，自动化生成目标的语言公式。

**战术结构：**通过对原子命题进行特定的组合，我们能形成一系列具有特定结构的偏序集合。而这样能生成固定偏序集合的形式化语言结构，我们称为战术结构。我们总共设计了六种战术结构。

顺序进攻：依次输入原子命题，根据顺序先后执行攻击任务。

基础进攻：并行的执行输入的原子命题，原子命题的动作皆为进攻。

顺序侦察：依次输入原子命题，根据顺序先后执行侦察任务。

基础侦察：并行的执行输入的原子命题，原子命题的动作皆为侦察。

顺序支持：依次输入原子命题，根据顺序先后执行支持任务。

基础支持：并行的执行输入的原子命题，原子命题的动作皆为支持。

同时，这些战术设定不是固定的，也可以设置出其他结构的组合，允许开发者自行二次开发。

通过对战术进行组合，我们可以确定最终的战争战法。首先，我们加入了波次的概念，即每个波次内可以调用多个战术，但是只有当这个波次的任务完全结束后，才能执行下一个波次的任务。

这里给出了预设的6个战法：

全面进攻:

先后对敌方重点区域进行攻击，攻击时考虑敌方不同单位的顺序，优先攻击火力强大的敌方阵地，再消灭敌方战略纵深。

波次 1

基础进攻 ： {主体: 红方全体, '地点': 002, 目标: 敌方全体 },

{主体: 红方全体, '地点': 001, 目标: 敌方全体 },

{主体: 红方全体, '地点': 003, 目标: 敌方全体,}

波次 2

基础进攻: {'主体': 红方全体 ,'地点': 001, 目标: 步兵团,}

{'主体': 红方全体,'地点': 001, 目标: 炮兵团

波次3

基础进攻: 主体: 红方全体, 地点: 001, 目标: 敌方全体,

步步为营:

逐步蚕食敌方区域，并占据已经攻下的区域，避免敌军反扑或者逃跑。

波次 1

基础进攻 主体 : 红方全体, 地点 : 001, 目标: 步兵团

主体 : 红方全体, 地点 : 001, 目标: 炮兵团

波次2

基础进攻 主体: 红方全体, 地点 : 001, 目标: 敌方全体

波次3:

基础支持：主体: 红方全体,地点: 001, 目标: 敌方全体

基础进攻: 主体 : 红方全体, 地点: 002, 目标: 敌方全体

波次4:

基础支持: 主体 : 红方全体, 地点 : 002, 目标: 敌方全体

基础进攻: 主体 : 红方全体,地点: 003, 目标: 敌方全体

波次5

基础支持: 主体 : 红方全体, 地点: 003, 目标: 敌方全体

围点打援：

攻击核心区域，并打击两翼来犯的敌人。

波次 1:

顺序侦察: 主体 : 红方全体, 地点 : 002, 目标: 敌方全体

主体 : 红方全体, 地点 : 001, 目标: 敌方全体

主体 : 红方全体, 地点: 003, 目标: 敌方全体

波次2:

基础支持: 主体 ：红方全体, 地点: 003, 目标: 敌方全体

主体: 红方全体, 地点 : 002, 目标: 敌方全体

顺序进攻: 主体 : 红方全体, 地点: 001, 目标: 步兵团

主体 :红方全体, 地点 : 001, 目标: 炮兵团

主体 :红方全体, 地点: 001, 目标: 敌方全体

占场扫描

占据目标区域，对潜伏的敌人进行扫荡。并最终清理目标区域。

波次1

顺序侦察: 主体: 红方全体, 地点: 002, 目标: 敌方全体

主体: 红方全体, 地点: 001, 目标: 敌方全体

主体: 红方全体, 地点: 003, 目标: 敌方全体

波次2

顺序进攻 主体 : 红方全体, 地点 : 001, 目标: 步兵团

主体 : 红方全体, 地点: 001, 目标: 炮兵团

主体 : 红方全体, 地点: 001, 目标: 敌方全体

基础进攻: 主体: 红方全体, 地点 : 002, 目标: 敌方全体

主体: 红方全体, 地点 : 003, 目标: 炮兵团

波次 3 基础支持 主体 : 红方全体, 地点: 003, 目标: 敌方全体

主体: 红方全体, 地点: 002, 目标: 敌方全体

基础侦察: 主体 : 红方全体,地点: 002, 目标: 敌方全体

主体 : 红方全体, 地点: 001, 目标: 敌方全体

重点防御 :

根据敌方的动向，在兵力不足时，优先包围我方单位，并时刻观察敌军动向。

波次 1:基础侦察: 主体: 红方全体, 地点: 002, 目标: 敌方全体

主体: 红方全体, 地点: 001, 目标: 敌方全体

主体: 红方全体, 地点: 003, 目标: 敌方全体

基础支持:主体: 红方全体,地点: 003, 目标: 敌方全体

主体: 红方全体,地点: 002, 目标: 敌方全体

防御反击 :

面对敌军突袭，保护我方指挥区域，侦察全局。在保卫住突袭后，对敌方进行反击。

波次1: 基础侦察:主体: 红方全体, 地点: 002, 目标: 敌方全体

主体 : 红方全体, 地点: 001, 目标: 敌方全体

主体: 红方全体, 地点: 003, 目标: 敌方全体

基础支持:主体: 红方全体,地点: 003, 目标: 敌方全体

主体: 红方全体,地点: 002, 目标: 敌方全体

波次2: 基础进攻: 主体 : 红方全体, 地点: 002, 目标: 敌方全体

主体: 红方全体,地点： 003, 目标: 炮兵团

## 任务自动机的预处理算法设计

传统的形式化语言决策方法使用了有限转换系统和Bchi自动机之间的笛卡尔积，这使得最终获得的搜索的解空间将发生状态爆炸。这些缺点限制了大规模的LTL公式或者多智能体的应用，使得传统形式化多智能体任务决策只停留在少数的任务和智能体中。我们采用新的任务分析算法，通过分析由LTL公式转换获得的自动机，获取一系列不可拆分的能够影响任务动态的最小动作，并分析获取这些任务内部的关系。最终以一个偏序集的形式对任务关系进行描述。在获取形式化语言LTL公式后，先会利用基础的转换方法，将LTL公式转换为Bchi自动机，之后会利用模型检测技术将Bchi自动机分析得到一系列强时序联系的任务序列，并通过对任务时序，冲突的分析，获取描述这些时序联系的偏序集。这样极大的减小计算的复杂程度，进而提高计算效率。

### 自动机生成

由LTL公式所转换而来的NBA（Nodetermine Bchi automata非确定性Bchi自动机）可以表示成一个元组

（4-1）

其中，表示有限状态集，表示初始状态；表示所有原子命题的组合构成的集；表示状态间转移关系；为可接受状态集。

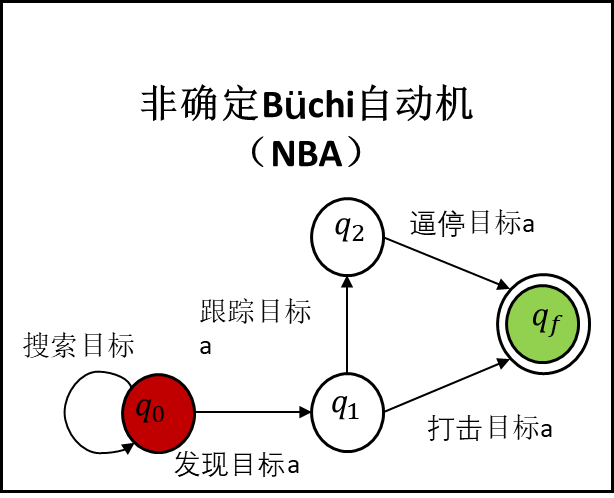


图4-1 非确定Bchi自动机转换关系

在元组*B*中搜索得到的可行性路径是充分的，但是由于线性时序逻辑的限制，命题集只能通过顺序时序连接起来，即可行性的任务序列都是串行的，在完成上一个任务的情况下才能执行下一个任务，这在复杂动态的战场环境中无法发挥智能体集群的高通讯，快变化的优势。因此我们先利用形式化任务分解的方法，对可能并行的命题序列进行独立性判断搜索，将多种可行路径归纳为由可并发执行和必须顺序执行关系所构成的偏序集，从而放宽任务执行的优化空间，且加强任务执行的效率。

利用异步执行的分发机制，避免无关任务之间的堵塞，仅保留必要的时序关系，根据并行裕度进行排序，选取宽度最大，即约束最少，最为灵活的的命题偏序集*P*，对子集群进行以最小执行时间为目标的任务分发，这样只有标为顺序执行的任务才需要等待前置任务的完成，而相互独立的任务将无时间顺序的要求，从而实现执行时间最短。同时，我们可以定义开放性的协同任务，通过对子智能体的能力组合，形成含有多个智能体协同执行的协同任务，由于原子命题的包容性，协同子任务可以是多种无人机、无人艇之间的有序结合，在任务分发层面，需要参与智能体同时执行该任务，并由具体的控制层算法实现，保证整个框架的完备性。

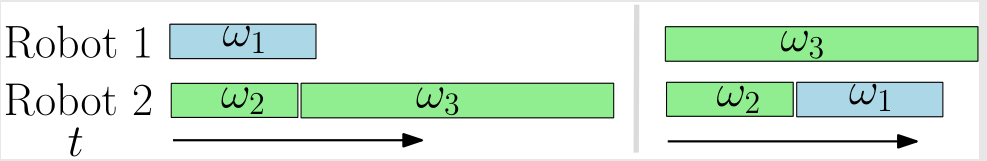


图4-2 串行与并行执行

### 自动机减枝

在获取得到自动机*B*后，可能会出现过多的冗余边和节点。这些冗余的边一方面具有不可执行性，另外一方面具有可以通过一系列子任务执行实现同样的效果。通常来说，例如，针对一个侦查任务，需要侦查一系列的目标点，以及布置对应的警戒措施，而某些冗余的边会提出不合理的要求，比如说同时布置所有的警戒点，一口气侦查所有的目标等。这样既没有指导意义，也会造成搜索空间的指数爆炸，因此，我们首先对自动机*B*进行减枝处理。具体来说，NBA的剪枝包括以下三个步骤。

#### 消除不可行的过渡

对于*B*中的任意转移边，若满足转移边所需要的战斗单元数量已经超过当前组织所能够提供的数量，则对于所考虑的系统是不可行的。我们可以通过检查是否存在一组战斗单元以同时到达一系列的目标区域，并执行相关的动作，来验证。或者说是多个战斗单元组合完成相对复杂的目标任务。如果这些都是不可行的，那么则在剪枝自动机中去除这样的过度。

#### 消除不可达的转移边

在自动机中会出现无法从初始状态达到的节点，通常来说，这是由于客观条件的限制导致的某些边被认为无效，或者是一些冗余节点之间边的去除所造成的。此类转移和节点的严格定义为对于*B*中的任何节点*q*，如果不能从任何初始状态到达，则称为无效状态;或者它不能达到任何可接受的状态，而这些状态反过来又可以从它自身到达，那么也是无效状态。无效状态不能成为接受路径的一部分，从而在自动机的修剪过程中将被删除。

#### 删除可分解的转移边

由于多智能体系统的集中式特性，强制两个或多个子任务在同一时刻实时终止是不现实的。因此，如果可能的话，任何需要同时满足几个子任务的转换都被分解成等效的转换。可分解转换在的正式定义如下：

一个从到的可分解的，当且仅当有另外一个节点满足

（4-2）

且有

（4-3）

换句话说，一个可分解转换总可以由其他的转换实现逐步，且不会引入额外的约束。如图6.10所示，可分解的边将在自动机中被删除。因此在自动机的减枝过程中最终只会保留不可删除的边，也就是无法拆分的最小的能推动整体任务发展的任务单元。这些保留下来的子任务最终将组成分发给排级战斗单元的战斗层任务。

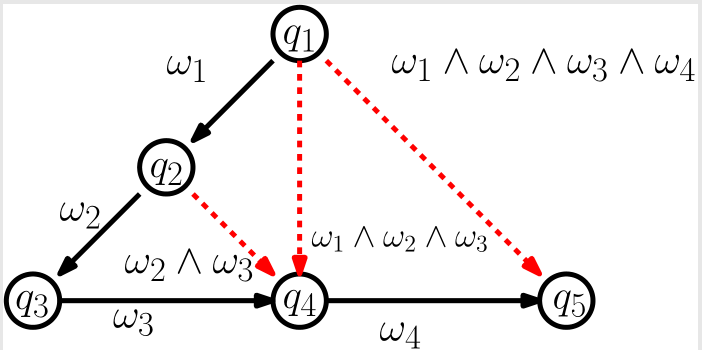


图4-3 自动机剪枝方法

在算法上，此类可分解性可以通过简单地组合和比较与每个转换相关的命题公式来检查。在以往的实践中，自动机的减枝过程能够减少10%的节点以及80%-95%的边，这意味着最终采用图搜索得到的计算空间将极大的减小，以迅速完成中枢规划要求。

### 偏序分析

在战场环境中，战场任务是由紧凑的时间任务公式所组成的，一方面要根据严格的时间要求加以约束，另一方面需要结合总体任务，对手集群态势以及己方集群态势信息和战场变化，对每个任务的具体分配以不同优化目标的决策，包括安全优先，效率优先，损失最小，战果最大等等的决策逻辑。因此，获得绝对满足任务要求的偏序分析尤为重要，与简单的可达性任务不同，时间任务对子任务的排序有严格的约束：例如，任务无人车在续航和战损容许的情况下，不断在争议区B执行驱离任务；在续航不足或战损严重的情况下返航。

表示返航任务必须在驱离任务实现后，或者是战损事件发生后，才能执行返航这样的优先级相对较低的任务。因此，对子任务之间的排序约束进行抽象总结是正确性的关键，若提前执行返航任务，则会导致整体任务的失败。本模块给出如何从修剪过的自动机中提取子任务并进行偏序分析，并给出了一个高效的anytime（任意时间）算法，以保证在在时间限制内能知道次优的可行性方案。

**定义9**（子任务分解） 考虑一个满足的可接受运行，其中且。那么一组可能的将原始任务分解为子任务将在如下被定义，这是一组二元数组：

（4-4）

其中，是子任务的序号，且子任务满足两个条件：和，其中。

换而言之，一个子任务由它的序号和它一系列位置和动作集合所构成。它的序号不可忽略，因为同样的子任务很可能在一次运行中出现多次，必须通过不同的序号将他们区分开。此外，此上的两个任务要求每个子任务仅保留必须的带转移的标签，举一个简单的例子若{侦查a，攻击a，侦查a& 攻击a}都能满足一次转移，那么只有{侦查a，攻击a}会被选中作为本质上的必须子任务。

与偏序的定义类似，子任务之间是部分有序的关系，同时他们之间关系是非自反的、传递的和不对称的。在极端情况下，若序列内完全不包含时序关系，那么这些任务将通过环境约束和本身的执行价值进行对应的决策执行，若这些序列包含了所有的时序关系，比如在战斗任务中，必须先炮击，再坦克突击，最后步兵占领，那么这些任务就将严格的按照时间关系进行执行。

值得注意的是，生成所有偏序集的完整集（用*P*表示）的计算成本很高。它包含从初始状态集到接受状态集的所有简单路径的计算。给定剪枝自动机，最坏情况下计算复杂度为*O(Q + F)*。因此，对于实时应用程序，设计一种可以随时使用的算法是比较有益的，这种算法被称为anytime算法。总能在给定的时间预算内生成至少一个有效的偏序集，同时在时间允许的情况下增加更多的偏序集。

在本模块的其余部分，我们详细描述了一个随时算法，该算法构造了给定修剪自动机的所有偏序集合{*P*}。算法如下所示：

|  |
| --- |
| **任意时间偏序分析算法** |
| 输入：修剪后的自动机 |
| 输出：偏序集和对应的子任务序列 |
| 生成已发现的接受词集合 |
| 从初始状态集合遍历状态：  若当前时间超出时间预算，则跳出循环 |
| 从点开始在深度优先搜索直到找到一个可接受的词 |
| 当深度优先搜索尚未结束：  若不在内，将加入中  初始化子任务序列  初始化严格的偏序集合  遍历每一对偏序关系：  核验去除该关系后，是否都可接受  若接受则去除该偏序  若不接受则保留该偏序集  将加入到中  将当前偏序集合加入到中 |
| 继续深度搜索 |
| 返回 |

在获取任务偏序集合后，依据推演环节获得的子任务作战危险性、集群完好度和胜利概率量化指标，子任务消耗特殊资源等量化指标，从而实现从态势理解，到最终的任务执行任务闭环。

1. Backus J. Programming in America in the 1950s—some personal impressions. A History of Computing in the twentieth century: Elsevier; 1980. p. 125-35.

2. 杨芙清, 梅宏, 吕建, 电子学报 金J. 浅论软件技术发展. 2002;30(S1):1901.

3. Pnueli A, editor The temporal logic of programs. 18th Annual Symposium on Foundations of Computer Science (sfcs 1977); 1977: ieee.

4. Goguen JA, Winkler T, Meseguer J, Futatsugi K, Jouannaud J-P. Introducing obj: Springer; 2000.

5. Supalla SJ, Cripps JH, Byrne APJAaotd. Why American sign language gloss must matter. 2017;161(5):540-51.

6. Lamport L. Introduction to TLA. 1994.

7. Schneider S. Concurrent and real-time systems: the CSP approach: John Wiley & Sons; 1999.

8. Chan JY, Bauer CFJJoRiST. Effect of peer‐led team learning (PLTL) on student achievement, attitude, and self‐concept in college general chemistry in randomized and quasi experimental designs. 2015;52(3):319-46.

9. 黄荦, 工业控制计算机 丁J. 民用飞机机载软件同行评审过程研究. 2014(8):36-7.

10. 电脑开发与应用 景J. 软件项目的同行评审. 2011;24(11):68-70.

11. 安金霞, 王国庆, 李树芳, 软件学报 朱J. 基于多维度覆盖率的软件测试动态评价方法. 2010(9):13.

12. 张新华, 科技视界 何J. 软件测试方法概述. 2012.

13. 河南师范大学学报：自然科学版 李J. 计算机应用系统中错误信息的捕捉和恢复. 1995;23(4):3.

14. 陕西教育：高教版 宋J. VB程序的调试和常见错误的捕获及错误处理方法. 2011(7):2.

15. 李文, 王恒才, 计算机研究与发展 唐J. 一种CPU芯片硬件验证调试平台的设计与实现. 2003;40(6):5.

16. 李鸿翔. 基于VMM的硬件验证技术研究及应用: 哈尔滨工程大学; 2010.

17. Baier C, Katoen J-P. Principles of model checking: MIT press; 2008.

18. Fainekos GE, Kress-Gazit H, Pappas GJ, editors. Temporal logic motion planning for mobile robots. Proceedings of the 2005 IEEE International Conference on Robotics and Automation; 2005: IEEE.

19. Choset H, Lynch K, Kavraki L, Burgard W, Hutchinson S, Kantor G, et al. Robotic Motion Planning: Foundations and Implementation. 2004. preparation.

20. Van Kreveld M, Schwarzkopf O, de Berg M, Overmars M. Computational geometry algorithms and applications: Springer; 2000.

21. Clarke EM, editor Model checking. International Conference on Foundations of Software Technology and Theoretical Computer Science; 1997: Springer.

22. Cimatti A, Clarke E, Giunchiglia E, Giunchiglia F, Pistore M, Roveri M, et al., editors. Nusmv 2: An opensource tool for symbolic model checking. International Conference on Computer Aided Verification; 2002: Springer.

23. Belta C, Habets L, editors. Constructing decidable hybrid systems with velocity bounds. 2004 43rd IEEE Conference on Decision and Control (CDC)(IEEE Cat No 04CH37601); 2004: IEEE.