1.

2018 届研究生硕士学位论文

分类号：_____ 学校代码：10269

密　级：_____ 学　　号：51151500079

華東師範大學

East China Normal University

硕 士 学 位 论 文

MASTER'S DISSERTATION

论文题目：
自主移动机器人空间永恒探索算法的符号模型检测方法

院　　系：	计算机科学与软件工程学院
专 业 名 称：	软件工程
研 究 方 向：	高可信计算理论与技术
指 导 教 师：	张民 副教授
学位申请人：	蔡晓伟

2017 年 11 月

Dissertation for master degree in 2018 University Code: 10269

Student ID: 51151500079

EAST CHINA NORMAL UNIVERSITY

On Symbolic Model Checking of Mobile Robots Perpetual Exploration Algorithm

Department:	School of Computer Science and Software Engineering
Major:	Software Engineering
Research direction:	Trustworthy Computing Theory and Technique
Supervisor:	Assoc Prof. Zhang Min
Candidate:	Cai Xiaowei

2017.11

蔡晓伟 硕士学位论文答辩委员会成员名单

姓名	职称	单位	备注
-	-	-	-
-	-	-	-
-	-	-	-

摘　要

20 世纪 70 年代末，随着智能机器人学、传感技术和分布式系统的快速发展，自主移动机器人 (autonomous mobile robots) 逐渐步入了人类的生活当中，尤其是在工业生产、空间探索和军事等领域得到了广泛的应用。多自主移动机器人协作对未知空间探索问题是近期该领域的一个研究热点。

移动算法是多自主移动机器人协作完成空间探索问题的核心，自主移动机器人通过视觉传感器获取周围环境中其他机器人的位置信息，将位置信息与移动算法进行匹配，做出对应的移动决策。未知空间模型由连续二维欧几里德空间模型，逐渐演化成为有限位置的离散空间模型。在离散空间模型中，机器人有三种调度模型：完全同步模型 FSYNC(Full-synchronous model)、半同步调度模型 SSYNC(Semi-synchronous model)、异步调度模型 ASYNC(Asynchronous model)。验证不同调度模型下，移动算法是否满足空间探索性质大多数使用的是手动推演的方法，手动推演的方式不仅过程冗长复杂，而且演算过程中容易出现错误。对于异步调度模型中，由于存在使用过时环境位置信息作为移动决策依据的情况，根本无法使用手动推演的方式进行验证。

而形式化验证方法因其自动性、高效性和严谨性，弥补了手动推演的不足之处，逐渐被用于各种自主移动机器人算法的验证。近期，B Bérard 使用 DVE 形式化语言对机器人移动算进行了建模，并使用 DiVinE 和 ITS 形式化工具验证了机器人移动算法模型是否满足永恒探索性质。另外，Ha 等人使用 Maude 重写逻辑形式化语言实现移动机器人永恒探索算法的验证。已知的这些方法多针对某种特定的调度模型进行建模, 并通过人为设定一个具体的初始状态进行验证. 然而自主移动机器人算法中系统的初始状态多是未知的。

本文提出自主移动机器人符号模型检测方法, 该方法不依赖某个具体的初始状态, 且适用于不同的调度模型. 同时, 借助符号模型检测的高效性, 有效避免状态爆炸问题. 利用 nuXmv 符号模型验证工具对机器人探索算法在三种调度模型: 完全同步模型、半同步模型 SSYNC、异步模型进行建模并利用 LTL 公式定义算法的永恒探索性质, 最终实现移动算法的形式化验证. 验证结果表明在假设初始状态未知的条件下依然可验证性质不被满足并找到反例. 同时, 实验数据表明了符号模型检测对自主移动机器人算法形式化验证的可行性与高效性.

关键词: nuXmv，移动机器人，空间探索，符号模型检测，形式化验证

ABSTRACT

In recent years, with the rapid developments of society and science, the role that autonomous mobile robots play in people's daily life .It has a wide range of applications in some areas,such as electronics industry、aerospace、automotive production and so on.As human activity space.With the expansion of human activity space,started to hope that autonomous mobile robots can explore unknown space in place of human.researchers envision groups of mobile robots self-organizing and cooperating to solving some predefined tasks,including exploration of unknown environments,without any central coordinating authority.

Exploration Algorithm is the core problem for the autonomous mobile robot to complete exploration task,the movement of robot depends on the exploration algorithm.In the past,the vast majority of research on mobile robots considers that mobile robots are locating in a continuous two-dimensional Euclidian space. A recent trend was to the discrete model take the place of the classical continuous mode. In the discrete model,there exist three different scheduling modes called full synchronous model(FSYNC), semi-synchronous model(SSYNC) and asynchronous model(ASYNC).There exist many research finding,such as the Min-Algorithm、the Max-Algorithm and so on,depending on handwritten proofs,which, particularly in asynchronous execution models,are both cumbersome and error-prone.

Formal methods for the automatic, rigorous and efficient has been used to verify the correctness of mobile autonomous robots algorithm。for example, using DiVinE and ITS tools to model check mobile robot protocols.and applying Maude to model the Min-Algorithm.

We use the state-of-the-art symbolic model checker nuXmv to verify the mobile robot

perpetual exploration algorithm under three different scheduling modes called full synchronous model (FSYNC), semi-synchronous model (SSYNC) and asynchronous model(ASYNC). Experimental results show that even without providing specific initial states a counterexample can be found in our approach for the perpetual exploration property, which concides with the existing verification result which is obtained by model checking with specific initial states. Meanwhile, the experimental data shows the feasibility and efficiency of symbolic model checking in the formal verification of autonomous mobile robot systems.

Keywords: *nuXmv, LTL, Mobile robots, space exploration, symbolic model checking*

目录

第一章　绪 论

1.1　研究背景

机器人是一类自动执行工作的机械装置，既可以按照人类指令，又可以预先编写的指令，也可以根据人工智能指定的工作原则行动。机器人的诞生是协助或者取代人类完成工作，机器人可以分为五大类：工业机器人、娱乐机器人、家庭机器人、竞赛机器人和军用机器人。目前像工业机器人、娱乐机器人、家庭机器人相关的技术十分成熟，而且已经应用在人类的工业生产和日产生活当中。竞赛机器人处于一种发展和研究的阶段，目前最大的竞赛机器人是机器人世界杯（RoboCup），它是一个国际合作项目。其目标是到 2050 年左右，研发出完全自主仿人的机器人足球队。军用机器人主要用于敌情侦查、物资运输、后勤保障等等。军用机器人又可以细分为地面机器人、水下机器人和空间机器人。地面机器人主要是指轮式或者履带式车辆，通过远程操控，是机器人完成危险任务，如拆卸炸弹。水下机器人有两种：有人机器人和无人机器人，有人潜水机器人应变能力强，并且机动灵活，便于处理复杂水下作业任务，同时危险度高，价格昂贵。空间机器人代替人类完成太空科学实验、舱外操作、空间探索等活动，可以大大降低风险。

最早诞生由计算机系统控制的自主移动机器人名叫 Shakey，是 1966 年至 1972 年间，由美国斯坦福国际研究所 (Stanford Research Institute, SRI) 研制 [1], 主要用于研究在复杂的环境下利用人工智能技术对机器人的运动推理、路径规划 [2]。但是受限于当时的计算机技术发展水平，Shakey 运算速度缓慢，往往需要数小时的时间来进行图像处理和路径规划。后续随着高精度地理环境信息传感器、

高速中央处理器的诞生，计算机技术飞速发展，人类开始对移动机器人展开了更深层此的研究。

目前，许多应用设想在没有中央调度结构的情况下，移动机器人通过自主组织和相互协作，共同完成指定任务，包括地图的构建、环境检测、城市搜索救援、表面清理、危险区域监控、未知空间的探索等等 [9]。在这类系统中机器人移动算法是机器人协作完成任务的关键，机器人根据移动算法做出移动决策，进而完成预定任务。

本文中主要是研究自主移动机器人空间探索问题 [5][6]，空间探索是指空间区域巡查，探知空间区域地理信息。空间探索问题中有两种典型的探索方案：探索终止 (exploration with stop)[11] 和永恒探索 (perpetual exclusive exploration)r5。探索终止是空间中所有的位置都被机器人探索之后，达到某种状态，所有机器人永远停止移动。永恒探索是空间中所有的机器人对空间中的每个位置，都进行重复访问，这个过程永不停止。

移动空间模型由原始的连续二维欧几里德空间模型, 逐渐演化成为有限位置的离散空间模型。离散空间使用图来描述，图的结点代表空间中机器人可以到达的位置，图的边表示机器人可以由一个位置结点到达相邻的结点的路径。这样塑造空间模型更加简便直观，更加关注机器人和空间位置结点数量关系对移动算法设定的影响。离散空间模型中，每个位置结点在同一时刻至多只有一个机器人占据。机器人有以下三个特点：1、机器人无记忆存储器，即机器人无法记录历史移动过程;2、无方向传感器，机器人在离散空间中，无法辨别方向，并且没有方向偏好。3、无通讯功能，即机器人之间无法接收或者发送消息；4、拥有视觉传感器，通过视觉传感器机器人可以获取空间中其他机器人的位置信息；5、运算单元，处理移动算法匹配问题，通过计算给机器人下达移动指令，控制机器人的移动；6、移动装置，通过移动装置，机器人可以沿着可移动的路径做出移动动作；7、匿名性，机器人无法通过无表进行区分，即所有机器人都是相同的，且在同一个离散空间系统中，所有机器人执行相同的移动算法。

设计机器人移动算法，并保证机器人设定移动算法之后，能满足探索终止或者永恒探索是目前机器人巡逻问题的一个研究热点。机器人移动算法的设计和验证，大多是以手动推演的方法为主。手动推演不仅过程冗长复杂，在推演的过程中也容易出现错误或者疏漏，尤其是在某些特定的设置条件下，手动推演方式根本无法进行验证。形式化验证方法因其自动性、高效性、严谨性逐渐被应用于各种移动机器人算法的验证。

nuXmv 是一个新的形式化的符号模型检测工具，不仅集成了最新的 SAT、BDD 模型检测的算法，而且还包含验证性能高效的基于 SMT 的验证技术。使用 nuXmv 工具建模验证时，可以根据实际需求选择验证方式，十分灵活。nuXmv 符号模型检测方法不仅不依赖于某个具体的初始化状态进行验证，而且模块化建模复用性高，大大简化了建模的复杂性。符号模型检测的高效性可以有效避免验证过程中状态爆炸问题。

1.2 国内外研究现状

自主移动机器人空间探索 [5][6] 是机器人领域的一个研究热点，机器人之间协作完成空间探索任务的核心是机器人移动算法 [5][7][8][9] 的设定。机器人移动算法决定自主移动机器人的移动行为，从而实现对空间的探索。目前离散空间模型中主要研究机器人探索问题两种探索方式：探索终止 [10][11][12] 和永恒探索 [13]。而大多数研究者都将研究重点放在永恒探索上。

类似于网络拓扑结构，离散空间模型的结构也有多种，例如树形空间、环形空间、星型空间。文献 [12] 描述树形空间位置结点数为 n 时，需要 $\Omega(n)$ 个机器人并且树的深度最大为 3 时，才可能实现永恒探索。文献 [11] 在环形空间位置结点数是 n 的情况下，需要机器人数 $k=\iota(\log n)$ 并且机器人数 k 和环形空间位置结点数 n 互质时，才可能满足永恒探索性质。文献 [9] 提出满足永恒探索性质的最小移动算法和最大移动算法，最小移动算法是指环形空间中，机器人数最少为 k=3，空间位置结点数至少为 n=10，且 n 与 k 的数量关系互质，并给出了证明当机器人

数为 k=3 时，空间位置结点数 n 小于 10 时，没有移动协议满足永恒探索性质。最大移动算法是环形空间位置结点数为 n，机器人数 k=n-5 且且 n 与 k 的数量关系互质时，才可能满足永恒探索形式。在文献 [9] 中也同时给出了最小移动算法和最大移动算法的实例，并利用手动推演的方式，进行了证明。文献 [14][15] 手动推演证明了网格空间的探索性质。

文献 [17] 和 [18] 探讨了机器人移动算法自动化验证的可能性。文献 [17] 使用 LUSTRE[19] 对 3×3 网格空间永恒探索性质进行了描述，通过穷尽搜索的方式验证 3 个机器人在半同步调度策略下，没有搜索到满足永恒探索性质的移动算法。文献 [18] 中描述环形空间大小为 n，机器人数为 k 时的永恒探索算法，并自动生成在半同步调度策略下的移动算法。

在采用形式化自动化验证方面，文献 [20] 首先使用形式化模型描述离散空间中机器人移动算法，并将形式化模型转化成 DiVinE 模型验证器 [21] 和 ITS 工具 [22] 支持的 DVE 语言，实现了移动算法的自动化验证。文献 [23] 则使用基于重写逻辑 Maude 形式化语言对移动算法进行建模，并使用 LTL 描述移动算法的永恒探索性和互排斥性，通过 Maude LTL 模型验证器 [24][25][26] 实现模型的验证。

上述对永恒探索算法形式化建模、验证和分析工作，相对于之前手动验证机器人移动算法效率上得到可巨大的提升。但是受限于形式化工具自身的形式，通过人为设定一个具体的初始状态进行验证，然而机器人移动算法的机器人初始位置状态是任意的。尽管从理论上是可以穷举所有机器人可能的初始位置状态对机器人移动算法进行一一验证，然而然而在实际操作中不仅效率低下并容易发生遗漏而导致验证不完整。Aminof 等人提出一种参数化的模型检测方法用于聚集系统 (rendezvous systems)[27]。然而由于参数化模型检测的不可判定性 [28]，, 该方法只能用于某类特殊模型的验证。通常需要通过抽象 (abstraction) 或者归纳 (induction) 等手段对模型进行适当的转化使其模型检测问题变的可判定 [29][30]。

nuXmv[31] 是一种符号模型验证工具，具有高效的性能，可以对有限状态系统和无限状态系统进行建模与分析。利用 nuXmv 工具分别对机器人自主空间探索

算法进行建模并利用 LTL 公式对机器人的移动行为进行定义。利用 nuXmv 提供的基于 BDD 方法 [32] 及 SMT 方法 [32] 实现了协议的符号模型检测, 验证移动算法是否满足永恒探索的性质。

1.3 研究内容和方法

本文的主要研究内容是机器人巡逻问题中自主移动机器人永恒探索算法的形式化方法的改进。机器人移动算法是自主移动机器人协作完成永恒探索任务的核心。空间中所有机器人执行相同移动算法，在没有任何外部控制器的情况下，通过视觉传感器获取的其他机器人的位置信息，与机器人移动算法进行匹配，做出移动决策。机器人不断反复获取空间位置信息、匹配移动算法、做出移动操作，使得每个机器人对离散空间模型中每个位置结点进行反复的访问，称为永恒探索。满足永恒探索的移动算法称为永恒探索算法。本文针对离散空间中机器人在不同的调度模型下移动过程的详细形式化描述和分析。

本文采用符号模型检测的方法实现对机器人移动算法建模，验证移动算法是否是永恒探索算法。nuXmv 是新的符号模型检测工具，能胜任离散空间移动算法的建模验证与分析。对机器人、离散空间、移动算法分别进行模块化建模，模块化代码的可移植性很好，简单清晰，十分方便。完全同步调度模型、半同步调度模型、完全异步调度模型中，模块化代码可以复用，使得复杂的编码过程变得容易，只需针对不同调度模型做出一些对应调整。使用 LTL 公式分别描述永恒探索算法所需满足的非冲撞性、非互换性、非终止性，并可以通过 nuXmv 分别进行验证，在不满足任何一条性质时，nuXmv 可以给出反例状态路径，便于原因分析。为了进一步提升移动算法验证的效率，nuXmv 验证代码通过代码生成器进行生成，只需录入移动算法序列、机器人数等相关信息。

1.4 论文结构

全文一共分为五个章节:

第一章介绍了文章的研究背景，简单描述了自主移动机器人永恒探索算法，并引出了形式化符号模型检测工具 nuXmv，说明使用 nuXmv 工具实现永恒探索算法建模的优点。国内外相关课题的研究现状，以及本文使用 nuXmv 对永恒探索算法的形式化建模验证的方法。

第二章对 nuXmv 工具进行详细介绍，包括符号化建模适用范围，有限状态机的定义，模块化建模，CTL 和 LTL 模型检测命令。通过具体的代码示例描述 nuXmv 的建模和验证过程。

第三章具体介绍探索空间的可能的拓扑结构，详细描述了机器人，探索空间、移动算法的具体内容给出了确切的定义。具体分析自主移动机器人永恒探索算法，包括机器人的移动阶段，机器人移动算法匹配过程，永恒探索的性质。提出三种移动机器人的调度模型，并分别调度模型进行了详细介绍。

第四章提出使用 nuXmv 符号模型检测方法，对自主移动机器人永恒探索算法进行建模验证和分析。以环形拓扑空间的最小移动算法为例，构建最小移动算法在三种不同的调度模型下的模型，并采用 nuXmv 中的基于 BDD 算法和 SMT 的验证指令，进行了形式化的验证，对实验结果也进行详细的分析。

第五章对全文工作做了总结。对实验方法和验证结果进行了分析，总结本文的主要贡献。讨论了 nuXmv 符号模型检测在空间探索协议验证的 BDD 方法和 SMT 方法的选择性，指出了本文工作中的一些不足点，对未来自主机器人领域的研究内容进行了展望。

第二章　nuXmv 介绍

muXmv 是一种新的符号模型验证工具，主要用于有限和无限状态同步变换系统的验证与分析。nuXmv 是在经典符号模型验证工具 NuSMV 的基础上创建的，不仅完全继承了 NuSMV 的功能，而且还从两个方面进行了拓展：1、对于有限状态系统，利用最新的 SAT-based 算法补充 NuSMV 的基本验证技术；2、对于无限状态系统，在 NuSMV 的基础上补充了两种新的数据类型，而且还提供了先进的基于 SMT 的形式化验证技术。除了功能上的拓展，nuXmv 还从验证性能上得到了进一步提升，nuXmv 作为一种验证工具被广泛应用于工业项目当中，例如需求分析、分布式系统模型检测、安全评估和软件模型验证。

本章节从应用角度对 nuXmv 的基本数据类型、模块和验证方法进行了说明，并使用具体的例子对其使用方法进行了详细的介绍。为后续的自主移动机器人移动算法建模做铺垫。

2.1　基本数据类型

在 nuXmv 中基本数据类型有 7 种，分别是布尔类型 (Boolean)、枚举类型 (Enumeration)、字 (Word)、整数 (Integer)、实数 (Reals)、数组 (Array) 和集合 (Set)。下面对每种类型的使用做一下说明。

布尔类型　类似于 C++，Java 等高级语言，nuXmv 中的布尔类型数据只有真 (TRUE) 和假 (FALSE) 两种取值。

枚举类型 同样类似于 C++，Java 等高级语言，枚举类型指定了该枚举的所有可能取值。举例说明：

```
Example 1 : {stopped, running, waiting, finished}
Example 2 : {2, 4, -2, 0}
Example 2 : {FAIL, 1, 3, 7, OK}
...
```

枚举数据类型的声明时，数据元素不能重复，数据元素的顺序没有严格要求。枚举当中元素的数据类型可以是相同的，如 Example 1 中使用都是符号常量，Example 2 中都是整数。也可以有混合类型的，如 Example 2 中个，既有符号常量也有整数。

字 数据类型字 (Word) 分为无符号字 (unsigned word) 和有符号字 (signed word), 在 nuXmv 中分别定义语法如下：

```
unsigned word : unsigned word[n]
  signed word : signed word[n]
```

无符号字使用关键词 unsigned word 声明，有符号字使用关键词 signed word。语法中 n 表示字的位宽（bit），例如 $(unsigned \quad word[3])$ 代表 3 位（bit）的无符号字，$(signed \quad word[7])$ 代表 7 位（bit）的有符号字。

整数 整数有正整数和负整数。首先应该注意的是在某些模型检查引擎和算法中不允许使用整数。整型的取值范围 $-2^{32}+1$ 到 $2^{32}-1$。

实数 实数类型的域是有理数。

数组 数组使用索引的上界和下界声明。举例说明，数组 1..8 代表数组中有从 1 到 8 的 8 个整型元素。数组类型与集合类型不兼容，即数组元素不能是集合类型。

集合 集合类型是用于标识一组值的表达式，对于集合只能使用范围常数 (range constant) 和联合运算符 (union)。

2.2 表达式

在 nuXmv 中，所有的表达式都有类型的约束和操作数的类型约束，假若表达式违反了类型的约束会被系统识别为错误的表达式。下面介绍一个 nuXmv 中表达式的一些性质和分类。

2.2.1 隐式转换

某些表达式操作数可以进行隐式类型转换。可以从一个类型转换成该类型的集合类型。例如，整数类型（integer）可以转换为整数集合类型（integer set）。在老版本的 NuSMV 中，支持整数类型（integer）隐式转换为布尔类型（boolean），但自从 2.5.1 版本之后不在支持整数类型（integer）隐式转换为布尔类型（boolean），同样 nuXmv 也不支持，只能通过必须使用显式转换运算符才能进行强制转换。

2.2.2 基本表达式

nuXmv 中包含大量的基本表达式，包括最常见的逻辑运算符与、或、非等等，也包含四则运算符加、减、乘、除等等以及其他的各类操作符，下面对 nuXmv 中几种比较特殊的基本表达式进行介绍，其他的请参见附录 [基本表达式]。

Case 表达式 Case 表达式类似于高级语言 C++，Java 中的条件语言if-else if-else。基本语法如下:

```
case_expr :: case case_body esac

case_body ::
          basic_expr : basic_expr ;
        | case_body basic_expr : basic_expr ;
```

语法中case_expr 的值是case_body 中第一个左边条件为真，冒号：右边的值。下面给出一个例子，做一下简单的解释：

```
case
    left_expression_1 : right_expression_1 ;
    left_expression_2 : right_expression_2 ;
```

```
4         ...
5         left_expression_N : right_expression_N ;
6     esac
```

关键词case 和esac 之间是case_body，case_body 中所有表达式的左边条件通用表达式可以写作left_expression_[k]，$kin1, 2, 3..., N$。对于所有的left_expression_[k] 都必须是布尔类型。某个数值 j 满足 $jin\{1, 2, 3..., N\}$，对于所有从 1 到 j-1 的 i 使得left_expression_[i] 都为 FALSE，且left_expression_[j] 为 TRUE 时，case 表达式的取值为left_expression_[j] 对应的右边值right_expression_[j]。在 nuXmv 中 Case 表达式必须有可能性的值，当所有的left_expression_[k] 都不满足时，nuXmv 会抛出异常。所以会在 Case 表达式的最后加上一个永真条件表达式。如下：

```
1     case
2         cond1 : expr1;
3         cond2 : expr2;
4         cond3 : expr3;
5         ...
6         TRUE : exprN; -- otherwise
7     esac
```

Next 表达式 在 nuXmv 中使用 Next 表达式表示某个变量下一个状态时的值。其语法如下：

```
1     next_expr :: next ( basic_expr )
```

举个例子说明一下，变量 V 是一个状态变量，next(V) 表示变量 V 在下一个状态时的值。

取模操作 nuXmv 中使用关键字 mod 进行取模运算，同C/C++ 程序语言中的取模的计算方法是一样的。即当 a/b 是有意义的，必须使得表达式 $(a/b) * b + (a \ mod \ b) = a$ 成立。下面给 nuXmv 取模运算的一组实例：

```
1     7/5 = 1      7 mod 5 = 2
2     -7/5 = -1    -7 mod 5 = -2
3     7/-5=-1      7 mod -5 = 2
4     -7/-5=1      -7 mod -5 = -2
```

2.3 有限状态机

有限状态机中可以定义变量、状态迁移关系、公平性约束等等。变量包括状态变量(state variables)、输入变量(input variables) 和固定变量(frozen variables)，这些变量可能随着状态的迁移，而有不同的取值。状态迁移关系表示的某个状态在不同的条件输入下，可能到达的状态。公平性约束描述状态机执行的有效路径约束。本文中所提到的约束(constraints) 是用于约束有限状态机的行为，而规格(specifications) 是用来描述有限状态机中被验证的性质。

2.3.1 变量

状态变量

状态变量使用关键词 VAR 进行声明，状态变量声明语法如下：

```
var_declaration :: VAR var_list

var_list :: identifier : type_specifier ;
          | var_list identifier : type_specifier ;
```

语法中var_list 表示通过 VAR 关键字可以声明一组状态变量。var_list 可以是一个变量标识符identifier，紧跟着冒号后面是改变量的类型type_specifier。var_list identifier : type_specifier 使用递归的方式表示声明一组状态变量。下面使用一个简单的例子进行简要说明：

```
VAR
    a : boolean;
    b : 0..10;
    c : {ready,busy};
```

例子中变量a 的类型是布尔类型。变量b 是一个限定值域范围的正数类型，其取值范围是0 到10 之间的11 个正数。变量c 是一个枚举类型，其取值范围是枚举中两个元素ready 和busy。

输入变量

输入变量使用关键词 IVAR 进行声明，常常被用于有限状态机中状态迁移关系中。输入变量的声明语法如下：

```
ivar_declaration :: IVAR simple_var_list

simple_var_list ::
    identifier : simple_type_specifier ;
    | simple_var_list identifier : simple_type_specifier ;
```

输入变量使用只能简单的数据类型simple_type_specifier 进行，简单的数据类型包含基础的布尔类型、整数类型、实数类型等等。identifier 表示输入变量的标识符。语法中simple_var_list 表示同状态变量声明一样，输入变量声明时也可以声明一组。

在nuXmv 中，输入变量比状态变量在使用上受到更多的限制，输入变量不能是一个模块实例。在模块和规格中同样也受到很大限制。下面给出一组具体的例子进行说明：

```
IVAR
  i : boolean;
ASSIGN
  init(i) := TRUE;
  next(i) := FALSE;
```

所有的输入变量赋值都是不允许的，例子中使用关键词IVAR 声明了一个布尔类型的输入变量i，后续使用赋值关键词ASSIGN，定义输入变量i 的初始化值为TRUE。这在代码执行时，nuXmv 会返回不合法的初始化值的错误提示。

```
IVAR i : boolean;
 VAR s : boolean;
INIT i = s
```

同样在 INIT 约束中，包含输入变量 i 是不合法的。

```
  IVAR i : boolean;
   VAR s : boolean;
  TRANS i -> s; – 合法
  TRANS next(i -> s); - 不合法
```

下一个时刻状态表达式next 中，不能包含输入型变量i。而输入变量在TRANS约束中，可以包含输入变量。

```
IVAR
   i : boolean;
VAR
   s : boolean;
SPEC     AF (i -> s); – 不合法
LTLSPEC F (X i -> s); – 合法
INVARSPEC (i -> s); – 合法
```

在一些规格说明语句中不能包含输入变量，这些规格说明是CTLSPEC，SPEC，COMPUTE 和PSLSPEC。而对于LTLSPEC，INVARSPEC 中包含输入变量是合法的。

固定变量

固定变量的值在状态机状态迁移过程中保持不变。固定变量的声明语法如下：

```
frozenvar_declaration ::
        FROZENVAR simple_var_list

simple_var_list ::
         identifier : simple_type_specifier ;
         | simple_var_list identifier : simple_type_specifier ;
```

固定变量使用关键词 FROZENVAR 进行声明，simple_var_list 使用递归的表达式说明固定变量可以初始化一组。下面给出一个简单例子说明固定变量的使用场景：

```
FROZENVAR
     a : boolean;
     b : boolean;
     c : boolean;

VAR d : boolean;

FROZENVAR e : boolean;

ASSIGN
     init(a) := d; -- 合法
     next(b) := d; -- 不合法
     c := d;       -- 不合法
     e := a;       -- 不合法
```

在例子中，声明了三个固定变量 a、b、c，后面定义一个状态变量 d，紧跟着又定义了一个固定变量 e。可以使用 init 对固定变量进行初始化，但不能使用 next

定义固定变量的下一个状态时刻的值。固定变量在初始化之后也不能使用赋值语句进行赋值操作。

2.3.2 常用约束

nuXmv 中有很多种约束，这些约束可以定义模块中变量赋值、状态迁移关系、状态有效路径约束等等。比较常用的约束有 INIT、ASSIGN、FAINESS 等等，下面我们对常用约束做一些简单的介绍。

INIT 约束

INIT 约束用于定义变量的初始值，其具体语法如下：

```
init_constrain :: INIT simple_expr [;]
```

在 INIT 约束中不能包含下一个状态值 next 的操作，表达式simple_expr 必须是布尔类型。nuXmv 支持同时声明多个初始化约束。

ASSIGN 约束

ASSIGN 约束在 nuXmv 中很常见，用于给变量定义初始值，或者状态迁移时定义变量当前值和下一个状态时的值之间的等价关系。ASSIGN 约束的语法如下：

```
assign_constraint ::
    ASSIGN assign_list

assign_list ::
    assign ;
    | assign_list assign ;

assign ::
    complex_identifier := simple_expr
    | init ( complex_identifier ) := simple_expr
    | next ( complex_identifier ) := next_expr
```

语法中assign_list 使用的是递归的表达方式，说明一个关键词 ASSIGN 可以定义多个 ASSIGN 约束表达式。assign 中不仅支持简单的表达式simple_expr，定义基本的等价关系，也可以使用 init 关键词初始化变量值，同时也可以使用 next 表达式，表示某些变量当前状态时的值和下一个状态时的值之间的等价关系。

FAIRNESS 约束

在异步模型的验证过程中，常常会有一种情况是几个模型实例，它们执行的时间片是随机分配的，可能会出现一种情况就是某些模型实例的所分得时间片较少或者几乎没有。在某些应用场景中，这是不符合模型本质特性的。为了几个实例模型能够公平的获取执行时间片，nuXmv 中使用关键字 FAIRNESS 对模块进行公平性的定义。公平性的定义很简单，只需要在模块中加上一段代码，如下：

```
FAIRNESS
   running
```

具体的公平性的使用，将在 nuXmv 的模块定义中进行详细说明。

2.3.3 模块的声明

nuXmv 中的模块类似于C++，Java 等面向对象思想。在 nuXmv 中使用模块化的建模方式，使得代码简洁、易于阅读。模块化建模的可移植性更强，减少不必要的重复代码。nuXmv 中模块的声明语法如下：

```
module :: MODULE identifier [( module_parameters )] [module_body]

module_parameters ::
        | module_parameters , identifier

module_body ::
          module_element
        | module_body module_element

module_element ::
          var_declaration
        | ivar_declaration
        | frozenvar_declaration
        | define_declaration
        | constants_declaration
        | assign_constraint
        | trans_constraint
        | init_constraint
        | invar_constraint
        | fairness_constraint
        | ctl_specification
        | invar_specification
        | ltl_specification
        | pslspec_specification
        | compute_specification
        | isa_declaration
        | pred_declaration
```

```
28        | mirror_declaration
```

使用关键词MODULE 声明一个模块，MODULE 后面的标识符identifier 是模块的名称。模块包含两个主要的部分模块的传入参数module_parameters 和模块的主体部分module_body。模块的传入参数可以是一个或者多个，在给模块定义参数时，直接写参数的名称。模块的主体部分种可以定义很多种声明、约束和规格。声明包括状态变量声明var_declaration、输入变量声明ivar_declaration 和固定变量声明frozenvar_declaration 等等。约束包括分配约束assign_constraint、初始化init_constraint 约束和公平性约束fairness_constraint。规格中包括线性时态规格ltl_specification，这也是后续验证过程中使用的验证性质描述方式。nuXmv 支持同步系统和异步系统的建模，在同步系统和异步系统建模时，模块的定义是相同的，只是在一些性质的描述方面不同，下面使用二进制计数器（Binary Counter）和互斥系统（Mutual Exclusion）两个 nuXmv 建模的例子，分别介绍一下同步系统和异步系统的详细过程。

同步系统 -二进制计数器 下面程序描述一个二进制计数器，其中计数器模块counter_cell 是一个可重用的模型。在主模块 main 中实例化了三个二进制计数器。

```
1  MODULE counter_cell(carry_in)
2      VAR
3          value : boolean;
4      ASSIGN
5          init(value) := FALSE;
6          next(value) := value xor carry_in;
7      DEFINE
8          carry_out := value & carry_in;
9
10 MODULE main
11     VAR
12         bit0 : counter_cell(TRUE);
13         bit1 : counter_cell(bit0.carry_out);
14         bit2 : counter_cell(bit1.carry_out);
```

程序中的主模块main 中，重复三次实例化计数器模块，分别用符号bit0、bit1、bit2 表示。计数器模块counter_cell 有一个传入参数carry_in。在实例bit0

中，传入参数为TRUE。在实例bit1 中，传入参数为是实例bit0 的输出参数carry_out。同样的，实例bit2 的输入参数是bit1 的输出参数carry_out。在计数器模块counter_cell 中，使用关键词VAR 声明了一个布尔类型的状态变量value。使用ASSIGN 约束初始化了状态变量value 的值为FASLE，和下一个状态时value 的值，下一个状态时value 的值等于当前value 和输入参数carry_in 异或值value xor carry_in。输出变量carray_out 的值是当前状态下value 和输入变量carry_in 的与值。

异步系统 -互斥系统 下面模型中描述的是一个互斥系统，这是一个异步系统建模的例子。在主模块 main 中声明公共的布尔类型状态变量semaphore 和两个两个模块user 的实例对象proc1 和proc2。并将状态变量semaphore 作为传入参数传入proc1 和proc2 两个实例对象中。实例对象proc1 和proc2 在声明时使用了关键词process，表示proc1 和proc2 是两个独立的异步进程。状态变量semaphore 的初始值为FALSE。

```
MODULE main
    VAR
        semaphore   : boolean;
        proc1       : process user(semaphore);
        proc2       : process user(semaphore);
    ASSIGN
        init(semaphore) := FALSE;

MODULE user(semaphore)
    VAR
        state : {idle, entering, critical, exiting};
    ASSIGN
        init(state) := idle;
        next(state) :=
            case
                state = idle                    : {idle, entering};
                state = entering & !semaphore : critical;
                state = critical                : {critical, exiting};
                state = exiting                 : idle;
                TRUE : state;
            esac;
        next(semaphore) :=
            case
                state = entering : TRUE;
                state = exiting  : FALSE;
                TRUE             : semaphore;
            esac;
```

```
28    FAIRNESS
29        running
```

用户模块user 的定义中，有传入参数semaphore。用户模块主体内部声明了状态变量state，state 的取值有四种分别是idle、entering、critical 和exiting。状态变量state 的初始值为idle，下一个状态时state 的值分几种情况：当状态变量state 是idle 时，下一个状态时变量state 取值为idle 或者entering，并且是随机分配的；当状态变量state 是entering，并且传入参数semaphore 是为FALSE 时，下一个状态时变量state 取值为critical；当状态变量state 是critical 时，下一个状态时变量state 取值为critical 或者exiting，也是随机分配的；当状态变量state 是exiting 时，下一个状态时变量state 取值为idle；最后的TRUE ：state 条件语句，表示在不满足上述几种情况时，一个状态时变量state 取值为当前状态时state 的值。传入参数semaphore 的下一个状态时的值也使用next(semaphore) 表示：当变量state 为entering 时，下一个状态时semaphore 的值为TRUE；当变量state 为exiting 时，下一个状态时semaphore 的值为FALSE；TRUE:semaphore 表示不满足上述两种情况时，下一个状态时semaphore 的值等于当前的semaphore 的值。最后，在用户模块user 中使用关键词FAIRNESS 声明了公平性，表示在异步模型中，用户模块user 的所有实例化进程是公平的执行。

2.4 交互式会话验证

交互式会话验证是指nuXmv 模型验证的一种模式，在交互式会话验证过程中，系统的每一步操作都会停止，显示可能出现的状态值，用户可以根据自己的判断选择其中一个，nuXmv 根据用户输入完成下一步验证。当用户想验证模型中某些特殊性质时，交互式会话验证十分有用。下面使用一个例子，简单介绍一下交互式会话验证的过程。

```
1  MODULE main
2      VAR
3          request : boolean;
4            state : {ready,busy};
5      ASSIGN
```

```
6       init(state) := ready;
7       next(state) :=
8           case
9               state = ready & request : busy;
10              TRUE                    : {ready,busy};
11          esac;
```

上面模型中主模块声明了两个状态变量request 和state，其中变量state 的初始值为ready，而布尔类型变量request 的初始值并未定义，表示变量request 的值为TRUE 或者FALSE 中的任意一个。变量state 在下一个状态时的值有两种情况：当state 为ready，并且request 为TRUE 时，下一个状态时state 的值为ready；其他情况下，下一个状态时state 的值为ready 和busy 中的任意一个。

下面是上述例子在交互式会话验证时，执行的过程:

```
1  system prompt>nuXmv -int demo.smv
2  nuXmv > go
3  nuXmv > pick_state -r
4  nuXmv > print_current_state -v
5  Current state is 1.1
6  request = FALSE
7  state = ready
8  nuXmv > simulate -r -k 3
9  ******** Simulation Starting From State 1.1 ********
10 nuXmv > show_traces -t
11 There is 1 trace currently available.
12 nuXmv > show_traces -v
13     <!-- ##################### Trace number: 1 ###################
14 Trace Description: Simulation Trace
15 Trace Type: Simulation
16   -> State: 1.1 <-
17     request = FALSE
18     state = ready
19   -> State: 1.2 <-
20     request = TRUE
21     state = ready
22   -> State: 1.3 <-
23     request = TRUE
24     state = busy
25   -> State: 1.4 <-
26     request = FALSE
27     state = ready
28 nuXmv >
```

指令nuXmv -int demo.smv 中的-int 表示nuXmv 的验证模式为交互式会话验证，demo.smv 是nuXmv 的代码文件，内容是模型的构建代码。指令go 表示交互式会话验证开始执行。指令pick_state -r 表示选中初始状态为验证的起始状态。

指令simulate -r -k 3 表示从选择的起始状态的下一步状态开始，构建三步的状态路径。指令show_traces -v 查看状态路径，可以在示例操作的最后看见包括初始状态在内的四个状态，每个状态都描述着变量request 和state 在当前状态下的值。根据这些状态值可以准确分析模型的状态路径。

2.5 线性时态逻辑规格介绍

通过上面的介绍，了解了nuXmv 的建模方式和适用范围。在实际适用过程中，使用nuXmv 的建模，并验证该模型是否满足某些性质。那么此时就需要使用规格来描述这些需要验证的性质。nuXmv 支持的规格有很多比如计算树逻辑(CTL，Computation Tree Logic) 规格、线性时态逻辑(LTL，Linear Temporal Logic) 规格、属性说明语言PSL(Property Specification Language) 规格等等。由于本文中使用的是线性时态逻辑规格描述机描述永恒探索算法的性质，所以此处只介绍线性时态逻辑规格。

2.5.1 线性时态逻辑规格语法

nuXmv 在模块定义中使用关键词LTLSPEC 定义线性时态逻辑规格，线性时态逻辑规格的语法如下：

```
ltl_specification ::
    LTLSPEC ltl_expr [;]

ltl_expr ::
    next_expr                  -- a next boolean expression
    | ( ltl_expr )
    | ! ltl_expr               -- logical not
    | ltl_expr & ltl_expr      -- logical and
    | ltl_expr | ltl_expr      -- logical or
    | ltl_expr xor ltl_expr    -- logical exclusive or
    | ltl_expr xnor ltl_expr   -- logical NOT exclusive or
    | ltl_expr -> ltl_expr     -- logical implies
    | ltl_expr <-> ltl_expr    -- logical equivalence
    -- FUTURE
    | X ltl_expr               -- next state
    | G ltl_expr               -- globally
    | G bound ltl_expr         -- bounded globally
    | F ltl_expr               -- finally
    | F bound ltl_expr         -- bounded finally
    | ltl_expr U ltl_expr      -- until
```

```
    | ltl_expr V ltl_expr        -- releases
    -- PAST
    | Y ltl_expr                 -- previous state
    | Z ltl_expr                 -- not previous state not
    | H ltl_expr                 -- historically
    | H bound ltl_expr           -- bounded historically
    | O ltl_expr                 -- once
    | O bound ltl_expr           -- bounded once
    | ltl_expr S ltl_expr        -- since
    | ltl_expr T ltl_expr        -- triggered

bound :: [ integer_number , integer_number ]
```

线性时态逻辑规格可以包含非(!)，与(&)，或()|，异或(xor) 等逻辑运算，也可以包含线性时态逻辑中的下一个状态(X)，某一个状态开始后续所有状态下都为真(G)，未来某一个状态下为真(F) 等等。

下面还是使用互斥系统模型为例，简要介绍一下线性时态逻辑规格在模型中的定义：

```
MODULE main
    VAR
        semaphore : boolean;
        proc1 : process user(semaphore);
        proc2 : process user(semaphore);
    ASSIGN
        init(semaphore) := FALSE;
    LTLSPEC G ! (proc1.state = critical & proc2.state = critical); --规格1
    LTLSPEC G (proc1.state = entering -> F proc1.state = critical); --规格2

MODULE user(semaphore)
    VAR
        ...
    ASSIGN
        ...
    FAIRNESS
        running
```

模型的详细内容前面已经介绍过了，这里重点解释一下主模块中新增的两条线性时态逻辑规格。规格 1 中描述的在未来没有一个状态时，满足进程proc1的变量state 和进程proc1 的变量state 的值同时为critical。规格 2 中描述的是当在某个状态进程proc1 变量state 的值为entering 时，在该状态之后必定有个状态满足proc1 变量state 的值为critical。

2.6 BDD 与 SMT 验证方法

nuXmv 支持 BDD(binary decision diagram)[3] 和 SMT(Satisfiability modulo theories)[4] 验证方式。BDD 是一个有向无环图，使用二叉树的形式表示布尔逻辑函数。这种二叉分支逻辑判定的算法，可以大大减少算法的复杂度，提升判定结果集搜索效率。SMT 求解方式是在 SAT 算法和一些优秀的求解框架的基础上发展起来的，目前 SMT 可满足性验证技术已经相当成熟，SMT 求解器拥有较强的验证能力，并广泛应用于计算机理论、嵌入式系统、生物科学等相关领域，尤其是在大规模系统的验证中，SMT 展现出十分高效的运算能力。nuXmv 是一个很方便的符号模型验证工具，在集成大量优秀的形式化验证算法和求解器的同时，也对这些验证算法和求解器进行了完美的封装，提供了对应的操作和执行命令。使用者只需要按照使用手册中的说明，选择自己需要的验证方法，执行验证命令进行求解。下面分别介绍 nuXmv 中的 BDD 和 SMT 的验证方法。

BDD 验证过程 如图2.1描述的是 nuXmv 中基于 BDD 验证 LTL 规格的命令执行流程。

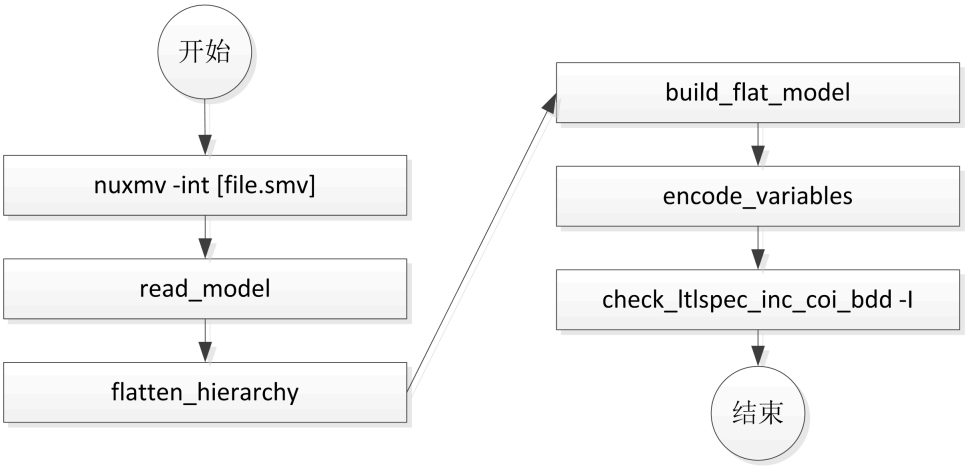

图 2.1: 基于 BDD 的验证方法命令流程

首先使用命令nuxmv -int [file.smv] 开启交互式验证模式，file.smv 是 nuXmv 建模文件。命令read_model 加载 nuXmv 的文件file.smv 到 nuXmv 的解析器中。命

令flatten_hierarchy 通知 nuXmv 将建模代码中模块和过程进行偏平层次实例化。命令build_flat_model 是将偏平层次实例装配到状态机中。命令encode_variables 是构建 BDD 数据结构的验证模型。命令check_ltlspec_inc_coi_bdd -I 是在 BDD 验证模型的基础上进行基于 COI 算法的 LTL 规格验证。

SMT 验证过程 如图2.2描述的是 nuXmv 中基于 SMT 验证 LTL 规格的命令执行流程。

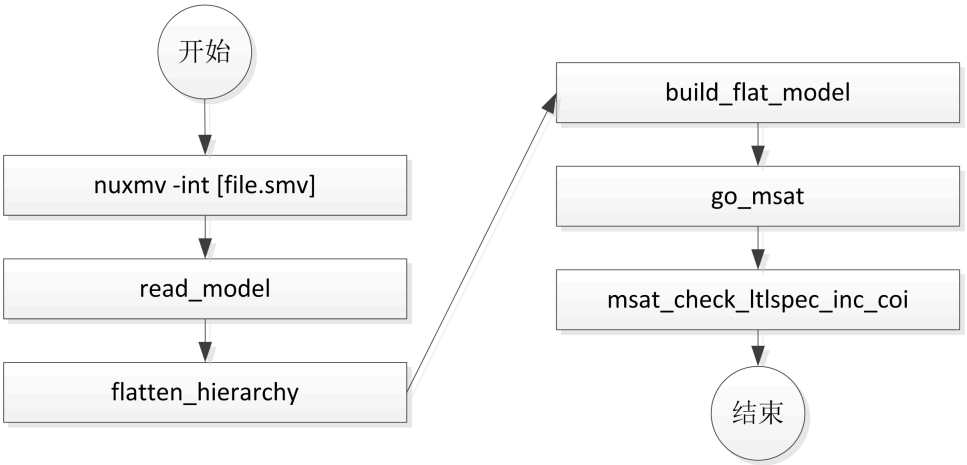

图 2.2: 基于 SMT 的验证方法命令流程

前面的执行命令与 BDD 验证过程中的相同，go_msat 是使用基于 SMT 验证方式初始化验证系统。命令msat_check_ltlspec_inc_coi 是使用基于 SMT 的 COI 算法的 LTL 规格验证。

2.7 本章小结

本章介绍了符号模型验证器 nuXmv 的基础数据类型，通过基础数据类型和模块定义可以实现有限状态机的建模。nuXmv 支持多种时序逻辑语言，本章节主要是讲解通过其中的 LTL 定义有限状态机中预验证性质的可满足性验证。并以异步信号系统为例，详细的介绍了模型的构建以及模块中状态迁移，重点说明 LTL 在模块中声明。此外，还阐述了 nuXmv 支持的各种 LTL 的验证方法，对于验证性

质的正确性,BDD 方法的效率相对较高, 而对于不被满足的性质, 基于 SMT 验证方法可以更快的找到不满足的反例。

第三章　机器人永恒探索算法

未知空间永恒探索的研究中，空间模型是离散空间模型，离散空间可以将空间划分为有限的空间位置结点和有限的移动路径。空间模型离散空间模型使用无向连通图来表示，图的结点表示离散空间的位置，图的边表示机器人可以通过的路径。在离散空间模型中，在同一时刻每个空间位置结点至多只能有一个机器人。

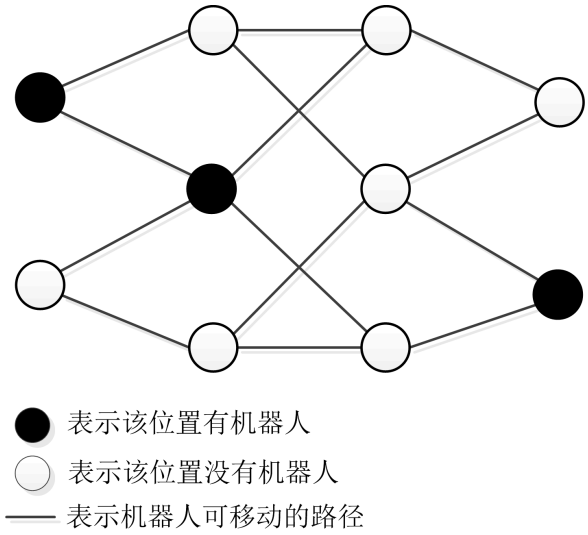

图 3.1: 无向连通图表示探索空间

如图3.1是一个简单的离散空间模型，黑色结点表示该空间位置在此刻有一个机器人，白色结点说明该空间位置上没有机器人，空间中有三个机器人，分别位于三个黑色的空间位置结点上。离散空间的结构有很多，比如总线拓扑结构、星型拓扑结构、环形拓扑结构、树形拓扑结构等等。

3.1 机器人移动

在离散空间模型中，机器人可以沿着某条边从一个位置结点到达相邻的位置结点，如图3.2 所示：

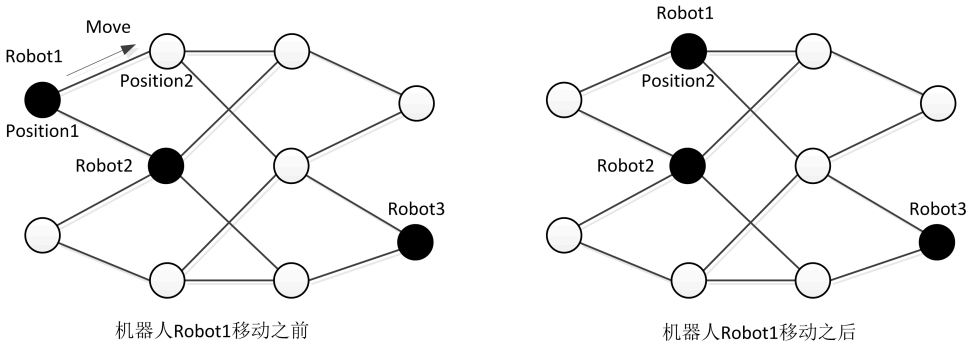

图 3.2: 机器人在离散空间模型中的移动

在位置结点 Position1 上的机器人 Robot1 沿着边移动，到达相邻的位置结点 Position2 上。相邻位置结点的定义是，两个位置结点通过一条边相连接，它们互为相邻位置结点，例如图中的位置结点 Position1 和位置结点 Position2 就是互为相邻位置结点。离散空间模型中的机器人移动两个基本特点：第一，有通往其他位置结点的边；第二，机器人每次移动之后，只能通过一条边，到达另外一个相邻位置结点之后，一个移动动作就完成，到达相邻位置结点之后，会重新开始一个移动动作。

自主移动机器人需要根据环境信息，匹配自身的移动算法做出移动决策，完成移动。如图3.3机器人的移动动作可以分为三个阶段：观察 (look)、计算 (compute) 和移动 (move)。机器人在观察阶段，通过装备的视觉传感器，获取未知空间中其他机器人的位置快照。在计算阶段，通过匹配移动算法，获取机移动决策。在移动阶段，机器人的动力装置按照计算阶段获取移动决策完成移动。

机器人的观察、计算和移动是周期性重复进行的，当机器人完成移动之后，会进入观察阶段，获取周围的环境快照，快照匹配移动算法获取移动决策，在移动阶段完成移动，然后又进入观察阶段。

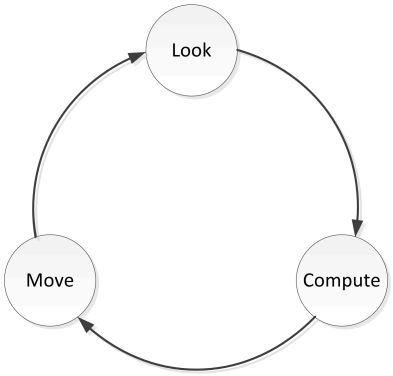

图 3.3: 机器人移动三阶段

3.2 调度策略

在多自主移动机器人协作完成空间探索任务中，机器人调度是指机器人移动的三个阶段是否同步执行。假若有两个机器人 A 和 B，机器人 A 在观察阶段时，机器人 B 也在观察阶段；机器人 A 在计算阶段时，机器人 B 也在计算阶段；机器人 A 在移动阶段时，机器人 B 也在移动阶段，那么称机器人 A 和 B 是同步的，即机器人 A 和 B 的移动动作具备原子性。机器人的调度策略有三种：半同步调度策略(semi-synchronous model，SSYNC)、完全同步调度策略(fully-synchronous model，FSYNC) 和完全异步调度策略 (asynchronous model)

为了方便描述调度策略，首先介绍以下几个概念。使用集合 $Rob = \{k \in N^+ | r_1, r_2, r_3, ..., r_k\}$ 表示未知空间中所有机器人，k 是机器人个数，r_k 表示机器人。使用集合 $Pos = \{n \in N^+ | 1, 2, ..., n\}$ 表示空间所有空间位置结点，n 是空间位置结点个数，使用数字给空间中位置结点都进行了编号 $1, 2, ..., n$。映射关系 $c : \{Rob \to Pos\}$ 表示机器人所在的空间位置。机器人 r 的空间位置定义为 $c(r)$。对于集合 Rob 中的任意两个机器人 $r_i, r_j\,(i \neq j)$, 在同一时刻满足 $c(r_i) \neq c(r_j)$，即同一时刻，空间中任意一个位置结点上至多只有一个机器人。

图3.4中描述的离散空间中有三个机器人 r_1, r_2, r_3, 使用集合表示为 $Rob = \{r_1, r_2, r_3\}$。空间位置结点的集合为 $Pos = \{n \in N^+ | 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$。机器人 r_1 的位置表示为 $c(r_1) = 1$，机器人 r_2 的位置表示为 $c(r_2) = 4$，机器人 r_3 的位

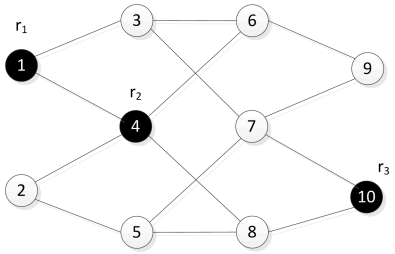

图 3.4: 空间位置结点和机器人符号描述

置表示为 $cr_3 = 10$。

半同步调度策略 半同步调度策略中，只有被调度器选中的机器人，才会执行移动过程。为了方便描述，这里定义调度集合 *Sched*，使用伪算法描述半同步调度策略：

```
SSYNC-SCHEDULE(Rob)
  while
    choose Sched from Rob
    foreach r in Sched{
       synchronous {
          r.look
          r.compute
          r.move
       }
    }
```

伪算法的名称为SSYNC-SCHEDULE 表示半同步调度策略，空间中所有的机器人组成的集合 *Sched* 作为伪算法的参数参数。*choose Sched from Rob* 表示从机器人集合 *Rob* 中选取一个子集 *Sched*，并且子集 *Sched* 非空。foreach r in Sched 表示对于集合 *Sched* 中的每个机器人都同步执行观察、计算和移动过程。等移动完成之后，又进入下一个移动动作。在下一个移动动作开始之前，又重新选择调度子集 *Sched*，while 表示重复上述机器人移动执行过程。这就是多机器人半同步调度策略的过程。

完全同步调度策略 完全同步调度模型是半同步调度模型中一种很特殊的情况，每次移动动作之前选取调度子集 *Sched* 等于 *Rob*，即每次调度的时候，集合 *Rob* 中的所有机器人都被选中。完全同步调度策略的伪算法描述如下：

```
FSYNC-SCHEDULE(Rob)
  while
    foreach r in Rob{
      synchronous {
        r.look
        r.compute
        r.move
      }
    }
```

伪算法的名称为FSYNC-SCHEDULE表示完全同步调度策略，除了 $foreach\, r\, in\, Rob$ 这部分与半同步调度策略的伪算法不一样，其他部分都相同。由于每次选取的调度子集 $Sched$ 等于 Rob，所以这部分伪算法简化了调度过程，直接使用集合 Rob 作为调度集合。

完全异步调度策略 完全异步调度策略中，所有的机器人观察、计算、移动都是异步，没有原子性。类似于计算机系统的多线程，每个机器人的移动都是并行且互相之间没有同步约束，当一个机器人处于观察阶段时，其他机器人可能处于计算阶段或者移动阶段。完全异步调度策略的伪算法如下：

```
ASYNC-SCHEDULE(Rob)
  foreach r in Rob{
    asynchronous {
      while{
        r.look
        r.compute
        r.move
      }
    }
  }
```

伪算法的名称为ASYNC-SCHEDULE，表示完全异步调度策略，关键字异步asynchronous 表示 Rob 中机器人是独立执行移动动作，执行观察、计算和移动的快慢，完全是自己执行速度所决定的。可能出现一种情况当某个机器人已经获取了空间环境快照，还未能执行计算或者移动时，其他机器人却完成了移动，改变了空间环境，这就会出现某些机器人使用过时环境信息，做出移动决策。while 表示在完全异步调度策略中，所有机器人都是不断循环的执行观察、计算和移动。

3.3 环形空间探索算法

未知空间结构有总线结构、星型结构、环形结构、树形结构等等。通常的研究是根据某种空间结构的特点，对空间进行建模与分析。本文使用环形空间为研究对象，详细介绍在环形空间中机器人的视觉快照和移动算法表示方法，以及移动算法匹配过程。最后，介绍一下环形空间最小移动算法。

3.3.1 环形空间

如图3.5是一个典型的环形空间模型，空间位置结点按照顺时针方向依次进行编号 $1, 2, 3..., n$，图中有 n 个空间位置结点。

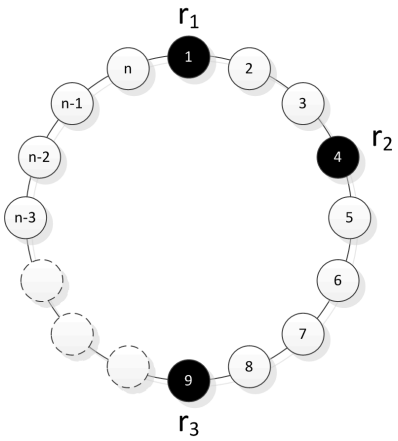

图 3.5: 环形空间

环形空间中位置结点的集合表示为 $Pos = \{1, 2, ..., n\}$，环形空间有自身的一些特点:1、每个空间位置结点都有左右两个相邻的位置结点，例如位置结点 1 的左边 (逆时针) 是位置结点 n, 右边 (顺时针) 是位置结点 2。2、每个空间位置与左右相邻的位置结点，都有各自有条边 (路径) 连通。3、机器人每次移动，只能顺时针或者逆时针进行移动。例如图中机器人 r_1 可以顺时针移动到位置结点 2, 也可以逆时针移动到位置结点 n。环形空间中的位置结点上，也必须是在同一时刻至多只能有一个机器人。

3.3.2 环境快照

机器人可以通过视觉传感器获取环境快照，在环形空间中，可以沿着顺时针或者逆时针方向，获取环境快照。对于机器人而言，只能判断某个空间位置结点上是否有机器人。

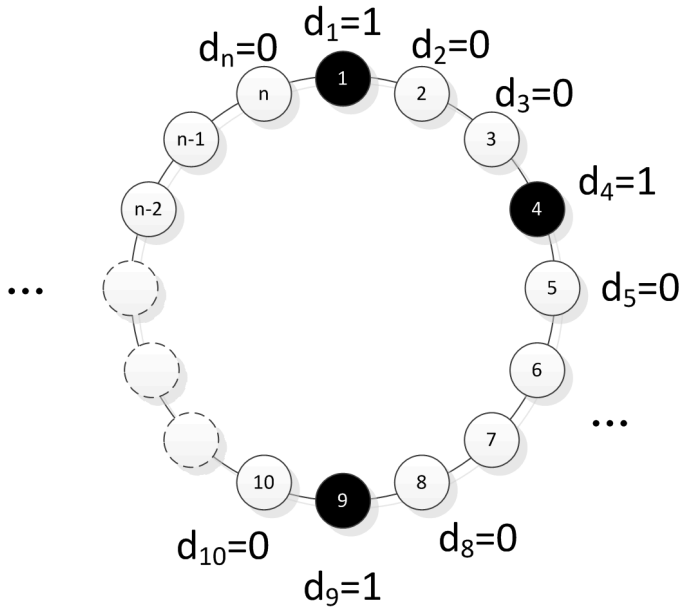

图 3.6: 结点机器人的描述

如图3.6使用符号 d_j 表示空间位置结点 j 上是否有机器人，其中 $j \in Pos$。当空间位置结点 j 上，有机器人时，那么 $d_j = 1$，表示该空间位置结点上有 1 个机器人。当空间位置结点上没有机器人时，那么 $d_j = 0$，表示该空间位置结点上有 0 个机器人。

假设在空间位置 j 上有一个机器人 r，即 $d_j = 1 \wedge c(r) = j$。使用符号 $\delta^F_{c(r)}$ 表示机器人 r 的环境快照，其中F 表示机器人获取环境快照的方向，$F \in \{+, -\}$，$+$ 表示顺时针，- 表示逆时针。

在有n 个位置结点的环形空间上，空间位置结点j 上机器人 r 的顺时针和逆时针环境快照如下：

顺时针环境快照：$\delta^+_{c(r)} = \langle d_j, d_{j+1}, ..., d_{j+n-1} \rangle$.

逆时针环境快照：$\delta^-_{c(r)} = \langle d_j, d_{j-1}, ..., d_{j-n+1} \rangle$.

具体看一个实例，在图3.6中空间位置结点1 上的机器人顺时针和逆时针环境快照如下：

顺时针环境快照： $\delta_1^+ = \langle 1,0,0,1,0,0,0,0,1,0,... \rangle$.

逆时针环境快照： $\delta_1^- = \langle 1,0,...,0,1,0,0,0,0,1,0,0 \rangle$.

使用上述环境快照表达方式，比较直接、简洁。但是当环形空间中位置结点数较多或者未知时，这种表达式就十分长，不容易书写或者存储。F-R 表达式 [9] 描述环境快照就可以很好的避免这些问题，F-R 表达方式中使用 F_m 表示连续 m 个没有机器人空间位置结点数，R_n 表示连续n 个有机器人空间位置结点数。

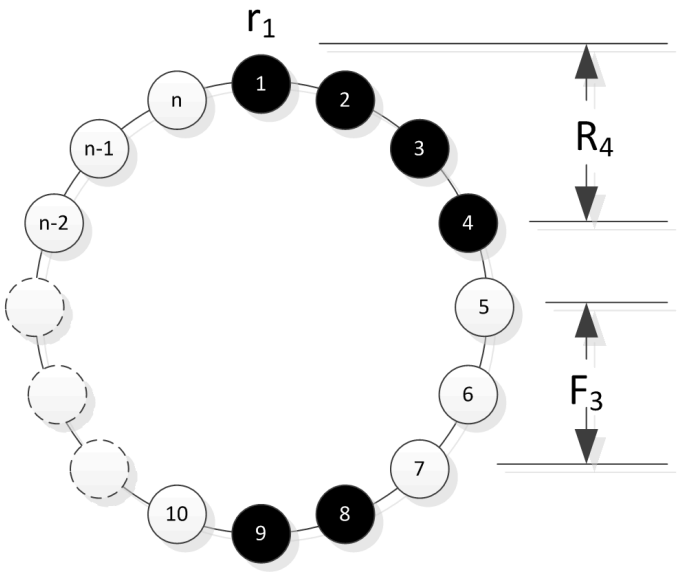

图 3.7: 环形空间 F-R 快照表达式

图3.7中有n 个机器人，机器人 r_1 顺时针方向，有连续 4 个空间位置上有机器人，可以使用 R_4 表示。紧跟顺时针方向又有连续的三个无机器人的空间位置结使用 F_3 表示。那么在空间位置结点 1 的上的机器人 r_1 的顺时针F-R 快照表达式为 $\delta^+_{c(r_1)} = \langle R_4, F_3, R_2, F_{n-9} \rangle$。可以看出在F-R 快照表达式中，使用具体整数或者未知参数表示连续的空间位置，这样的描述不仅缩短了表达式的长度，而且描述能力更强。当在未知环形空间位置结点数量，F-R 快照表达式也可以很方便的进行描述。

3.3.3 对称性

环形空间中，机器人的位置关系可能会出现对称性的情况。若是出现对称，可能导致某个机器人的顺时针和逆时针环境快照完全相同，或者两个对称的机器人环境快照相同。

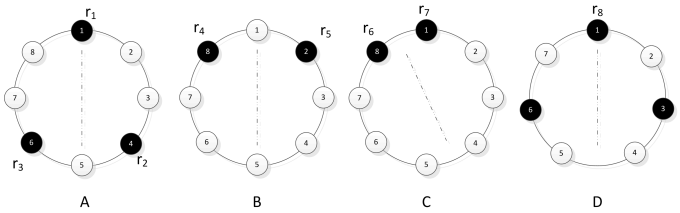

图 3.8: 环形空间对称

如图3.7，有四组对称环形空间模型，A 中机器人 r_1 的顺时针和逆时针 F-R 快照表达式为：

$\delta^{+}_{c(r_1)} = \langle R_1, F_2, R_1, F_1, R_1, F_2 \rangle$.

$\delta^{-}_{c(r_1)} = \langle R_1, F_2, R_1, F_1, R_1, F_2 \rangle$.

可以看出 $\delta^{+}_{c(r_1)}$ 和 $\delta^{-}_{c(r_1)}$ 完全相同。B 中机器人 r_4 和 r_5，B 中机器人 r_6 和 r_7 是对称的。A 中位置结点个数是偶数，D 中位置结点个数是奇数，表明环形空间位置结点数，无论是奇数还是偶数，都可能会出现对称的情况。

3.3.4 移动算法

在环形空间中，机器人有三种可能的移动策略：不移动，顺时针移动、逆时针移动。而对于无方向识别能力的机器人而言，其移动策略只能依据当前环境快照做出前进或者后退的移动策略。而环形空间具有对称性，所以机器人的移动策略有四种：$(Idle)$、前进 $(Front)$、后退 $(Back)$、未确定 $(Doubt)$。移动策略集合 $MOVE = \{Idle, Front, Back, Doubt\}$。其中Doubt 表示机器人可以前进或者后退。

移动算法是一组移动规则，移动规则使用符号L 表示，其结构如下：

$L \square \delta^{F}_{c(r)} \rightarrow r.MOVE$

机器人r 的环境快照与L 的 $\delta^{F}_{c(r)}$ 匹配时，机器人r 就执行移动规则L 中的移动决策 $MOVE$。

下面通过最小移动算法 [9] 的介绍，可以更详细的了解移动算法。最小移动算法是环形空间中使用最少的机器人完成未知空间永恒探索 [9]。具体最小移动算法的条件是空间位置结点数 $n \geq 10$, 机器人数 $k = 3$，且n 与k 互质。最小移动算法分为两个阶段：稳定阶段 $(Legitimate \quad phase)$ 和收敛阶段 $(Convergence \quad phase)$。

表 3.1: 最小移动算法稳定阶段

$RL1$	$\delta^{F}_{c(r)} = \langle R_2, F_2, R_1, F_{n-5} \rangle .$	$\rightarrow$	$r.Back$
$RL2$	$\delta^{F}_{c(r)} = \langle R_1, F_1, R_1, F_{n-6}, R_1, F_2 \rangle .$	$\rightarrow$	$r.Front$
$RL3$	$\delta^{F}_{c(r)} = \langle R_1, F_3, R_2, F_{n-6} \rangle .$	$\rightarrow$	$r.Front$

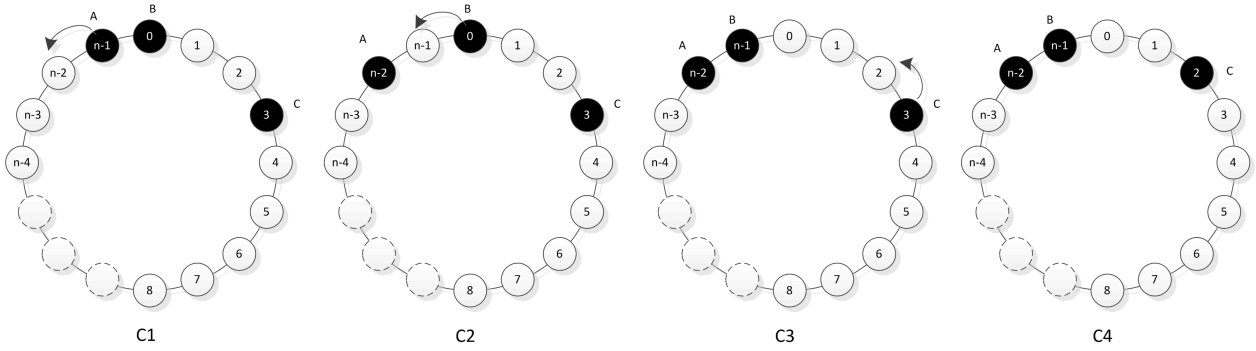

图 3.9: 最小移动算法的稳定阶段

在图3.9中，所有机器人是依据最小移动算法执行移动。C1 中机器人 A 的顺时针环境快照 $\delta^{+}_{c(A)}$ 匹配移动算法 $RL1$ 做出后退的移动决策，机器人 A 从位置结点 $n-1$ 移动到 $n-2$, 其他机器人 B 和 C 此时没有匹配任何移动算法，保持静止。C2 中机器人 B 的逆时针环境快照 $delta^{-}_{c(B)}$ 匹配移动算法 $RL2$，做出前进的移动决策，机器人 A 从位置结点 0 移动到 $n-1$。同样机器人 C 的逆时针环境快照，匹配移动 $RL3$ 做出前进的移动决策，从位置结点 3 移动到 2。此时，C4 相对于C1 是所有的机器人向顺时针方向移动了一个位置。C4 和C1 所有机器人的环境快照完全相同。意味着所有机器人会按照上述 C1 到 C4 过程重复执行移动过程。

像这样所有机器人在某一个状态开始，循环执行一组移动规则，而且在移动规则下，所有机器人都向某一个固定的方向移动，称为稳定阶段。相对于稳定阶段的是收敛阶段：

表 3.2: 最小移动算法收敛阶段

$RC1$	$4 \leq x \leq z \wedge \delta^F_{c(r)} = \langle R_1, F_x, R_2, F_z \rangle$.	$\rightarrow$	$r.Front$
$RC2$	$x \neq y, x > 0 \wedge \delta^F_{c(r)} = \langle R_1, F_x, R_1, F_y, R_1, F_x \rangle$.	$\rightarrow$	$r.Doubt$
$RC3$	$0 < x < z < y \wedge (x, y) \neq (1, 2) \wedge \delta^F_{c(r)} = \langle R_1, F_3, R_2, F_{n-6} \rangle$.	$\rightarrow$	$r.Front$
$RC4$	$\delta^F_{c(r)} = \langle R_3, F_{n-3} \rangle$.	$\rightarrow$	$r.Back$
$RC5$	$\delta^F_{c(r)} = \langle R_1, F_1, R_2, F_{n-4} \rangle$.	$\rightarrow$	$r.Back$

在表**??**中是最小算法的收敛阶段。在未知空间中，机器人的初始位置可以是任意的，机器人通过收敛阶段可以到达稳定阶段。

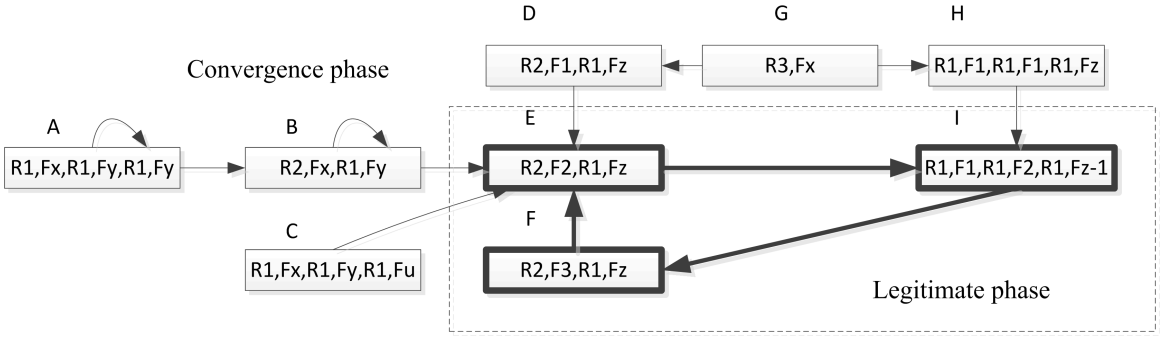

图 3.10: 稳定阶段和收敛阶段

通过图3.10来解释最小移动算法的收敛阶段和稳定阶段。图中每个长方形块表示整个环形空间可能出现的环境状态。收敛阶段使用集合LC 表示，集合 LC 包含 $\{A, B, C, D, G, H\}$ 环境状态。稳定阶段环境使用集合RL 表示，集合 RL 包含 $\{E, F, I\}$ 环境状态。图中包含了所有最小移动算法中，机器人所有可能初始的位置。

下面详细介绍一下E 到I 的状态变化过程，状态变化过程也基本相同，可以参照理解。

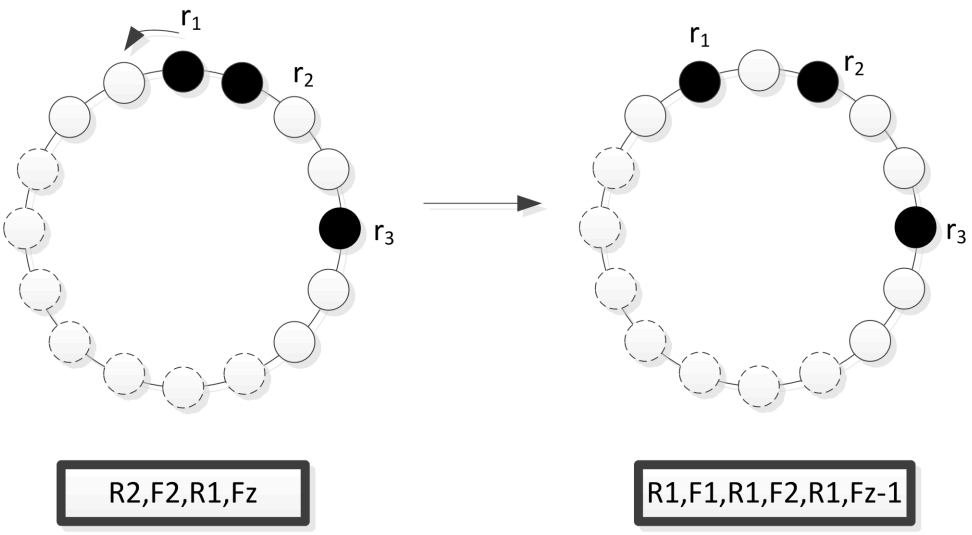

图 3.11: 环境快照状态变化

环境状态 $R2, F2, R1, Fz$ 变化 $R1, F1, R1, F2, R1, Fz-1$ 的过程中，是由于机器人 r_1 匹配移动规则 RL_1 之后，做出向后移动决策。由此可知图3.10中是由于机器人的移动，导致环境状态的变化。

由图3.10可知，最小移动机器人算法中所有的非稳定状态，都可以通过收敛阶段到达稳定状态。进入稳定状态之后，就不会回到收敛状态。稳定状态是一个闭环图，进入稳定状态之后，所有机器人都循环执行稳定阶段的移动算法。

3.4 永恒探索

永恒探索是空间中所有机器人满足对空间中所有位置结点进行重复访问的性质。永恒探索只是移动算法满足非冲撞性 (No collision)，非互换性 (No switch)，非终止性 (Live)。

3.4.1 非冲撞性

冲撞是指在环形空间中，某个空间位置结点上，同一时刻有 2 个或者更多的机器人。

如图3.12所示，当机器人 r_2 移动到机器人 r_1 所在空间位置结点或者机器人 r_1

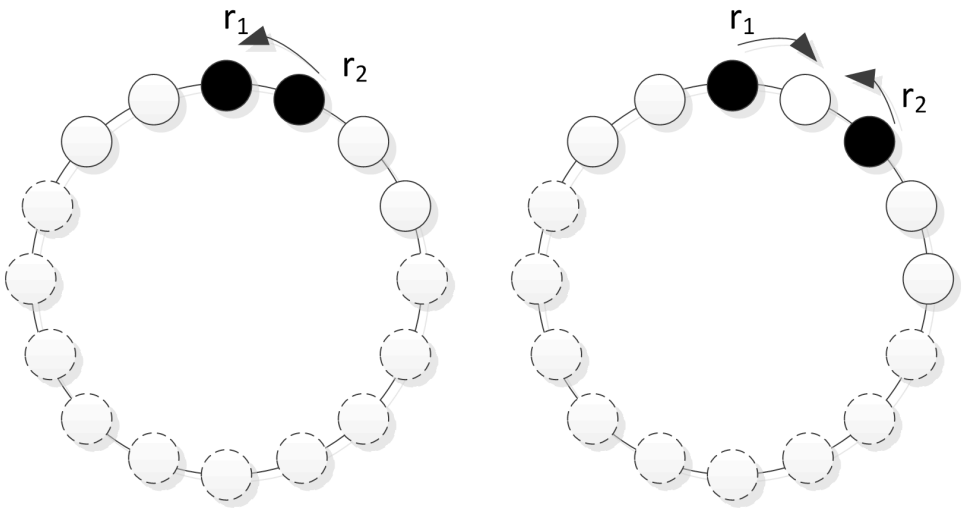

图 3.12: 冲撞

和 r_2 同时移动到同一个空间位置结点时，就会发生碰撞，所谓非冲撞性，则是指所有机器人移动过程中，永远不会发生冲撞。

3.4.2 非互换性

互换是指相邻的两个机器人，下一个状态时，它们所在的空间位置发生的互换。

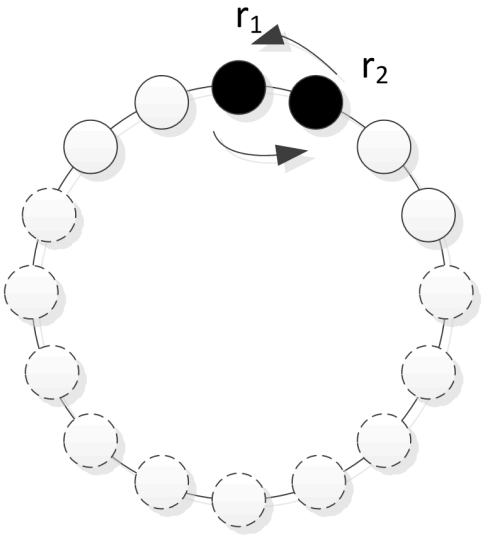

图 3.13: 互换

如图3.13所示，机器人 r_1 和 r_2 同时相对移动，导致下一个状态时，机器人的

位置发生的互换。非互换性是指这种机器人互换空间位置的情况在机器人移动过程中，永远不会发生。

3.4.3 非终止性

非终止性包含两层意思：第一层意思是，空间中每个机器人对空间中的每个位置结点都进行访问；第二层意思是，每个机器人对每个空间位置结点进行重复的访问，而且是永不停止的重复。

3.5 本章小结

本章节介绍了离散空间模型中，自主移动机器人的移动，分别对机器人的三个移动阶段：观察、计算和移动，进行了详细的介绍。对机器人的三种调度策略：半同步调度策略、完全同步调度策略和完全异步调度策略进行了描述。对环形空间和环形空间快照、最小移动算法分别进行了介绍和举例。最后，介绍了永恒探索的三条性质。

第四章　永恒探索算法建模

判定移动机器人移动算法为永恒探索算法的核心，是验证其在公平性先决条件下是否同时满足永恒探索算法的三条性质：非冲撞性、非互换性和非终止性。将机器人和探索空间视为一个完整的系统，在这个系统中不仅包括机器人和空间位置，而且包括机器人移动算法和调度模型。使用 nuXmv 对系统中空间位置、机器人性质、移动算法以及不同调度进行抽象，从而模拟移动算法中的状态迁移。本文中使用的空间模型是离散空间模型，不仅简化了模型的构建，而且将建模的核心放在了移动算法和机器人性质的构建上。考虑到不同拓扑结构的离散空间有不同的性质和特点，从而对于不同的结构的离散空间，会根据其自身特点进行模型的定义。为了更加具体说明永恒探索算法的建模与验证过程，本文以验证环形空间上最小移动算法是否是永恒探索算法为例，介绍符号模型检验方法在永恒探索算法中的应用思想。

4.1　机器人建模

利用 nuXmv 提供的参数化模块 (parameterized module) 对机器人进行建模，参数化模块在实例化模块是可以给模块实例传入所有机器人位置编号变量。在机器人模块中定义变量 phase 表示机器人所处的操作阶段。机器人中观察 (Look) 和计算 (Compute) 在模型中可以合并为一个操作，称作观察计算，符号定义为 lc(LookCompute)，因为这两个操作只是做出了一个移动决策。机器人处于移动操作使用符号 m 表示，那么 phase 变量只有两种取值 lc 和 m。机器人在观察移动操作做出的移动决策可以使用移动变量 move 表示，在环形离散空间模型中，move

有三种取值 -1、0 和 1，-1 表示机器人逆时针移动，0 表示机器人不发生移动，1 表示机器人顺时针移动。

```
MODULE robot(p1,p2,...)
    VAR
        phase : {lc,m};
        move : -1..1;
    ...
```

上述声明的是机器人模块，模块名称为 robot，并且模块带有位置环形空间所有机器人位置传入参数 p1,p2,...。在机器人模块实例化时，参数列表第一个位置值p1 赋值为当前实例机器人在环形空间中的位置值，以当前机器人位置为起点，沿着环形空间顺时针方向依次是 p2、p3...，即参数列表是以当前实例机器人的位置为起点，沿着环形空间顺时针方向上的机器人依次传入。

4.1.1 移动算法匹配

机器人的移动算法是机器人在计算操作时，作为移动决策的判断依据。在 nuXmv 建模过程采用顺时针或者逆时针相邻两个机器人的之间的没有机器人空间位置结点的个数作为匹配依据。相邻机器人 A 到机器人 B 之间没有机器人空间位置结点个数使用 $gap^{F}_{A\to B}$ 表示，其中 $F \in \{+,-\}$ 表示gap 的计算取值方向，$+$ 表示顺时针，- 表示逆时针。

$gap^{F}_{A\to B}$ 的计算公式如下:

$$gap^{+}_{A\to B} = (c\,(B) - c\,(A) + n)\ \ mod\, n$$

$$gap^{-}_{A\to B} = (n - ((c\,(B) - c\,(A) + n)\ \ mod\, n))\ \ mod\, n$$

计算公式中 n 表示环形空间位置结点个数。机器人在环形空间中任意一个时刻都有占据一个空间位置。如图4.1中所示，环形空间结点的位置是从数字 0 开始验证环形空间顺时针方向依次递增编号，最后一个编号为 n-1，其中 n 是表示环形空间中位置结点的个数。图中顺时针 1 和顺时针 2、逆时针 1 和逆时针 2 可以看

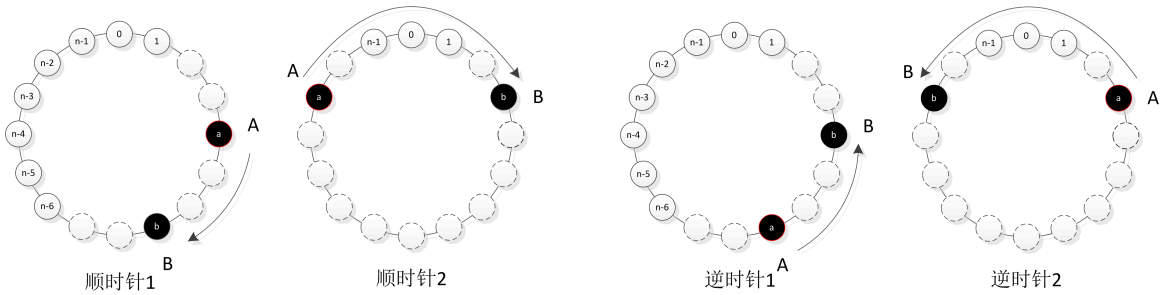

图 4.1: 机器人 A 和 B 之间顺时针和逆时针 gap 计算示意图

出，当顺时针和逆时针计算机器人 A 到机器人 B 的 $gap^F_{A\to B}$ 时，机器人 A 和机器人 B 之间包含位置结点为 0 的情况。机器人 B 与机器人 A 之间的顺时针间隔为两者节点位置的差加 n 再以 n 为基数取模。逆时针的间隔则为 n 减去顺时针的间隔得到的结果再以 n 为基数取模. 之所以继续取模是考虑 A 与 B 位置重叠的情况，此时，顺时针与逆时针方向的间隔均为 0。

假设机器人 r 的快照 F-R 视觉快照 $\delta^F_{c(r)}$ 满足移动算法中的某条规则 L，记为 $match\left(\delta^F_{c(r)}, L\right)$。其中规则 L 可确定的移动方向使用符号 $\beta_{(L)}$ 表示。机器人在每次计算阶段之后，会获取一个移动决策前进 (Front)，后退 (Back) 或者保持不动 (Idle)，那么分别对应的移动量为 1，-1，0。所以机器人 r 的移动决策可以使用其移动量 $M_{(r)}$ 表示，具体计算公式如下：

$$M_{(r)}=\begin{cases}1 & if\left(match\left(\delta^F_{c(r)},L\right)\wedge F=+\wedge\beta\left(L\right)=Front\right)\vee\\ & \left(match\left(\delta^F_{c(r)},L\right)\wedge F=-\wedge\beta\left(L\right)=Back\right)\\ -1 & if\left(match\left(\delta^F_{c(r)},L\right)\wedge F=-\wedge\beta\left(L\right)=Front\right)\vee\\ & \left(match\left(\delta^F_{c(r)},L\right)\wedge F=+\wedge\beta\left(L\right)=Back\right)\\ 0 & otherwise\end{cases}$$

对于环形空间，若机器人r 在空间位置 $c\left(r\right)$ 的顺时针和逆时针快照相同 $\delta^+_{c(r)}=c\left(r\right)^-$，也就是说机器人r 的快照是对称的。若此时机器人 r 的快照匹配移动算法中某条规则 L, 根据规则 L 得出的移动方向 $\beta\left(L\right)$ 无论是前进或者后退，即机器人的移动量可以是 1 或者 -1，此时机器人会随机选择其中一个作为最终

的移动决策。若r 的为快照不匹配算法当中的任意一个移动规则时移动量就为 $M_{(r)}=0$。

下面代码描述环形空间的节点数为 n=10，机器人个数为 k=3 时的最小移动算法。$next\,(move)$ 是机器人r 的移动量 $M_{(r)}$。$phase=lc$ 是移动规则匹配的前置条件，当机器人处于观察计算阶段时，才会匹配移动算法。

```
1  next(move) :=          ——变量 move 在下一步的值
2      case
3          phase = lc & ((p2 - p1 + 10) mod 10 = 1) & ((p3 - p2 + 10) mod 10 = 3) : -1;
4          ——顺时针RC1
5          phase = lc & ((10 - ((p3 - p1 + 10) mod 10)) mod 10 = 1) & ((10 - (p2 - p3 + 10) mod 10) mod 10
               = 3) : 1;
6          ——逆时针RC1
7          ...
8
9          phase = lc & ((p2 - p1 + 10) mod 10 >= 1) & (((p2 - p1 + 10) mod 10) = ((p1 - p3 + 10) mod 10)
               ) & ((p3 - p2 + 10) mod 10 != (p2 - p1 + 10) mod 10) : {1,-1};
10         ——顺时针RC2
11         phase = lc & ((10 - ((p3 - p1 + 10) mod 10)) mod 10 >= 1) & ((10 - ((p3 - p1 + 10) mod 10)) mod
               10 = (10 - ((p1 - p2 + 10) mod 10)) mod 10) & ((10 - (p2 - p3 + 10) mod 10) mod 10 != (10
               - ((p3 - p1 + 10) mod 10)) mod 10) : {1,-1};
12         ——顺时针RC2
13
14         ...
15
16         TRUE :  0;
17         ——other
18     esac;
```

代码 3-4 行描述规则 RC1 的顺时针方向的匹配，p1 表示当前机器人实例的自身位置，在 p1 位置上的机器人 r 从自身开始, 按照顺时针方向, 依次计算 p1 到 p2、p2 到 p3 之间的间隔个数. 对应匹配 RL1 中机器人的之间的间隔个数, 以此作为机器人 r 顺时针匹配移动算法的依据，当满足匹配时next(move) 的取值为 -1。代码 5-6 行描述 RC1 的逆时针方向的匹配，逆时针方向的匹配则是计算 p1 到 p3、p3 到 p2 之间的间隔个数作为匹配依据, 当满足匹配时next(move) 的取值为 1。代码 9-12 行描述规则 RC2 的顺时针和逆时针匹配，由于 RC2 规则描述的是对称情况下的移动规则，对于匹配该规则机器人的移动量可能是 -1 或者 1。

4.1.2 机器人移动

机器人经过观察计算阶段之后，会进入移动阶段。机器人在观察计算和移动两种阶段之间切换。

```
next(phase) :=
        case
            phase = lc     :  m;
            phase = m      :  lc;
            TRUE           :  phase;
        esac;
```

在 nuXmv 建模过程中，机器人移动阶段状态分为两个：观察计算，符号 lc 表示; 移动，符号 m 表示。下一个移动阶段状态next(phase) 取决于当前移动阶段状态 phase：当当前阶段状态 phase 为 lc 下一个移动阶段状态next(phase) 的值为 m; 当当前阶段状态 phase 为 m，下一个移动阶段状态next(phase) 的值为观察计算阶段 lc;

当机器人移动阶段状态为移动 m 时，其下一个位置结点编号为 $next\left(c\left(r\right)\right) = c\left(r\right)$。机器人移动至新的空间位置上，改变了整个空间中位置快照，所有机器人会根据新的位置快照，做出对应的移动决策。

```
next(p1) :=
        case
            phase = m & (move + p1) < 0                    : 10 + (move + p1);
            phase = m & (move + p1) >= 0 & (move + p1) < 10 : (move + p1);
            phase = m & (move + p1) >= 10                  : (move + p1) - 10;
            TRUE                                           : p1;
        esac;
```

next(p1) 表示当前模块实例化的机器人下一个位置，这个过程只在机器人当前移动阶段状态 phase 为 m 时才会进行。考虑到空间位置范围问题，当机器人顺时针和逆时针通过位置为 0 的结点时，都会对新的空间位置值进行对应的调整。将机器人的新的位置计算过程归纳如下：

$$next\left(p1\right) = \begin{cases} (move + p1) + n & if\ (move + p1) < 0 \\ (move + p1) & if\ (move + p1) >= 0 \wedge (move + p1) < n \\ (move + p1) - n & if\ (move + p1) >= n \end{cases}$$

4.2 调度策略建模

nuXmv 模块化建模的优点是代码可移植性强，在完全同步调度策略、半同步调度策略和完全异步调度策略进行建模时，主模块和机器人模块的建模代码可以复用。三种不同的调度策略情况下，只需根据不同的调度特有性质进行建模调整。

半同步调度策略建模 对于半同步调度策略, 每个机器人在调度过程中只有选中和非选中两种情况。在机器人模型中定义调度变量dispatcher，变量声明如下：

```
VAR dispatcher : {choose,steady};
```

使用关键字VAR 定义符号枚举类型变量dispatcher，表示机器人的调度标识。其枚举元素有choose 和steady，分别表示机器人被选中和没被选中。在半同步调度模型中，机器人的每个移动过程中，是随机调度的。机器人随机调度建模如下：

```
next(dispatcher) :=
        case
            TRUE        :   {choose,steady};
        esac;
```

建模代码中机器人下一个调度标识只有一个永真的条件分支，并且取值是{choose,steady}，即机器人每次移动过程中下一个调度标识是choose 和steady当中随机取值。机器人在匹配移动算法时，若 dispatcher 为 choose，才进行移动算法的匹配。

```
    next(move) :=          ——变量 move 在下一步的值
        case
            dispatcher = choose & phase = lc & ((p2 - p1 + 10) mod 10 = 1) & ((p3 - p2 + 10) mod 10 = 3) :
                -1;
            ——顺时针RC1
            dispatcher = choose & phase = lc & ((10 - ((p3 - p1 + 10) mod 10)) mod 10 = 1) & ((10 - (p2 - p3
                + 10) mod 10) mod 10 = 3) : 1;
            ——逆时针RC1

            ...

            TRUE :  0;
        esac;
```

移动算法建模中，对每个移动规则匹配添加先决条件dispatcher = choose，表明机器人只有在选中是才会计算移动量，否则移动量永远为 0。

完全同步调度策略建模 完全同步调度策略是半同步调度策略的一种特殊情况，空间中所有机器人在每次移动过程中都会被调度。在半同步调度策略建模的基础上，调整模型中调度标识的值永远是选中choose:

```
next(dispatcher) :=
        case
          TRUE          :  choose;
        esac;
```

完全异步调度策略建模 完全异步调度策略中所有机器人的移动都是并行的，在nuXmv 中实例化机器人模型时，通过关键字process 声明实例是异步的。

```
VAR
    r1 : process robot(pos1,pos2,pos3);
    r2 : process robot(pos2,pos3,pos1);
    r3 : process robot(pos3,pos1,pos2);
```

实例化三个异步机器人 r1、r2、r3，在模型验证的过程中，为了确保机器人公平获得执行时间片，可以在机器人模块定义中添加公平性约束。

```
FAIRNESS
    running
```

4.3 永恒探索性质

判定移动算法是否是永恒探索算法，需要移动算法满足三条性质：非冲撞性、非互换性和非终止性。下面以环形空间位置数为 10，机器人数为 3 的最小移动算法为例，任意机器人在任意初始位置，使用 LTL 公式描述永恒探索算法的三条性质并验证模型对性质的满足性。主模块中声明三个机器人位置变量 pos1、pos2、pos3，作为全局变量记录三个机器人的位置。建模时设定三个机器人位置排列顺序是按照顺时针方向依次排列 pos1、pos2、pos3。

```
MODULE main
  VAR
    pos1 : 0..9;
    pos2 : 0..9;
    pos3 : 0..9;
    r1 : robot(pos1,pos2,pos3);
    r2 : robot(pos2,pos3,pos1);
    r3 : robot(pos3,pos1,pos2);
```

环形空间结点数为 n 时，空间位置结点编号为 0...n-1。当 n=10 时，机器人的位置编号范围是 0 到 9。在机器人模块实例化时，传入参数列表的第一个参数为当前机器人实例的位置值，后续依次是顺时针方向上机器人位置。例如机器人 r1 实例化中, 第一位参数 pos1，表示当前实例是 pos1 位置上的机器人，并且后续是以 pos1 为起点顺时针方向依次是 pos2 和 pos3，进行位置参数传入。

使用 LTL 定义机器人初始位置时，只需限制每个机器人位置的范围和顺时针排列次序。

```
1 (((pos3 > pos2) & (pos2 > pos1)) | ((pos1 > pos3) & (pos3 > pos2)) | ((pos2 > pos1) & (pos1 > pos3)))
```

按照顺时针方向，pos1、pos2、pos3 的大小次序有三种可能性:1、pos1<pos2<pos3;2、pos2<pos3<pos1;3、pos3<pos1<pos2; 在 nuXmv 中未给变量分配初始值时，会对该变量的所有取值可能性进行穷举验证。该初始化定义可以实现机器人初始位置任意性的描述。

非冲撞性 非冲撞性是指离散空间模型中，同一时刻同一位置结点上至多只有一个机器人，使用 LTL 描述就是同一时刻任意两个机器人所在位置不能相同，如下：

```
1 LTLSPEC (((pos3 > pos2) & (pos2 > pos1)) | ((pos1 > pos3) & (pos3 > pos2)) | ((pos2 > pos1) & (pos1 >
      pos3))) -> (G ((pos1 != pos2) & (pos2 != pos3) & (pos1 != pos3)))
```

上述 LTL 定义中，三个机器人位置 pos1、pos2 和 pos3 的初始化位置是任意的。nuXmv 中 LTL 符号 G 表示在将来所有时刻都满足一定条件，在非冲撞性中，表示将来所有时刻都满足 pos1、pos2 和 pos3 中任意两个位置都不都相同。

非互换性 非互换性是指非互换性对应的 LTL 公式表示如果两个机器人 r_i 与 r_j 物理上相邻, 即其所在节点的编号相差 1, 则一定不会出现 r_i 与 r_j 在下一个状态位置互换的情况。

```
1 LTLSPEC (((pos3 > pos2) & (pos2 > pos1)) | ((pos1 > pos3) & (pos3 > pos2)) | ((pos2 > pos1) & (pos1 >
      pos3))) -> G ((((pos1 + 1) mod 10) = pos2) -> (X (((pos2+1) mod 10) != pos1)) & (((pos2 + 1) mod
      10) = pos3) -> (X (((pos3+1) mod 10) != pos2)) & (((pos3 + 1) mod 10) = pos1) -> (X (((pos1+1) mod
       10) != pos3)))
```

非互换性中机器人初始位置定义与非冲撞性中的相同。nuXmv 中 LTL 符号 X 表示在下一个时刻都满足一定条件。由于先决条件是机器人位置 pos1、pos2 和 pos3 是在环形空间中是顺时针方向获取的，若当前机器人位置加上 1 后取模等于前面的机器人位置，那么说明它们相邻，在下一个时刻前面的机器人位置加上 1 后取模等于前面的机器人位置，表明两者交换了位置。例如 ((pos1 + 1) mod 10) 等于前面位置上的机器人位置 pos2，表示 pos1 与 pos2 相邻，下一个时刻 pos2+1 取模不能等于 pos1，即 pos1 和 pos2 位置不能互换。

非终止性 非终止性离散空间中每个机器人对每个空间位置结点进行反复的巡查。

```
1 LTLSPEC (((pos3 > pos2) & (pos2 > pos1)) | ((pos1 > pos3) & (pos3 > pos2)) | ((pos2 > pos1) & (pos1 >
      pos3))) -> (((G F (pos1 = 0)) & (G F (pos1 = 1)) &...& (G F (pos1 = 9))) & ((G F (pos2 = 0)) & (G F
      (pos2 = 1)) &...& (G F (pos2 = 9))) & ((G F (pos2 = 0)) & (G F (pos2 = 1)) &...& (G F (pos2 = 9))))
```

命题 p 描述机器人的位置取值，例如 pos1=1，pos1=2 等等。LTL 公式 G F p 等价于 G (F p),F p 表示将来的某个时刻 p 为真，G (F p) 表示将来 (F p) 一直为真，即 p 在未来的某些时刻为真。在此表示机器人对某个空间结点反复巡查。

4.4 本章小结

本章以最小移动算法为例，详细介绍了使用 nuXmv 对机器人模块定义包括移动算法建模、移动算法的匹配。根据三种不同调度策略的性质，对建模做了对应调整。使用 LTL 公式描述永恒探索算法的三条性质：非冲撞性、非互换性和非终止性。通过模型真实的模拟了环形空间中机器人的移动行为，定义了永恒探索算法的性质。

第五章　自主移动机器人永恒探索算法验证

本文通过 nuXmv 符号模型检测工具对自主移动机器人永恒探索算法进行建模。通过参数化模块定义了机器人模块，包括移动算法、移动算法匹配和移动决策的执行。目前存在三种机器人调度策略，所以针对三种调度策略对机器人模块分别进行了对应的模型调整。为了验证永恒探索探索算法的完备性、准确性和满足性，通过 LTL 公式对机器人的任意初始位置和永恒探索算法的性质进行了定义。

以最小移动算法为验证对象，在完全同步调度策略、半同步调度策略和完全异步调度策略三种调度策略下进行形式化建模，并分别验证移动算法是否满足非冲撞性、非互换性和非终止性。验证内容如表 5.1 所示。

表 5.1: 最小移动算法验证概览

调度策略	非碰撞性	非互换性	非终止性
完全同步调度策略	验证	验证	验证
半同步调度策略	验证	验证	验证
完全异步调度策略	验证	验证	验证

5.1　验证结果与分析

根据上一章节中定义的自主移动机器人空间永恒探索算法模型，利用 nuXmv 符号模型检测工具对最小移动算法非冲撞性、非互换性和非终止性在不同的调

度策略下进行模型检测。实验运行的系统环境为 Windows 7 旗舰版，硬件环境为 CPU Intel Xeon(R) 3.40GHz,16G 内存。图5.1给出了非碰撞性、非碰撞性和非互换性三个性质在不同机器人调度策略下的验证结果和验证所需时间。实验中考虑了节点个数分别为 10 到 17 的情况，其中由于算法要求机器人个数与节点个数互质, 节点为 12 与 15 的情况在实验中略去。

节点数	完全同步调度模型						半同步调度模型						异步调度模型					
	非碰撞性		非终止性		非互换性		非碰撞性		非终止性		非互换性		非碰撞性		非终止性		非互换性	
	结果	耗时	结果	耗时	结果	耗时	结果	耗时	结果	耗时	结果	耗时	结果	耗时	结果	耗时	结果	耗时
10	✓	1.4s	✓	484.0s	✓	1.9s	✓	0.7s	✓	83.1s	✓	1.7s	×	1.2s※	×	14.7s※	✓	0.9s
11	✓	5.4s	✓	3181.1s	✓	4.3s	✓	1.1s	✓	723.1s	✓	4.3s	×	9.3s※	×	18.1s※	✓	0.7s
13	✓	8.3s	✓	42.4s*	✓	6.8s	✓	1.1s	✓	63.9s*	✓	6.7s	×	10.7s※	×	2.1s※	✓	0.6s
14	✓	8.9s	✓	90.4s*	✓	8.4s	✓	1.1s	✓	129.2s*	✓	8.8s	×	9.7s※	×	48.8s※	✓	0.6s
16	✓	1.2s	✓	168.4s*	✓	2.3s	✓	0.6s	✓	389.2s*	✓	2.4s	×	4.9 s※	×	19.6s※	✓	0.7s
17	✓	13.0s	✓	315.7s*	✓	11.4s	✓	2.0s	✓	562.3s*	✓	12.2s	×	27.5s※	×	102.2s※	✓	0.6s

注: ✓表示验证结果为真;×表示验证结果为假;*表示在设置初始状态的情况下验证所需时间;-表示超时;
※表示采用 SMT 方法验证所需时间;未标注的时间均表示采用 BDD 方法所需的验证时间.

图 5.1: 最小移动算法实验结果

实验结果表明最小移动算法在完全同步调度策略和半同步调度策略下，满足对环形空间永恒探索性质，即移动算法同时满足非冲撞性、非互换性和非终止性。而在异步调度模型下，验证的结果为移动算法不满足非碰撞性与非终止性，验证过程中 nuXmv 返回相应的反例。根据反例分析，永恒探索性质不能被满足的原因是异步过程中机器人使用过时的快照信息做出的移动决策会导致相邻的机器人发生碰撞，此时同一个位置结点上有两个机器人，后续所有机器人都没有与移动算法相匹配，导致所有的机器人都不能移动。由此可见, 非终止性不被满足的原因是非碰撞性没有被满足。这一结果与 Ha 等人利用 Maude 模型检测得出的反例相同。在其反例中，用于出现了两个机器人碰撞的情况而导致所有机器人无法移动，Ha 等人将此情况称为死锁状态。与 Ha 等人的验证不同的是其方法需要给出具体的初始状态才发现了反例，而如何发现导致反例的初始状态文章并没有交待. 本文提出的方法可以在不给出初始状态的前提下依然找到对应的反例。

nuXmv 支持基于 BDD 和 SMT 方法的验证。对于验证性质的正确性，BDD 方法的效率相对较高，而对于不被满足的性质，SMT 方法可以更快的找到反例。在实验中采用两种方法对三个性质进行验证。结果表明大部分验证都可以在相对较短的时间内完成。虽然随着空间节点数的增长验证所需的时间也会有所增加，但依然可以在较合理的时间如一小时内完成。同 Béatrice 与 Ha 等人的工作相比，nuXmv 找到反例的时间更短。然而在验证被算法满足的性质时，nuXmv 所需的时间相对较长，这是因为 nuXmv 不需要设定具体的初始状态，因此其搜索的状态空间比固定初始状态时更大，所需的时间则较长。实验数据中有部分实验是设定机器人初始状态的情况下进行的，验证所需的时间明显缩短。

5.2 本章小结

本章详细列出了最小移动算法的永恒探索性质的实验结果，从实验结果中可以得出符号模型对移动机器人空间永恒探索算法验证的高效性。

第六章　总结和展望

6.1　总结

本文使用 nuXmv 符号模型验证工具对自主移动机器人空间永恒探索算法进行建模与验证，对实验结果进行分析。使用参数模块定义了机器人模型，LTL 描述永恒探索的性质。通过使用 nuXmv 工具对最小移动算法的验证，提出了一种对空间永恒探索算法的验证方法。

本文中主要的工作包括以下几点：

第一，本文提出了机器人符号化建模方式，使用参数模块定义机器人模型，对移动算法及其匹配进行了描述。同时，使用 LTL 对空间永恒探索性质也进行了定义。

第二，本文在完全同步调度策略、半同步调度策略和完全异步调度策略下，实现了空间探索算法的模型构建，并使用 BDD 和 SMT 验证方式，分别验证模型的永恒探索性质。

第三，本文在已有自主移动机器人用探索算法形式化验证的基础上，实现了符号化模型的验证与分析。不仅提升的机器人移动算法的验证效率，而且扩展了验证功能。能够在任意机器人初始位置时，验证移动算法性质。

6.2　展望

本文主要是研究机器人空间探索问题。目前在使用形式化方法对机器人移动算法进行验证的研究工作并不是很多。本文也只是针对机器人空间探索问题的一小部分进行了研究。

机器人领域的研究十分广泛，除了空间探索之外，还有仿生学、网络机器人、多机器人系统等等。目前机器人领域的研究已经取得了很多成果，但是在某些方面离实用要求还是有些差距。随着传感器技术、机器学习和嵌入式技术的不断发展，机器人在未来人们生活和生产中逐渐扮演重要角色。

对于自主移动机器人空间探索问题，使用符号模型检测方法可以有效的避免状态爆炸问题，然而验证随着空间位置结点和机器人数增加，符号模型的验证的时间也在快速增长。本文提出的符号模型验证方法，目前还不能直接用于实际系统中空间结点或机器人结点较多的应用场景。在符号模型验证方法的基础上，通过抽象或者归纳技术，如 nuXmv 内置的 k-induction 方法，解决验证效率问题，是未来需要进一步深入研究的工作。

参考文献

[1] 贺伟. 未知环境中移动机器人的自定位算法研究 [D]. 中南大学, 2005.

[2] 杨敏. 轮式移动机器人控制算法研究及其伺服系统设计 [D]. 南京航空航天大学, 2014.

[3] Bryant R E. Bryant, R.E.: Graph-Based Algorithms for Boolean Function Manipulation. IEEE Trans. Computers 35(8), 677-691[J]. 1986, C-35(8):677-691.

[4] Cimatti A, Griggio A, Schaafsma B J, et al. The MathSAT5 SMT Solver[J]. 2013.

[5] Flocchini P, Prencipe G, Santoro N. Distributed Computing by Oblivious Mobile Robots[J]. 2012, 3(2):1-185.

[6] Almeida A, Ramalho G, Santana H, et al. Recent Advances on Multi-agent Patrolling[C]// Advances in Artificial Intelligence - Sbia 2004, Brazilian Symposium on Artificial Intelligence, São Luis, Maranhão, Brazil, September 29 - October 1, 2004, Proceedings. DBLP, 2004:474-483.

[7] Suzuki I, Yamashita M. Distributed Anonymous Mobile Robots: Formation of Geometric Patterns. Sirocco'96, the, International Colloquium on Structural Information Communication Complexity, 1999:313-330.

[8] Suzuki I, Yamashita M. Erratum: Distributed anonymous mobile robots: formation of geometric patterns. Siam Journal on Computing, 1999, 28(4):1347-1363.

[9] Blin L, Milani A, Potop-Butucaru M, et al. Exclusive Perpetual Ring Exploration without Chirality. Lecture Notes in Computer Science, 2010, 6343:312–327.

[10] Devismes S, Petit F, Tixeuil S. Optimal Probabilistic Ring Exploration by Asynchronous Oblivious Robots[C]// International Conference on Structural Information and Communication Complexity. Springer-Verlag, 2009:195-208.

[11] Flocchini P, Ilcinkas D, Pelc A, et al. Computing Without Communicating: Ring Exploration by Asynchronous Oblivious Robots[M]// Principles of Distributed Systems. Springer Berlin Heidelberg, 2007:105-118.

[12] Flocchini P, Ilcinkas D, Pelc A, et al. Remembering without memory: Tree exploration by asynchronous oblivious robots[C]// International Colloquium on Structural Information and Communication Complexity. Springer Berlin Heidelberg, 2008:33-47.

[13] Baldoni R, Bonnet F, Milani A, et al. Brief Announcement: On the Solvability of Anonymous Partial Grids Exploration by Mobile Robots[C]// International Symposium on Distributed Computing. Springer-Verlag, 2008:496-497.

[14] Baldoni R, Bonnet F, Milani A, et al. On the Solvability of Anonymous Partial Grids Exploration by Mobile Robots. Principles of Distributed Systems. Springer Berlin Heidelberg, 2008:428-445.

[15] Devismes S, Lamani A, Petit F, et al. Optimal Grid Exploration by Asynchronous Oblivious Robots. Symposium on Self-Stabilizing Systems. Springer Berlin Heidelberg, 2012:64-76.

[16] Suzuki I, Yamashita M. Erratum: ”Distributed anonymous mobile robots: formation of geometric patterns” [SIAM J. Comput. 28 (1999), no. 4, 1347–1363 (electronic); MR1681010].[J]. Siam Journal on Computing, 2006, 36(4):279-280.

[17] Devismes S, Lamani A, Petit F, et al. Optimal grid exploration by asynchronous oblivious robots[J]. Lecture Notes in Computer Science, 2011, 7596:64-76.

[18] Potop-Butucaru M G, Potop-Butucaru M G. Brief announcement : discovering and assessing fine-grained metrics in robot networks protocols[C]// International Conference on Stabilization, Safety, and Security of Distributed Systems. Springer-Verlag, 2012:282-284.

[19] Halbwachs N, Caspi P, Raymond P, et al. The synchronous data flow programming language LUSTRE[J]. Proceedings of the IEEE, 2002, 79(9):1305-1320.

[20] Bérard B, Lafourcade P, Millet L, et al. Formal verification of mobile robot protocols[J]. Distributed Computing, 2016:1-29.

[21] Barnat J, Brim L, Havel V, et al. DiVinE 3.0 – An Explicit-State Model Checker for Multithreaded C C++ Programs[C]// International Conference on Computer Aided Verification. Springer-Verlag, 2013:863-868.

[22] Colange M, Baarir S, Kordon F, et al. Towards Distributed Software Model-Checking Using Decision Diagrams[M]// Computer Aided Verification. Springer Berlin Heidelberg, 2013:830-845.

[23] Doan H T T, Bonnet F, Ogata K. Model Checking of a Mobile Robots Perpetual Exploration Algorithm[C]// International Workshop on Structured Object-Oriented Formal Language and Method. Springer-Verlag New York, Inc. 2016:201-219.

[24] Eker S, Meseguer J, Sridharanarayanan A. The Maude LTL Model Checker[J]. Electronic Notes in Theoretical Computer Science, 2004, 71(05):162-187.

[25] Clavel M, Durán F, Eker S, et al. All About Maude - A High-Performance Logical Framework[J]. 2007, 4350.

[26] Goranko V. Logic in Computer Science : Modelling and Reasoning About Systems[J]. Journal of Logic Language Information, 2007, 16(1):117-120.

[27] Hagiya M, Mitchell J C. Proceedings of the International Conference on Theoretical Aspects of Computer Software[C]// International Conference on Theoretical Aspects of Computer Software. Springer-Verlag, 1994:257-271.

[28] Apt K R, Kozen D C. Limits for Automatic Verification of Finite-State Concurrent Systems.[J]. Information Processing Letters, 1986, 22(6):307-309.

[29] Clarke E M, Grumberg O, Jha S. Veryfying Parameterized Networks using Abstraction and Regular Languages[C]// International Conference on Concurrency Theory. Springer-Verlag, 1995:395-407.

[30] Bae K, Meseguer J. Model checking linear temporal logic of rewriting formulas under localized fairness[J]. Science of Computer Programming, 2015, 99:193-234.

[31] Cavada R, Cimatti A, Dorigatti M, et al. The nuXmv Symbolic Model Checker[C]// International Conference on Computer Aided Verification. Springer-Verlag New York, Inc. 2014:334-342.

[32] Clarke, E., Grumberg, O., Peled, D.: Model Checking. MIT Press, Cambridge, 2001.

[33] Bonnet F, Potop-Butucaru M, Tixeuil S. Asynchronous Gathering in Rings with 4 Robots[M]// Ad-hoc, Mobile, and Wireless Networks. Springer International Publishing, 2016.

[34] D’ Angelo G, Stefano G D, Navarra A, et al. Computing on Rings by Oblivious Robots: A Unified Approach for Different Tasks[J]. Algorithmica, 2015, 72(4):1055-1096.

[35] Huth M, Ryan M D. Logic in computer science - modelling and reasoning about systems (2. ed.)[J]. Ceskoslovenská Oftalmologie, 2004, 29(4):295-7.

[36] Kawamura A, Kobayashi Y. Fence Patrolling by Mobile Agents with Distinct Speeds[C]// International Symposium on Algorithms and Computation. Springer Berlin Heidelberg, 2012:598-608.

[37] Suzuki I, Yamashita M. Distributed Anonymous Mobile Robots: Formation of Geometric Patterns.[C]// Sirocco'96, the, International Colloquium on Structural Information Communication Complexity, Siena, Italy, June. DBLP, 1999:313-330.

[38] Clerentin A, Delahoche L, Brassart E, et al. Imprecision and Uncertainty Quantification for the Problem of Mobile Robot Localization[J]. Autonomous Robots, 2008, 24(3):267-283.

[39] D'Angelo G, Stefano G D, Navarra A. Gathering of Six Robots on Anonymous Symmetric Rings[M]// Structural Information and Communication Complexity. Springer Berlin Heidelberg, 2011:174-185.

[40] Kamei S, Lamani A, Ooshita F, et al. Gathering an Even Number of Robots in an Odd Ring without Global Multiplicity Detection[M]// Mathematical Foundations of Computer Science 2012. Springer Berlin Heidelberg, 2012:542-553.

[41] Flocchini P, Prencipe G, Santoro N, et al. Gathering of asynchronous robots with limited visibility □[J]. Theoretical Computer Science, 2005, 337(1):147-168.

[42] Millet L, Potop-Butucaru M, Sznajder N, et al. On the Synthesis of Mobile Robots Algorithms: The Case of Ring Gathering[J]. Computer Science, 2014, 8756:237-251.

[43] D'Angelo G, Stefano G D, Navarra A. How to gather asynchronous oblivious robots on anonymous rings[C]// International Conference on Distributed Computing. Springer-Verlag, 2012:326-340.

[44] D’ Angelo G, Stefano G D, Navarra A. Gathering on rings under the Look–Compute–Move model[J]. Distributed Computing, 2014, 27(4):255-285.

[45] D’Angelo G, Stefano G D, Navarra A. Gathering six oblivious robots on anonymous symmetric rings □, □□[J]. Journal of Discrete Algorithms, 2014, 26:16-27.

[46] D’ Angelo G, Navarra A, Nisse N. Gathering and Exclusive Searching on Rings under Minimal Assumptions[C]// International Conference on Distributed Computing and Networking. Springer Berlin Heidelberg, 2014:149-164.

[47] Stefano G D, Montanari P, Navarra A. About Ungatherability of Oblivious and Asynchronous Robots on Anonymous Rings[M]// Combinatorial Algorithms. Springer International Publishing, 2015.

[48] Izumi T, Izumi T, Kamei S, et al. Time-Optimal Gathering Algorithm of Mobile Robots with Local Weak Multiplicity Detection in Rings[J]. Ieice Transactions on Fundamentals of Electronics Communications Computer Sciences, 2013, E96.A(6):1072-1080.

[49] Kamei S, Lamani A, Ooshita F, et al. Asynchronous Mobile Robot Gathering from Symmetric Configurations without Global Multiplicity Detection[J]. 2011, 6796:150-161.

[50] Klasing R, Kosowski A, Navarra A. Taking Advantage of Symmetries: Gathering of many Asynchronous Oblivious Robots on a Ring[C]// Principles of Distributed Systems, International Conference, Opodis 2008, Luxor, Egypt, December 15-18, 2008. Proceedings. DBLP, 2010:446-462.

[51] Mostéfaoui A, Rajsbaum S, Raynal M, et al. Condition-based consensus solvability: a hierarchy of conditions and efficient protocols[J]. Distributed Computing, 2004, 17(1):1-20.

[52] Ooshita F, Tixeuil S. On the self-stabilization of mobile oblivious robots in uniform rings □[J]. Theoretical Computer Science, 2015, 568(C):84-96.

攻读硕士学位期间发表论文和科研情况

■ 已公开发表论文