2016 届研究生博士学位论文

分类号：_____ 学校代码：10269

密　级：_____ 学　　号：52121500013

華東師範大學

East China Normal University
博士学位论文

DOCTORAL DISSERTATION

基于协同验证与混成关系的混成系统形式化分析验证

院　　系：	计算机科学与软件工程学院
专　　业：	软件工程
研 究 方 向：	形式化方法
指 导 教 师：	朱惠彪　教授
学位申请人：	方徽星

2016 年 5 月 20 日

Dissertation for doctoral degree in 2016 University Code: 10269

Student ID: 52121500013

East China Normal University

Formal Analysis and Verification of Hybrid Systems via Co-Verification and Hybrid Relations

Department:	**School of Computer Science and Software Engineering**
Major:	**Software Engineering**
Research direction:	**Formal Methods**
Supervisor:	**Prof. ZHU Huibiao**
Candidate:	**FANG Huixing**

May, 2016

华东师范大学学位论文原创性声明

郑重声明：本人呈交的学位论文《基于协同验证与混成关系的混成系统形式化分析验证》，是在华东师范大学攻读硕士/博士（请勾选）学位期间，在导师的指导下进行的研究工作及取得的研究成果。除文中已经注明引用的内容外，本论文不包含其他个人已经发表或撰写过的研究成果。对本文的研究做出重要贡献的个人和集体，均已在文中作了明确说明并表示谢意。

作者签名：_____ **日期：** 年 月 日

华东师范大学学位论文著作权使用声明

《基于协同验证与混成关系的混成系统形式化分析验证》系本人在华东师范大学攻读学位期间在导师指导下完成的硕士/博士（请勾选）学位论文，本论文的研究成果归华东师范大学所有。本人同意华东师范大学根据相关规定保留和使用此学位论文，并向主管部门和相关机构如国家图书馆、中信所和“知网”送交学位论文的印刷版和电子版；允许学位论文进入华东师范大学图书馆及数据库被查阅、借阅；同意学校将学位论文加入全国博士、硕士学位论文共建单位数据库进行检索，将学位论文的标题和摘要汇编出版，采用影印、缩印或者其它方式合理复制学位论文。

本学位论文属于（请勾选）

（）1. 经华东师范大学相关部门审查核定的“内部”或“涉密”学位论文 *，于 年 月 日解密，解密后适用上述授权。

（）2. 不保密，适用上述授权。

导师签名：_____ **本人签名：**_____

年 月 日

*“涉密”学位论文应是已经华东师范大学学位评定委员会办公室或保密委员会审定过的学位论文（需附获批的《华东师范大学研究生申请学位论文“涉密”审批表》方为有效），未经上述部门审定的学位论文均为公开学位论文。此声明栏不填写的，默认为公开学位论文，均适用上述授权。

方徽星 博士学位论文答辩委员会成员名单

姓名	职称	单位	备注
缪淮扣	教授	上海大学	主席
赵建军	教授	上海交通大学	
阚海斌	教授	复旦大学	
陈仪香	教授	华东师范大学	
曾振柄	教授	上海大学	

摘　　要

随着软件、集成电路、网络及通信技术的飞速发展，人类生存的自然物理世界与计算机科学领域之间的联系和相互影响愈加的紧密和重要。混成系统，作为一个融合并体现自然物理世界的连续特性及计算的离散特性的基本模型，在最近三十年间，一直受到来自数学、控制、计算机甚至生物化学等众多学科无数专家学者的研究和关注。目前，形式化方法在混成系统的众多研究理论和方法中占有重要的一席。

本文，我们主要研究混成系统的建模、仿真和形式化验证。现有的形式化验证技术无法克服状态空间爆炸问题，当混成系统的规模较大时，验证工具往往无法给出验证结果，从而无法判定模型的正确性和安全性，这使得混成系统的形式化验证理论和方法对工业界的影响相当有限。为了融合现有的仿真和验证技术，我们提出协同验证方法，使仿真与验证的方法和技术相结合，充分发挥仿真技术在大规模复杂系统分析上的优势，基于仿真结果的反馈，对验证模型进行抽象，同时利用验证技术在模型检验方面的优势保障模型的正确性和安全性。另一方面，基于验证结果的反馈，可实现仿真模型的优化和更有效的仿真。如此，通过验证与仿真的相互反馈，逐步实现高效、可信的混成系统设计与开发。

此外，混成系统建模语言对于实现形式化方法在工业界的广泛应用也是极其重要的。一个结构良好、使用方便的建模语言往往更易于被工业界的设计和开发人员所接受，从而得以推广和应用。华东师范大学何积丰院士最新提出的混成系统建模语言（在本文中，我们使用 HML（Hybrid Modeling Language）表示该建模语言），在语法结构上具有设计良好、语法简单等特性。此外，该建模语言还具有很强的可扩展性，非常适合系统设计和开发人员在实际工作中使用。为了支持 HML 语言的形式化验证，我们基于混成关系理论给出相应的霍尔逻辑推理规则，并将 HML 模型转换为 SMT（Satisfiability Modulo Theories）公式进行约束求解，以实现混成系统的仿真和自动化验证。

下面，我们从三个方面对本论文的主要贡献进行总结：

- **方法贡献**

  基于对混成系统仿真和验证技术的优势及劣势分析，我们提出反馈演进的混成系统模型设计和分析方法。通过仿真技术，我们可以实现大规模复杂系统的分析，但是仿真无法完全覆盖系统所有可能的状态和执行路径，所

以基于仿真的系统设计和开发方法，容易产生遗漏和失误，从而为工业生产和公共安全带来严重的安全隐患。基于形式化验证，我们可以对系统所有可能的状态进行分析和检查，从而能够保证系统的正确性和安全性。不过，形式化验证不能有效地对大规模复杂系统进行分析。因此，整合仿真与验证来对混成系统进行分析是很有必要的。我们提出的反馈演进方法充分利用仿真与验证各自的优势，使二者的分析验证能力得以互补，通过分析验证结果互反馈的方式逐步实现大规模复杂混成系统的可信设计与开发。

- **理论贡献**

  混成关系理论将混成系统的行为表示为三种系统执行状态下的条件约束，基于离散和连续动态行为的特性，分别对动态行为执行前、执行过程中以及执行结束时所满足的条件进行约定。受混成关系基本思想的启发，我们将混成关系与霍尔逻辑相结合，给出一种新的霍尔逻辑推理规则，以支持混成系统建模语言 HML 的推理验证。与以往的推理系统不同的是，在微分方程的处理上，我们的推理规则并不依赖于特定的方程或不变式求解技术。在我们的推理系统中，推理规则实现的是一个验证的框架，当我们需要对具体的微分方程进行处理时可以自由地选择数学领域成熟的技术来分析。逻辑规则与数学求解的分离可以极大地简化推理规则的复杂度，并支持规则的可扩展性，使我们更好地专注于模型结构和系统行为验证过程的逻辑部分。

- **实践贡献**

  本文包含丰富的混成系统案例分析，如：地铁屏蔽门控制、列车防碰撞控制、水箱水位控制、倒立摆控制等等。这些案例分析是对我们提出的理论和方法的检验，能够体现所提出的理论和方法的可行性与有效性。此外，我们还基于混成关系理论的基本思想，提出将基于 HML 语言建立的混成系统模型转换为 SMT 公式的方法，并整合 SMT 求解器 dReal 的约束求解技术，开发了相应的原型工具，实现了 HML 模型仿真和验证的自动化。该工作在混成系统开发和验证方面为我们的下一步研究打下了坚实的基础。

关键词: 混成系统，形式化方法，仿真，验证，霍尔逻辑，混成关系，可满足性模理论

ABSTRACT

With the rapid development of technologies on software engineering, integrated circuit, network, and communication, the connections and interactions increasingly become much more intimate and significant between human beings and the natural world. Hybrid system, as a fundamental model combining and revealing continuous feature of natural world and discrete feature for computation, has been studied by numerous researchers in the areas ranged over mathematics, automatic control, computer science, and even biological chemistry in the last 30 years. Nowadays formal methods play an important role among numerous theories and methods on hybrid systems.

In this thesis, we focus on modeling, simulation, and formal verification of hybrid systems. Current approaches available for verification of hybrid systems are unable to overcome the state-explosion problem. Usually, software tools for verification cannot give proper conclusions when we consider large-scale systems. Therefore, the correctness and safety for the corresponding models are undecidable, the influences of theories and methods proposed from the area of hybrid systems are rather limited for industrial applications. In order to integrate existing simulation and verification techniques, we propose a co-verification approach for hybrid systems. The advantage of simulation is the capability of analyzing large-scale systems. Based on the feedback of simulation we can abstract models of hybrid systems for verification, then utilize the advantages of verification on the checking of abstract models to guarantee correctness and safety for hybrid systems. Meanwhile, based on the feedback of verification, we can optimize the model for simulation and make simulation be more effective. Therefore, an efficient and reliable procedure for the design and development of hybrid systems can be developed based on mutual feedbacks between formal verification and simulation.

In addition, modeling languages for hybrid systems are also important to the widespread application of formal methods in industry. A well-structured and convenient modeling language is easier to be accepted by designers or developers, then allowed to be applied broadly in industry. The hybrid modeling language

HML which was recently proposed by Prof. HE Jifeng, is well-designed and its syntax is easier to be implemented and used for modeling hybrid systems. Moreover, HML has good extensibility that is suitable for designers and developers in practical applications. In this thesis, we present inference rules in the style of Hoare triples for the formal verification of HML models. We also translate HML models into SMT (Satisfiability Modulo Theories) formulas that can be analyzed based on constraint-solving techniques, which brings about a prototype tool for the simulation and automated verification for hybrid systems.

The contributions of this thesis can be summarized as follows:

- **Methodological Contribution**

  We present a feedback-advancement method for design and analysis of hybrid systems based on the analysis for advantages and disadvantages of simulation and formal verification techniques. We are able to analyze large-scale complex systems through simulation, but simulation is failed to cover all possible states or execution paths of a system. As a result, simulation is error-prone and always keep mistakes or bugs in the model, which brings potential risks into industrial activities and poses threats for public safety. In contrast, formal verification is capable of checking all possible states of a system, thus it can be used to guarantee the correctness and safety. Unfortunately, formal verification is unable to analyze large-scale complex systems effectively. Therefore, it is necessary to combine simulation and formal verification for the analysis of hybrid systems. Our feedback-advancement method takes the advantages of simulation and verification adequately, and also achieves complementation of these two techniques. As a result, the method is capable of achieving reliable design and development for large-scale complex systems based on feedbacks between simulation and verification.

- **Theoretic Contribution**

  In hybrid relation theory, the behavior of a hybrid system is expressed with constraints for three execution statuses: the constraint before the behavior starts, the constraint during its execution, and the constraint when the be-

havior terminates, respectively. Inspired by the basic idea of hybrid relation theory, we integrate Hoare logic with hybrid relations, which results in a set of new inference rules for HML models. The new inference rules are independent of specific solving techniques for equations or invariants with respect to previous works. In our inference system, the inference rules constitute a framework for verification. It is free to choose established techniques when we have to deal with differential equations during the verification. The separation of mathematical solving techniques from inference rules simplifies the complexity of the rules without losing the extendibility of the framework, and we can just focus on the structures of models and the logical parts of the verification for the system behaviors.

- **Practical Contribution**

  This thesis contains plentiful case studies on hybrid systems, for instance, platform screen doors system, collision-avoidance control, and inverted pendulum. These case studies are used to examine theories and methods presented in this thesis, meanwhile exemplify the feasibility and effectiveness. Moreover, we propose an approach for the translation of HML models to SMT formulas following the basic ideas of hybrid relation theory. Also, we develop a prototype tool for simulation and automated verification of HML models based on an SMT solver dReal, which builds a solid foundation for our future work on the development and verification of hybrid systems.

Key Words: *Hybrid Systems, Formal Methods, Simulation, Verification, Hoare Logic, Hybrid Relation, SMT*

目录

插图

表格

第一章　绪论

混成系统是对现实中既含有离散动作又包含连续演化行为的一类动态系统进行规范化的基本数学模型。混成系统的研究涉及经典力学、控制、嵌入式软件及网络通信等相关学科领域。本章主要阐述混成系统相关研究领域的发展历程，以及在系统建模语言和分析验证等方面的研究进展，最后我们对本论文的主要研究内容和贡献进行了说明。

1.1　引言

科学研究本着“万有之本，必涵其因”的探索精神，在人类科学发展历程中，对客观世界不断地进行着肯定与否定的判定过程。从古至今，人们通过观察和记录，了解气候变化，探索天体运动规律，逐渐掌握了物体运动的基本规律。到了 19 世纪，经典力学成为物理学中日臻成熟的学科。经典力学研究的对象是低速机械运动的宏观物体而非微观粒子，物体的运动速度相对于光速来说属于低速。虽然，经典力学在接近光速的高速系统中，以及在强引力场或者微观尺度（如：微观粒子）系统中并不适用，但是，经典力学在现实中的应用仍旧十分广泛。日常生活中，人们可以直接观察和感受到的，如：行星的运动以及火车、飞机等交通工具的运动甚至有机分子的运动，都属于经典力学的应用范围。而且，相比于量子力学和相对论力学的复杂深奥，经典力学理论是简单的，且容易被人们接受和应用，在日常生活中，经典力学的影响是无处不在的。

19 世纪末 20 世纪初，随着法国数学家庞加莱（Jules Henri Poincaré）对经

典力学中三体问题[1]的研究 [118, 72]，以及俄罗斯数学家李雅普诺夫（Aleksandr Mikhailovich Lyapunov）对微分方程稳定性的探索而逐渐兴起了对动态系统的研究。动态系统又称动力学系统，在这类系统中，系统状态变量是一个关于时间的函数，状态随时间变化，或者系统状态遵循确定性规则且随时间变化。20 世纪 50 年代中期，在美苏争霸的格局之下空间技术迅速兴起，在此背景下，现代控制理论得以蓬勃发展，同时也促进了动态系统领域的研究，使动态系统的应用由经典力学推广到控制领域。

随后，计算机和网络技术的发展，尤其是 20 世纪 70 年代嵌入式系统的出现，使得基于微处理器的控制系统更易于设计和实现，并具有更好的性能。20 世纪 90 年代，在分布式控制、柔性制造、移动通信等需求驱动之下，嵌入式系统中软件的重要性日趋明显，同时系统实时性要求的提高使得实时多任务操作系统逐渐成为嵌入式系统的主流软件平台。随着现代控制系统中自动化要求的提高，嵌入式系统应用越来越广泛，在汽车电子、智能机器人、航天器、智能交通、空间探测器、化工生产、核工业等控制领域，嵌入式系统可用于实现可靠安全和高效实时的控制。

更进一步，为了实现对外部物理对象和环境的实时感知、可靠性信息传输、智能信息处理、精确动态控制与协调等目标，人们又提出了信息物理融合系统的概念。信息物理融合系统是实现计算、通信与控制领域的技术与资源深度融合的下一代智能系统 [90, 146]。当前，信息物理融合系统的应用已十分普及，从日常生活中的智能家居、可穿戴设备、体感游戏等，到多智体机器人系统、环境监测与控制，甚至智能电网、空中交通控制系统、战地机器人、国际空间站等都有信息物理融合系统的应用。随着系统复杂程度的提高，如何保障信息物理融合系统尤其是安全攸关系统的正确性与安全性是一个至关重要且极具挑战的问题。因此，使用严谨的数学方法对系统进行建模和分析验证是十分必要的。在系统模型方面，混成系统可作为信息物理融合系统的数学模型，其不仅能够对连续的物理

[1]指研究三个可视为质点的天体在万有引力作用下的运动规律，其中每个天体的质量、初始时刻天体在空间中所处位置以及运动速度均为任意的

行为以及离散的控制逻辑进行建模，还能表示连续与离散行为之间的交互，此外，基于事件或信号的抽象表达，混成系统还能对多个组件之间的协调互动行为进行建模。在系统分析方面，在工业界应用较多的则是仿真与测试技术。然而，仿真与测试无法对系统的所有可能的状态进行完全覆盖。因此，系统中往往存在着不易发现的严重缺陷。一旦这些缺陷在实际应用中暴露，则往往会导致巨大的灾难和损失[2]。因此，使用基于精确数学语义、严谨证明系统、模型验证方法的形式化方法 [139] 来对系统进行设计和分析是完全有必要的。形式化方法在一定程度上可以克服仿真与测试的不足，为系统的设计和开发，提供正确性和安全性的保障。

1.2　相关工作

混成系统是一种动态系统，其同时具有连续和离散的动态行为特征，在经典的混成系统模型中，连续行为由微分方程来表示，离散行为则由差分方程来表示。在工业应用中，混成系统可以作为嵌入式系统设计的数学模型。混成系统在现实中无处不在，如：汽车电子、高速列车、轨道交通、航空航天以及工业生产等等安全攸关的领域，这些应用领域中的控制系统都可以由混成系统的理论和方法来设计开发，并进行相关的模型仿真和形式化分析验证，从而提高系统的可信度，减少或辅助消除控制软件中的缺陷，降低系统的研发成本，缩短研发周期，实现高可信的系统设计开发实现，最终产生显著的经济和社会效益。下面，我们从建模和分析验证这两个方面阐述混成系统的相关研究工作。

1.2.1　混成系统的建模

法国 Oded Maler 等在 1992 年提出相位迁移系统（Phase Transition Systems [95]），该研究是对离散迁移系统的扩展。混成系统的行为由一系列的相位构成，

[2]最著名的例子莫过于发生在 1996 年 6 月 4 日的 Ariane 5 火箭失败事件，其初次航行即在升空后一分钟内，因内部软件发生浮点转换失败而偏离航道，最终引发爆炸，致使损失数亿美元 [87]

其中，连续相位消耗时间并允许变量的连续变化，离散相位由有限步的离散迁移构成，每个迁移有可能导致变量的值非连续变化。

1993 年，Rejeev Alur 等提出混成自动机（Hybrid Automata[7]）建模语言。混成自动机是有限状态机的一种扩展语言，其中连续行为可以由微分方程进行约束，离散行为则由模态迁移中的差分方程表达。从图形语言的角度来看，在自动机所对应的图形中，顶点表示连续行为，边表示离散行为。为了支持对混成系统描述的分解，Lynch 等提出混成输入/输出自动机（HIOA：Hybrid I/O Automata，[93]）建模框架。模型分解的关键是，该建模框架对于一个混成输入/输出自动机还定义了外部行为的概念，表示该自动机与环境之间的离散和连续交互行为。

在基于进程代数的混成系统建模语言方面，主要有 Cuijpers 等提出的 HyPA[39] 语言。HyPA 是进程代数语言 ACP[15] 的扩展，可以支持连续行为的表达。该建模语言有两个特殊操作符，其一为中断操作符，表示进程中断，可用于异常处理和模态切换等行为的表达，其二为封装操作符，用于将进程中特定的离散动作阻塞而对外部环境隐藏，如在进程并发组合时，可用于表示多个离散动作的同步执行。

在 CSP（Communicating Sequential Processes[70]）基础上，何积丰院士以及周巢尘院士提出 HCSP（Hybrid CSP[74, 31]）语言，其中连续行为用微分方程来表示，在语法上，还引入了超时和中断相关的语句。进程间的通信方式沿用了 CSP 的发送/接收机制。

在工业界使用较多的混成系统建模语言有 Simulink/Stateflow[97, 98]，其特点是具有丰富的原子组件库，支持非线性物理行为和层次化模型，并且能够在模型中使用 Matlab 函数实现复杂的控制逻辑。另一个在工业界使用较多的则是 Modelica[100]，Modelica 提供了一种面向对象的建模方法。该语言可以实现组件重用，能够适用于规模较大的系统建模。然而，其语义中存有歧义，导致对该语言的解释往往含有不一致之处，如事件是同步还是异步的问题 [107]，还有因不同的仿真引擎选择的执行顺序不同而使得仿真结果不一致问题 [19] 等等。

此外，还有 Mosterman 等提出的混成键合图的建模方法 [101]，即在键合图（Bond Graph，键合图建模方法认为系统的能量是守恒的，系统的动态行为则反映为功率流（Power Flow）的重新分布与调整 [114]）中引入离散行为，系统中能量的变化可对应为离散事件的发生。

另一个用于仿真的建模语言 CHARON[9] 是由 Rajeev Alur 等提出的，该语言支持模块化和层次化建模，并可使用外部定义的 Java 函数实现复杂的离散控制逻辑，对于连续行为，可以使用微分和代数约束以及不变式约束来指定物理对象的连续演化规则和变量的取值范围。

除了上述研究工作外，还有 Hybrid Petri Net[40]、Hybrid Program[117]、Hybrid Action System[124] 等混成系统的建模语言和方法，在此不再详述。

1.2.2 混成系统的分析与验证

混成系统的形式化模型建立之后，分析并判定模型是否满足需求是一项极其重要的工作。本小节将从模型仿真和形式化验证这两个主要方面展开叙述。

首先，在仿真方面，应用最为广泛的是由美国 MathWorks[3]公司出品的数学软件 Matlab，它将数据可视化、科学计算以及动态系统的建模和仿真等诸多强大功能集成在一个易于研究者和工业界开发人员使用的可视化图形界面中。Matlab 的配套软件包 Simulink 提供了一个可用于混成系统模拟、嵌入式系统开发等方面的可视化开发环境。此外，在 Matlab 中还有一个图形化建模工具 Stateflow，该工具基于有限状态机和流程图，可用于构建包含复杂逻辑控制和状态迁移的系统，并可对非线性行为以及时间相关的物理行为进行建模和仿真，Stateflow 可以作为 Simulink 模型的子系统直接嵌入到 Simulink 模型中，从而能够使用 Simulink 对复杂和规模较大的混成系统进行仿真。Ricardo G. Sanfelice 等提出的 HyEQ 工具箱 [125] 由一系列 Matlab/Simulink 脚本组成，通过数值求解混成系统中混成方程（Hybrid Equation，即由微分和差分方程组成的一组方

[3] www.mathworks.com

程代表混成系统中连续和离散的动态行为）的迹并以图形方式展示仿真结果。HyEQ 可以计算芝诺（Zeno，即在有限时间内发生无限多个动作的现象）混成方程的迹，以及在同一时刻点具有多个离散跳变的系统行为的迹。同时，HyEQ 也支持事件和过零检测，以及多个混成系统之间的交互行为的仿真。

Rajeev Alur 等提出的 CHARON[4]工具包 [8] 是混成系统建模语言 CHARON 的仿真工具。该工具支持文本和图形方式建模，支持从模型自动生成 Java 代码，仿真器可对混成系统模型中所声明的断言性质进行检验，若发现断言被违反，则仿真器将停止仿真，并产生相应的仿真路径以便分析和调试。

参考南京大学卜磊等所撰写的研究综述 [145]，在形式化验证方面，我们将从模型检验和定理证明这两个主要的方面回顾与混成系统相关的研究工作。

法国 VERIMAG 实验室的 Oded Maler 等提出将混成系统的可达状态集合以凸多面体（convex polyhedra）结构进行逼近，并实现了验证工具 d/dt[13, 12]，可以对分段线性和非线性混成系统进行可达性分析。但是，该工具的终止条件依赖于可达状态的下逼近（under-approximation），同时也容易产生因有限精度运算而导致的各种缺陷。

基于多面体的可达状态逼近，在保证精度的前提下 Goran Frehse 将多面体的约束和参数进行限定，从而能够处理较为复杂的与多面体相关的运算。对于仿射动态行为，Goran Frehse 提出可达状态的动态逼近和状态空间划分等方法，并在验证工具 PHAVer[48] 中实现，这也是我们对混成自动机的形式化验证所采用的主要验证工具。

另一种逼近方法使用椭圆体（ellipsoidal）[23, 85] 来对可达状态集进行逼近。由于椭圆体容易受自身形状的限制，当我们在某个维度试图实现紧逼近时，其他维度上的逼近将会变得不够精确。因此，找到一种合适的逼近策略而不会导致上述问题发生将是很关键的而且是极具挑战的。

全对称多胞形（zonotope）是多面体的一个子类，该数学对象在闵可夫斯基和（Minkowski Sum：可达状态集合计算过程中的一个重要的运算）的计算上要

[4]http://rtg.cis.upenn.edu/mobies/charon/

比普通的多面体效率高。但是，在实际中往往会出现构成全对称多胞形的矢量（generator）数目过大的问题，而且 zonotope 也存在类似椭圆体的在几何形状上的限制问题。相关研究有 [86, 54, 55, 6]，其中，[54] 通过构造全对称多胞形的区间壳（interval hull）将全对称多胞形的构造矢量减少，从而提高算法的有效性。此外，还有交运算不封闭问题，即两个全对称多胞形的交集不是全对称多胞形，这样就需要在可达状态与离散迁移条件进行交运算之后重新对可达状态进行逼近。在 [6] 中，Matthias Althoff 等提出使用半空间（halfspace）表示方法对全对称多胞形进行逼近，这样在做交集的时候，可以在半空间结构上进行，避免了交运算的不封闭问题。

Colas Le Guernic 等基于紧凸集（compact convex set）的支撑函数（support function）对混成系统进行可达性分析 [60, 61]，该方法不局限于特定的可达状态集的表达方式，只要系统的不变式、迁移条件和可达状态等可以表示为紧凸集的形式就可以利用紧凸集的支撑函数进行可达状态的计算。基于该方法，Goran Frehse 等实现了相应的验证工具 SpaceEx [49]，可以支持 100 个变量的混成系统的形式化验证。

对于非线性混成系统，验证工具 flow*[34] 基于泰勒模型（Taylor model）的流管（flowpipe：从初始状态集出发，连续动态行为在给定时间区间内所有可达状态的集合）构造 [33] 技术，可以支持多项式非线性混成系统的验证。在离散迁移的处理上，[33] 提出域收缩（domain contraction）的方法，即将泰勒模型中与离散迁移条件相交为空的部分消去，以降低泰勒模型域的大小。另外，上述研究提出，先把泰勒模型转换为多面体结构，然后利用现有技术进行离散迁移的处理。可以看出，在泰勒模型中，离散迁移的处理是一个比较复杂而且直接影响混成系统验证效率的操作。

在非线性混成系统验证方面，南京大学卜磊等提出凸性混成自动机，并基于凸规划提出面向路径可达性判定以及有界可达性判定方法，可以对规模较大的凸性系统进行判定 [27]。

在定理证明方面，主要有美国卡耐基梅隆大学（Carnegie Mellon University）

的 André Platzer 等提出的微分动态逻辑推理系统 [117]。其特点是，以微分不变式（differential invariant）为基础对连续行为进行推理，避免了微分方程的复杂求解过程。相关的推理工具为 KeYmaera [116]，可以实现混成系统的自动定理证明。

在基于 SMT 约束求解来对混成系统进行形式化分析方面主要有美国卡耐基梅隆大学 Edmund M. Clarke 团队所做的研究工作。首先，对于混成系统中的非线性动态行为，需要解决非线性实运算的判定问题，即判断非线性约束是否可满足。

美国卡耐基梅隆大学 Edmund M. Clarke 教授的研究团队提出一套全新的方法来应对该问题 [52]。其方法整合区间约束传播技术（ICP：Interval Constraint Propagation）及支持线性实运算（LRA：Linear Real Arithmetic）的 SMT 求解器来对非线性实运算问题进行判定。该方法使用 ICP 技术搜寻非线性约束的近似区间解，然后，利用 LRA 求解器检查所得区间解是否正确，当所得的区间解违反某些线性约束时，利用这些约束逐步精炼 ICP 的搜寻空间，如此迭代地进行解空间的求解。当 ICP 对于新增的线性约束得出不可满足的结果时，或者所得区间解成功地被 LRA 求解器所检验时，该判定过程即可结束。如此，可将线性与非线性约束求解进行分离，使得解的正确性得以保证而且实现误差可控。

对于解的误差控制，该工作提出 δ 完备性定理，即对于给定的 ICP 误差上界，如果一个待判定的约束公式是可满足的，则验证过程会返回 SAT；如果对该公式施以 δ 强化扰动后得到的新公式是不可满足的，则验证过程返回 UNSAT。关于 δ 可判定性和完备性方面的研究，Edmund M. Clarke 等在 [51] 中对于实数域上的可计算性函数，包括满足利普希茨连续条件的（Lipschitz-continuous）微分方程及相应的有界一阶逻辑公式进行了严谨的分析。

在 [51] 中，给定一个微小的扰动 δ，研究人员考虑一阶逻辑语句的集合，该集合中的语句具有保持真值不变的特性。对于集合中的任一语句，在量词有界情况下，判定其为真或者其在 δ 扰动下为假，这样的判定问题称为 δ 判定问题。Edmund M. Clarke 等的研究表明，δ 判定问题的算法复杂性属于合理的复杂性

类别。如，逻辑公式中涉及的算术运算有指数（exp）和三角函数（sin）时，δ 判定问题对于有界 Σ-1 类语句的判定复杂性为 NP 完全的。而在一般情况下，这样的语句是不可判定的。另外，对于任意的量词有界语句，当其含有利普希茨连续的微分方程时，相应的 δ 判定问题为 PSPACE 完全的。上述复杂性并没有比判定带量词的布尔公式来得更复杂，这个结论对于非线性混成系统的有界验证是十分有利的。该研究指出，存在 δ 判定问题的算法可以在合理的时间和空间复杂度下，对有界的，包含可计算函数（Type-2，如：三角函数，多项式，指数函数，对数函数，利普希茨连续的微分方程等）的一阶逻辑语句进行判定。对于实时系统和混成系统，其模型可以使用一阶逻辑公式来表示，因此，这项工作的研究可以应用于混成系统等动态系统的有界模型检验过程中。在此基础上，Clarke 的研究团队，提出了 δ 完备的判定过程 [50] 解决涉及实数的 SMT 求解问题，并开发了相应的 SMT 求解器 dReal [53]。

Clarke 等提出的判定过程解决：给定一个 SMT 问题 φ 以及正有理数 δ，一个 δ 完备的判定过程会对 φ 做出判断，即 φ 是不可满足的，或者 φ 的 δ 弱化问题是可满足的。判定过程的存在性和复杂度在 [51] 也中得到确认和研究，该判定过程在 SMT 求解器 dReal 中得以实现，其中主要整合了 OpenSMT [26] 和 RealPaver [58] 分别用于 DPLL(T) [106, 84] 和区间约束传播算法的实现，工具的输入为 SMT-LIB 2.0 [18] 的公式及其扩展。对于非线性约束，dReal 支持可计算性非线性函数，如：sin, tan, arcsin, arctan, exp, log, pow, sinh 等。在应用方面，目前主要有心肌细胞的动作电位混成系统模型的参数分析 (parameter synthesis)[92]，并有效地研究了心脏紊乱相关的一些关键参数的取值范围。此外，Clarke 等还提出一个针对病人特异性的计算框架，用于确定雄激素消融治疗策略以延迟癌症的潜在复发 [91]。

Clarke 团队的研究工作基于有界模型检验的基本思想，利用变量微小扰动使得对于涉及实数的非线性约束求解问题的判定过程复杂性趋于合理化。在技术实现上，使用目前应用广泛的 SMT 技术，使得研究可扩展性和广泛性得以保证，为今后的研究工作建立了坚实的理论基础。但该技术在具体实例应用过程中对于

大规模模型的分析还有待改进，其算法的时间和空间复杂度有待优化，这可以成为未来研究的主要方向。

1.3 主要贡献

本文主要从混成系统的建模、仿真和形式化验证这三个方面展开研究。众所周知，当前形式化验证面临的问题是状态空间爆炸问题，该问题目前还没有彻底解决。当我们要验证的系统规模较大时，形式化分析工具无法在可接受的时间内给出验证结果，甚至验证过程无法终止，因而不能判定模型的正确性和安全性，这极大地限制了混成系统的验证理论和方法在工业界的广泛应用。

当前，在工业界应用较多的是仿真技术。仿真基于测试的基本思想，是对系统部分的执行路径进行模拟和测试。仿真可以处理大规模复杂系统的分析，并可以提供较丰富的建模手段以及系统的执行路径和状态的可视化展现功能。为了支持规模较大的混成系统的分析，我们提出协同验证方法，使仿真与验证的方法和技术相结合，充分发挥仿真技术在大规模复杂系统分析上的优势，以及形式化验证在系统正确性和安全性保障方面的优势，通过仿真与验证的互反馈，逐步实现高效、可信的混成系统设计与开发。

在混成系统建模语言方面，华东师范大学何积丰院士最新提出了混成系统建模语言 HML，该语言在语法结构上具有设计良好、语法简单、可扩展性强等特性，非常适合系统设计和开发人员在实际工作中使用。为了支持 HML 语言的形式化验证，我们基于混成关系理论提出了霍尔逻辑推理规则，并对混成系统的经典案例进行了分析。另外，我们还将 HML 模型转换为 SMT 公式，整合开源的求解器 dReal 的约束求解功能，实现了混成系统的仿真和自动化验证，并开发了相关的原型工具。

本论文的主要贡献可以总结如下：

一、反馈演进的方法贡献

我们提出以仿真与验证互反馈为基础使混成系统模型逐步演进的协同验证

方法。仿真技术的优势在于，仿真可以有效地支持大规模复杂系统的分析。仿真技术的劣势在于，仿真无法检验系统所有可能的状态和执行路径，因此容易产生遗漏和失误，可能为工业生产和公共安全带来严重的安全隐患。所以，模型的正确性和安全性无法通过仿真来保证。形式化验证的优势在于，验证可以对系统所有可能的状态和执行路径进行分析和检查，在一定程度上保证系统的正确性和安全性。形式化验证的劣势在于，验证不能有效地对大规模复杂系统进行分析。因此，我们对仿真与验证技术进行整合，提出反馈演进方法，充分利用仿真与验证各自的优势，使二者的分析验证能力得以互补。通过在仿真中建立与实际应用较接近的模型，而在验证中构建较抽象的模型，从而实现二者在模型分析和验证方面的优势互补，以分析结果的反馈逐步对模型进行优化，提高混成系统设计和开发的效率并最终实现对模型的正确性和安全性的保障。

二、混成系统验证的理论贡献

由何积丰院士提出的混成关系是对混成系统行为的一种逻辑刻画。在混成关系中，系统的行为由三个部分组成，分别对应为系统三种执行状态下的约束：系统行为执行前、执行过程中以及执行结束时所满足的条件约束。基于混成关系的基本思想，我们将混成关系与霍尔逻辑相结合，提出了一种新的霍尔逻辑推理规则，以支持混成系统建模语言 HML 的推理验证。该推理系统与以往的推理系统不同的是，在微分方程的处理上，我们的推理规则并不绑定特定的方程或不变式求解技术。我们实现的是一个验证的框架，当在推理过程中需要对具体的微分方程进行处理时，我们可以自由地选择数学领域成熟的技术来分析。逻辑规则与数学求解的分离可以极大地简化推理规则的复杂度，并支持规则的可扩展性，使我们能够更好地专注于模型结构和系统行为的推理验证。

三、混成系统开发的实践贡献

在实践方面，我们提供了比较丰富的混成系统案例分析，主要有地铁屏蔽门控制、列车防碰撞控制、水箱水位控制、倒立摆控制等。这些案例分析一方面是对我们提出的理论和方法的检验，另一方面，在一定程度上也可以体现我们所提出的理论和方法的可行性与有效性。此外，我们还开发了 HML 语言的仿真和验

证工具，该工具基于混成关系理论的基本思想，可以自动地把 HML 模型转换为 SMT 公式，并以 SMT 求解器 dReal 的约束求解技术为基础可以实现非线性混成系统的仿真和验证，HML 模型的仿真结果可以在原型工具中以图形化的方式进行展现，因此，该原型工具能够辅助开发人员进行混成系统的设计和开发。

1.4 论文结构

本文的主要研究工作按照以下的论文结构进行阐述：

第二章主要介绍了混成系统的基本概念、程序统一理论的基本思想、经典霍尔逻辑的推理规则以及可满足性模理论的基本知识等。

第三章提出混成系统协同验证的方法，并将该方法应用于地铁屏蔽门控制系统和列车防碰撞系统中。在这一章中，我们采用的仿真工具为在工业界应用十分广泛的 Simulink/Stateflow，形式化验证工具选择的是 SpaceEx/PHAVer，通过详尽的案例分析和丰富的实验结果展示，表明了在混成系统开发中仿真与验证技术相协同的可行性和必要性。

第四章基于混成关系理论，提出混成系统建模语言 HML 的霍尔逻辑推理规则，并应用于经典的水箱水位控制和列车防碰撞控制系统中。我们解释了建模语言 HML 的语法和并发程序交互的信号机制，给出了适用于 HML 语言模型的断言设计，并详细解释了基于混成关系的霍尔逻辑推理规则。

第五章主要基于混成关系的基本思想，将 HML 语言模型转换到 SMT 公式，并结合 SMT 求解器 dReal 的约束求解技术，开发了可用于 HML 语言模型仿真和验证的原型工具。此外，我们还详细阐述了倒立摆模型的案例分析，提出倒立摆的连续控制和离散控制方式，原型工具的可行性得以检验。

第六章主要介绍了原型工具的基本框架和使用说明，并展示了使用 HML 语言建立混成系统模型并分析的过程。

第七章是对我们在本文中介绍的研究工作的总结以及对未来相关工作的展望。

第二章　基本概念及准备知识

2.1　混成系统概念

混成系统是一种同时包含连续和离散行为的动态系统。混成系统的连续行为一般使用微分方程进行约束，离散跳变则使用差分方程来表示。混成系统主要包含如下几个基本的概念：

- 连续行为：混成系统的连续行为是指系统中物理对象随时间发生连续演变的行为。例如，飞行控制系统中飞机的高度和速度的连续变化行为。在混成系统中，当系统遵循特定的约束进行连续演化时，也称该系统的执行处于某一模态中。混成系统模态的约束条件可以使用微分方程或者时间函数进行表示。

- 同步事件：系统中不同组件之间进行交互和协同的机制。当系统各子组件之间需要同时进行模态迁移时，通过事件机制，可以实现不同组件之间的同步执行。一般地，在混成系统的各种建模语言中，同步事件以标签的形式用于标记模态的迁移，若两个组件含有相同的事件标签，则相应的模态迁移需要进行同步，即同时执行迁移。在混成系统中，同步事件往往与混成系统的模态迁移以及迁移发生时系统的离散动作进行组合的。

- 模态迁移：当混成系统的执行从一种连续行为转到另一种连续行为时，就产生一次模态的迁移。模态迁移后，系统的连续行为可能会和迁移之前的连续行为不同但也可能相同。若模态迁移只是组件的内部行为而不与其他

组件交互时，可以不用指定该迁移的同步事件，或者可以为其指定一个内部事件及标签。

- 离散动作：混成系统中的瞬时行为，可以使用差分方程或者关系谓词的形式表示，用于在模态迁移发生时重置系统变量的取值。

此外，还有模态不变式条件，主要用于对系统变量在连续演变过程中的取值范围进行约束。

混成系统的状态（state）：在任何时刻，混成系统的状态由系统所处的模态和所有系统变量的取值组成。混成系统的状态可以通过两种方式进行改变：

- 通过模态迁移时所执行的离散动作，可以同时改变系统的执行模态和变量的取值。

- 随着时间的消逝，根据连续行为的约束条件改变系统变量的值，但不会改变系统当前的执行模态。

若令 $s_i = (m_i, v_i)$ 表示混成系统的状态，其中 m_i 表示模态，v_i 表示变量取值，则混成系统的一趟执行（run）σ 可以表示为一个有限或无限序列：

$$\sigma = (m_0, v_0) \xrightarrow{\alpha_0} (m_1, v_1) \xrightarrow{\alpha_1} (m_2, v_2) \xrightarrow{\alpha_2} \cdots$$

其中，标签 α_i 表示为模态迁移所对应的标签或者由时间以及连续函数组成的连续标签, 比如连续标签可以使用二元组 (t_i, f_i) 表示，t_i 表示连续行为所消耗的时间，而且 $f_i(0) = v_i$，$f_i(t_i) = v_{i+1}$。那么，当 $0 \leq \tau \leq t_i$ 时，$f_i(\tau)$ 是满足模态的不变式条件的。

基于微分方程在混成系统模型中的重要性，我们在此介绍利普希茨连续的常微分方程。令 $V \subseteq \mathbb{R}^n$，$f_i : V \to \mathbb{R}$ 为连续函数，考虑如下常微分方程的初值问题：

$$\frac{d\vec{y}}{dt} = \vec{f}(\vec{y}(t, \vec{v}_0)), \quad \vec{y}(0, \vec{v}_0) = \vec{v}_0 \in V \tag{2.1}$$

该方程存在唯一解的充分条件是 f_i（其中，$1 \leq i \leq n$）为利普希茨连续函数，即存在常数 $k_i \in \mathbb{R}^+$，对于所有 $\vec{v}_1, \vec{v}_2 \in V$，满足条件：$\mid f_i(\vec{v}_1) - f_i(\vec{v}_2) \mid \leq k_i \parallel \vec{v}_1 - \vec{v}_2 \parallel$，

其中 $\|\cdot\|$ 表示最大范数。满足利普希茨连续条件的常微分方程称为利普希茨连续的常微分方程。

此外，常微分方程的解 $\vec{y}$ 是可计算的这个特性在 SMT 求解方法中具有重要的意义，下面我们介绍与可计算性相关的定义 [83, 137]。

定义 2.1.1 (名函数). *实数 $v \in \mathbb{R}$ 的名函数 γ_v 定义为：*

$$\gamma_v : \mathbb{N} \to \mathbb{D}$$

其中，$\mathbb{D}$ 为二进分数（dyadic rational）的集合。对于自然数 $i \in \mathbb{N}$，名函数 γ_v 满足 $|\gamma_v(i) - v| < 2^{-i}$。

对于多维向量 $\vec{v} \in \mathbb{R}^n$，$\gamma_{\vec{v}}(i) = \langle \gamma_{\vec{v}_1}(i), ..., \gamma_{\vec{v}_n}(i) \rangle$。实函数 f 的可计算性：如果存在一个图灵机可以利用函数 f 的任何参数 $\vec{v}$ 的名函数为 oracle [1]，并可计算满足任意精度 2^{-i}（其中，$i \in \mathbb{N}$）的函数值 $f(\vec{v})$，则称该实函数是可计算的。

定义 2.1.2 (可计算函数). *令 $V \subseteq \mathbb{R}^n$，若存在一个 oracle 图灵机 $\mathcal{M}_f$ 对于任意 $\vec{v} \in V$，任意 $\vec{v}$ 的名函数 $\gamma_{\vec{v}}$，以及任意 $i \in \mathbb{N}$，以 $\gamma_{\vec{v}}$ 为图灵机的 oracle，以 i 为输入参数，图灵机 $\mathcal{M}_f$ 可计算出有理数 $M_f^{\gamma_{\vec{v}}}(i)$，满足 $|M_f^{\gamma_{\vec{v}}}(i) - f(\vec{v})| < 2^{-i}$，则称函数 $f : V \to \mathbb{R}$ 是可计算的。*

命题 2.1.1. *形如微分方程 (2.1) 的利普希茨连续的常微分方程系统的解函数 $\vec{y}$ 在域 $\mathbb{R} \times V$ 上是可计算的 [83, 137]。*

基于微分方程解函数的可计算性，SMT 求解器 dReal 可以处理利普希茨连续的常微分方程系统的判定问题。

2.2　程序统一理论

1998 年，图灵奖获得者 C. A. R. Hoare 教授和华东师范大学何积丰院士共同提出了程序统一理论（UTP：Unifying Theories of Programming [71]）。在程

[1] 可以用于判断图灵机在给定输入情况下是否能够终止的一个黑盒 [42]

序语义方面，UTP 理论包含了 Dijkstra 提出的关于非确定性串行程序的指称语义 [43]，该指称语义基于 Tarski 关系理论 [133, 134] 以谓词的形式表示。为了适应实际程序开发，UTP 理论引入了设计（Design）关系，设计可以分解为假设（assumption）和承诺（commitment）。假设是设计者对程序执行前所需满足的条件的约定，承诺是指程序终止时需要满足的约束。通过设计关系的分解，可以实现大规模复杂软件系统的协同设计开发。此外，UTP 理论还为串行程序构建了完整的代数定律用于将程序转换为对应的规范型，为程序的精化、性质分析以及一致性判定等研究建立了坚实的理论基础。在语义一致性方面，UTP 提出了三种主要语义之间的联接理论，即：操作语义、指称语义和代数语义。经过近 20 年的研究和发展，UTP 理论已在各种语言和程序的研究中得到了广泛应用，相关工作涉及面向对象语言、Java 程序、硬件描述语言等等 [77, 28, 142]。

下面，我们简要回顾 UTP 理论中关系的定义和语义的表示方式。对于一个串行程序行为的观察，往往包含对程序执行前和执行后相关变量的取值的观察。从一般意义上看，这两类观察组成的集合即构成了程序的一种关系（relation）。在 UTP 理论中，为了描述程序的行为，关系以谓词的形式进行表示，其定义如下。

定义 2.2.1 (关系). *关系是一个二元组 $(\alpha P, P)$，P 是一个谓词，其字母表为 αP，即谓词 P 中除了 αP 中的变量外不含其他自由变量，αP 由两部分组成：*

$$\alpha P = in\alpha P \cup out\alpha P$$

其中，$in\alpha P$ 和 $out\alpha P$ 分别用于表示程序变量的初始取值和终止取值的情况。

一般地，字母表 $in\alpha P$ 称为 P 的输入变量，$out\alpha P$ 称为 P 的输出变量。在形式上，为了进行区分，$out\alpha P$ 中的变量名是带撇号（′）标记的，而 $in\alpha P$ 是不带撇号的。因此，二者满足关系：$out\alpha P = in\alpha' P$。关系可以用于表示程序的行为，并且关系谓词可以与程序结构进行组合而构造出比较复杂的关系。下面，我们介绍几种基本的关系类型，为了避免谓词与程序相互间的混淆，我们在阐述时使用了不同的符号进行区分。

赋值

假设赋值程序为 $AS =_{df} v_1 := e$，字母表 $A = \{v_i \mid 1 \leq i \leq n\} \cup \{v_i' \mid 1 \leq i \leq n\}$，$\alpha e \subseteq A$，赋值程序的关系谓词 P_{AS} 可以定义如下：

$$P_{AS} =_{df} (v_1' = e \wedge v_2' = v_2 \wedge ... \wedge v_n' = v_n)$$

其中，只有变量 v_1 的值可能发生了变化（$v_1' = e$），其他变量均没有发生改变（$v_i' = v_i$）。当不会产生歧义时，我们可以直接使用程序本身的符号来表示其对应的关系谓词。

条件

令条件选择程序为 $CC =_{df}$ `if`(b) `then` C_1 `else` C_2，关系谓词 P_1 表示程序分支 C_1 的行为，P_2 表示 C_2 的行为，则我们可以定义条件关系谓词 P_{CC} 为：

$$P_{CC} =_{df} P_1 \lhd b \rhd P_2 =_{df} (b \wedge P_1) \vee (\neg b \wedge P_2)$$

其中，$\alpha b \subseteq \alpha P_1 = \alpha P_2$。谓词 $P_1 \lhd b \rhd P_2$ 表示条件选择程序的行为：当条件 b 为真时，程序的行为可以由谓词 P_1 表示，当条件 b 为假时，程序的行为可以由谓词 P_2 表示。

组合

程序 S_1 与 S_2 的串行组合程序 $S_1; S_2$ 表示首先执行程序 S_1，然后执行程序 S_2。若关系谓词 P_1 和 P_2 分别表示程序 S_1 和 S_2 的关系谓词，则谓词组合可以定义如下：

$$P_1; P_2 =_{df} \exists v_0 \bullet P_1[v_0/v'] \wedge P_2[v_0/v]$$

其中，$out\alpha P_1 = in\alpha' P_2 = \{v'\}$，$v_0$ 代表变量 v 的中间取值，即程序 S_1 执行后的值，同时也是 S_2 开始执行时变量 v 的取值。$P_1[v_0/v']$ 表示将谓词 P_1 中的变量 v' 替换为 v_0。例如，如果 $P_1 = (v' = v + 1) \wedge v = 0.2$，则 $P_1[v_0/v'] = (v_0 = v + 1 \wedge v = 0.2)$。

在 UTP 理论中，还有更多复杂的关系以及关系之间的组合运算，在此我们

不再深入展开讨论。下面，我们介绍 UTP 理论的相关研究工作进展，主要有实时系统、概率程序、面向对象语言等方面的研究。

在实时系统方面，英国约克大学 Oliveira 等基于 UTP 理论，提出 Circus 语言 [138] 的指称语义 [109]，并基于定理证明技术实现了精化的自动证明 [110]。巴西贝南博古联邦大学（UFPE）Sherif 等，提出 Timed Circus 语言，并基于 UTP 理论对该语言进行了语义方面的研究 [128]。在概率程序方面，何积丰院士等研究了概率卫兵语言（PGCL）的指称语义和代数定律 [79]。爱尔兰都柏林大学圣三一学院 Butterfield 等对 UTP 理论中的设计（Design）概念进行扩展，加入了概率特性，并提出了 PGCL 的指称语义，研究了概率设计的健康条件 [25]。华东师范大学朱惠彪等在 PGCL 的基础上提出了集成概率、时间和共享变量特性的规范语言 PTSC，并研究了操作语义和指称语义，提出了概率首规范型，实现了语义的自动生成 [143, 144]。在面向对象语言方面，英国约克大学 Jim Woodcock 等研究人员基于 UTP 理论，对 Java 的存储模型进行了研究 [28]。何积丰院士等提出面向对象系统的精化演算理论 rCOS[77]，为对象系统的描述提出了统一表示方法。

2.3 霍尔逻辑

霍尔逻辑是由 C. A. R. Hoare 于 1969 年提出的 [69]。霍尔逻辑为程序的正确性验证提供了公理化的推理规则。在这一小节中，我们参考综述 [11]，主要介绍离散程序的霍尔逻辑基本推理规则。首先，考虑如下 while 语言：

$$W ::= \ v := e \ \mid \ W_1; W_2 \ \mid \ \texttt{if}(b)\ \{W_1\}\ \texttt{else}\ \{W_2\} \ \mid \ \texttt{while}(b)\ \{W\}$$

其中，v 代表变量，e 表示表达式，b 表示布尔条件。while 语言的程序总共有四类，$v := e$ 是赋值程序，即将表达式 e 的值赋给变量 v；$W_1; W_2$ 是串行程序；$\texttt{if}(b)\ \{W_1\}\ \texttt{else}\ \{W_2\}$ 是条件选择程序，当布尔条件 b 为真时，执行程序 W_1，否则执行程序 W_2；$\texttt{while}(b)\ \{W\}$ 表示循环程序，当布尔条件 b 为真时，执行程序 W，当 W 执行完成后继续判断布尔条件 b 是否满足，若满足则继续执行 W

直到布尔条件 b 不成立。

对于 while 语言的程序 W，霍尔逻辑断言公式可以表示为：$\{P\}\ W\ \{Q\}$，其中 P 和 Q 均为断言，W 为程序。程序断言 $\{P\}\ W\ \{Q\}$ 的直观含义是，若程序 W 执行前满足 P 而且程序可以终止，则终止时满足 Q。一般地，P 称为前条件（precondition），Q 称为后条件（postcondition）。下面，我们列出霍尔逻辑基本的公理和推理规则。

公理 1：赋值公理

$$\{Q[e/v]\}\ [v := e]\ \{Q\}$$

其中，$Q[e/v]$ 表示将 Q 中所有自由变量 v 替换为 e 后得到的新断言。$[v := e]$ 中的括号 '[' 和 ']' 用于分隔程序与谓词公式。

例 2.3.1. *赋值程序 $v := v + 1$，后条件 $Q = (v \geq 10)$，则根据公理 1，我们有*

$$\{v + 1 \geq 10\}\ [v := v + 1]\ \{v \geq 10\}$$

其中，$v + 1 \geq 10 \equiv Q[(v + 1)/v]$。

规则 1：串行组合规则

$$\frac{\{P\}\ [W_1]\ \{I\},\quad \{I\}\ [W_2]\ \{Q\}}{\{P\}\ [W_1; W_2]\ \{Q\}}$$

其中，串行程序的推理分解为程序 W_1 和 W_2 的推理，断言 I 为 W_1 执行结束时满足的条件，同时 I 也是 W_2 开始执行时所满足的条件。

例 2.3.2. *串行程序 $v := v + 1; v := v - 1$，后条件 $Q = (v = 0)$，前条件 $P = (v = 0)$，根据规则 1，我们有，*

$$\frac{\{v + 1 - 1 = 0\}\ [v := v + 1]\ \{v - 1 = 0\},\quad \{v - 1 = 0\}\ [v := v - 1]\ \{v = 0\}}{\{v = 0\}\ [v := v + 1; v := v - 1]\ \{v = 0\}}$$

其中，$(v+1-1 = 0) \equiv (v = 0) \equiv P$。中间断言 $I = (v-1 = 0) \equiv (v = 0)[(v-1)/v]$。

规则 2：条件选择规则

$$\frac{\{P \wedge b\}\ [W_1]\ \{Q\},\quad \{P \wedge \neg b\}\ [W_2]\ \{Q\}}{\{P\}\ [\texttt{if}(b)\ \{W_1\}\ \texttt{else}\ \{W_2\}]\ \{Q\}}$$

从上述规则可以知道，断言 P 和 Q 均组成了两个选择分支的断言。根据条件 b 的满足性，分别设置每个分支的前条件。由于 b 不能同时为真和假，因此，在同一时刻只有一个分支是可执行的。

例 2.3.3. 条件选择程序 $\texttt{if}(v=0)\ \{v := v+1\}\ \texttt{else}\ \{v := 0\}$，断言 $P = (v = 0)$，$Q = (v = 1)$，则根据规则 *2*，可得，

$$\frac{\{(v=0) \wedge (v=0)\}\ [v := v+1]\ \{v=1\},\quad \{(v=0) \wedge \neg(v=0)\}\ [v := 0]\ \{v=1\}}{\{v=0\}\ [\texttt{if}(v=0)\ \{v := v+1\}\ \texttt{else}\ \{v := 0\}]\ \{v=1\}}$$

其中，分支程序 $v := v+1$ 的前条件 $(v=0) \wedge (v=0) \equiv (v=0) \equiv (v=1)[(v+1)/v]$。分支程序 $v := 0$ 的前条件 $(v=0) \wedge \neg(v=0) \equiv false \equiv (v=1)[0/v]$。

规则 3：循环规则

$$\frac{\{Q \wedge b\}\ [W]\ \{Q\}}{\{Q\}\ [\texttt{while}(b)\ \{W\}]\ \{Q \wedge \neg b\}}$$

其中，Q 在程序的每次迭代开始和结束时均满足，通常这样的断言称为不变式（invariant）。另外，循环程序在执行完成时，条件 b 是不成立的，而在迭代开始前 b 是成立的。

例 2.3.4. 循环程序 $\texttt{while}(v>0)\ \{v := v+1\}$，不变式 $Q = (v > 1)$，根据规则 *3*，我们有，

$$\frac{\{(v>1) \wedge (v>0)\}\ [v := v+1]\ \{v>1\}}{\{v>1\}\ [\texttt{while}(v>0)\ \{v := v+1\}]\ \{(v>1) \wedge \neg(v>0)\}}$$

循环程序的后条件 $(v>1) \wedge \neg(v>0) \equiv false$，表示循环是不可终止的。迭代

程序 $v := v + 1$ 的前条件 $(v > 1) \wedge (v > 0) \equiv (v > 1) \neq (v > 1)[(v + 1)/v]$。但是，$(v > 1) \Rightarrow (v > 1)[(v + 1)/v]$。这表明，使用赋值公理得出的前条件是能够被 $(v > 1)$ 推出的。这在下面的推论规则中进行了表示。

规则 4：推论规则

$$\frac{P \Rightarrow \mathcal{P}, \quad \{\mathcal{P}\}\ [W]\ \{\mathcal{Q}\}, \quad \mathcal{Q} \Rightarrow Q}{\{P\}\ [W]\ \{Q\}}$$

其中，前提中的前条件 $\mathcal{P}$ 可以被结论中的前条件 P 推出，即 $\mathcal{P}$ 是 P 的弱化。对于后条件，$\mathcal{Q}$ 可以推出 Q，即后条件 $\mathcal{Q}$ 是 Q 的强化。

例 2.3.5. 对于例*2.3.4*中的前提，我们可以利用规则 *4* 进行推理，

$$\frac{(v > 1) \wedge (v > 0) \Rightarrow v > 0, \quad \{v > 0\}\ [v := v + 1]\ \{v > 1\}, \quad v > 1 \Rightarrow v > 1}{\{(v > 1) \wedge (v > 0)\}\ [v := v + 1]\ \{v > 1\}}$$

其中，$(v > 1) \wedge (v > 0) \equiv (v > 1)$，而且 $(v > 1) \Rightarrow (v > 0)$。

以上为基本的霍尔逻辑公理和规则，通过对霍尔逻辑进行扩展，我们可以对含有系统资源约束或并发特性的程序进行推理验证，相关的研究工作有分离逻辑（separation logic [122]）、Rely/Guarantee 推理方法 [81] 以及二者的融合 [136] 等。

2.4　可满足性模理论

简单来说，给定变量的约束，可满足性判定问题是一个查找变量赋值以使约束条件得以满足或者对不能满足的约束给出相应解释的求解过程。命题可满足性问题（SAT：propositional satisfiability）是约束可满足性问题中比较重要的一类。在 SAT 中，逻辑公式中的变量都是布尔变量，即变量的取值只能是真（`true`）或假（`false`）。但是，在有些问题中，我们要求公式的表达能力更强，比如，在公式中使用量词、数学函数、谓词符号、逻辑连接符等等。此时，满足公式的解

可以称为该公式的模型，该模型是能够使公式为真的变量、函数和谓词符号等的一种解释。通常，上述解释中所涉及的符号可能会受相关背景理论的约束。举例来说，算术理论可以对加法（+）和减法（−）符号进行约束。

可满足性模理论（SMT：satisfiability modulo theories）问题研究的是，基于背景理论判定逻辑公式的可满足性。SMT 的研究开始于形式化方法领域 Greg Nelson、Derek C. Oppen、Robert E. Shostak 以及 Robert S. Boyer 等学者在判定过程方面的研究工作 [129, 103, 130, 104, 131, 24]。

SMT 相关技术现已在各种分析验证工具中得以应用，如交互式定理证明器 HOL[2] [56]、Isabelle[108] 以及 PVS[111]，还有模型检验工具 BLAST[20]、SLAM[17] 等。下面介绍 SMT 中涉及到的部分理论。

数组

数组理论主要用于对程序中数组状态的变化行为进行编码。假设 S 为程序中的一个数组，当我们对数组进行写操作时，数组的状态可能会发生改变。在 SMT 中，我们可以使用 $write(S, i, t)$ 表示更新后的数组，其下标 i 处的元素为 t。

另外，与写操作相对应的是，$read(S, i)$ 返回数组 S 在下标 i 处的元素 $S[i]$。因此，对于数组理论，我们有如下等价关系：$read(write(S, i, t), i) = t$。当下标 i 和 k 都是数组 S 的有效下标，而且不相等时，$read(write(S, i, t), k) = read(S, k)$。

差分算术

差分算术（difference arithmetic）理论考虑的约束公式形式为：

$$u - v \leq n$$

其中，n 是一个整型常数，变量 u 和 v 的取值范围为整数集合 $\mathbb{Z}$。满足上述形式的约束称为差分约束。不具备上述形式的公式可以通过一定的转换方法重写为上述形式。

[2] http://www.cl.cam.ac.uk/research/hvg/HOL/

$$(u = v) \equiv (u - v \leq 0) \wedge (v - u \leq 0)$$

$$(u - v < m) \equiv u - v \leq m + 1$$

$$(u \leq n) \equiv u - zero \leq n$$

其中，n，m 均为整型常数，$zero$ 为代表数字 0 的特殊变量。对于多个差分约束的合取公式，可以将约束公式用使用带权有向图表示，并可以通过 Bellman-Ford 算法 [38] 进行有效的求解。

非线性算术

非线性算术（non-linear arithmetic）理论涉及的约束公式形式如下：

$$\theta_1 \cdot T_1 + ... + \theta_n \cdot T_n \sim \lambda$$

其中，$\sim \in \{=, <, \leq, \geq, >\}$，$\theta_i$ 和 λ 是整数或有理数，T_i 为非线性单项式，其形式为 $\prod_{j=0}^{m} x_j^{e_j}$，且 $m > 0$，$e_j > 0$。变量域为实数的无量词非线性实算术可满足性问题是可判定的，但是当问题涉及超越函数时则是不可判定的 [135]。

实现 SMT 求解器的方法主要有三种较成功的技术，分别是 eager、lazy 和 DPLL(T) 方法 [127, 126, 106]。

eager 方法利用与背景理论相关的推论和结果，通过设计专门的转换过程将 SMT 公式转换为等价的布尔公式，然后在 SAT 求解器中进行判定。该方法的优点是，转换后的公式可以使用任何现有的 SAT 求解器进行判定，而不依赖于特定的求解器。与此相反，lazy 方法本质上是构建特定于背景理论的推理系统。该方法的优点是在求解器实现时，我们可以设计特制的算法和数据结构以使求解的效率最高。

DPLL(T) 方法基于一个通用的框架 DPLL(X)，该框架不对具体的背景理论进行绑定。在实现求解器时，可以将背景理论（T）的求解器与该框架进行组合，从而实现 DPLL(T) 求解器。该方法的优点是比较灵活，在该框架中实现背景理论相关的求解器成本较低，同时又可借助 DPLL 算法 [41] 的高效率实现 SMT 问题的快速求解和判定。

2.5 本章小结

本章，我们主要介绍了混成系统和微分方程相关的概念和定义、程序统一理论的基本概念和相关示例。此外，我们介绍了经典的串行程序的霍尔逻辑推理规则，以及可满足性模理论的基本概念。这些基本概念和背景知识是后续章节内容的研究基础。

第三章　混成系统的协同验证

关于混成系统的形式化分析，在学术界，有基于混成自动机模型的形式化分析工具用于验证；在工业控制和嵌入式系统领域，使用较为广泛的则是基于Matlab Simulink/Stateflow工具套件的建模和仿真。形式化验证的优势在于其在一定程度上可用于确保系统的安全性，即对于需要验证的安全特性，我们可以将系统要避免的状态或不安全行为，以逻辑公式的形式来表达，若形式化验证工具返回的验证结果为不可达，则我们可以确认该系统满足所要求的安全性。然而，目前形式化验证方法所应用的模型往往存在规模小，系统行为简单等特点，对于大规模复杂系统的验证往往不能在可接受的时间内返回结果，或者经常产生状态爆炸现象而无法得出验证结果，这就极大地限制了形式化方法的应用范围。在工业界应用广泛的仿真工具却不存在上述问题，对于规模较大的系统，其仿真结果可以较快得出，并支持非线性系统的建模和分析，还提供丰富的仿真数据展示和二次开发平台。其原因在于，仿真只是分析系统部分的有效执行过程，对系统的状态未能做全覆盖分析，因而可实现快速的仿真结果反馈。仿真的劣势也缘于此，即系统的状态空间往往是一个无穷的集合，我们无法尝试所有可能的仿真参数取值情况和系统执行路径，因而无法确保系统的完全正确性和安全性。

鉴于此，本章我们提出一种协同验证过程, 其综合了形式化验证工具和工业界仿真技术的优势，对地铁控制系统中的屏蔽门控制系统，以及列车防碰撞控制，我们建立了相应的混成系统模型和状态图模型，并进行了形式化验证和仿真分析。

3.1 引言

混成系统的概念起源于嵌入式控制系统，其最大的特点是，在嵌入式系统中往往存在着连续的物理行为与离散的数字控制器之间的频繁交互。在地铁控制系统中，列车的行为属于连续的物理行为，无处不在的列车信号控制是基于计算机网络和数字技术的，因此，离散的数字控制器是实现地铁有序可控运营所必不可少的控制手段。我们在本章主要应用形式化方法和仿真技术对地铁控制系统中的两个案例进行分析，即屏蔽门控制系统和列车防碰撞控制。本章中，我们建立的地铁控制系统的混成系统模型均以混成自动机 [7, 67] 为形式化建模语言，以 Simulink/Stateflow 为仿真模型语言。

混成系统的验证一直是一个重要并极具挑战的研究方向 [12, 14, 55, 62, 85, 61]。对于线性混成系统，基于模型检验的算法研究主要有 [7, 10, 105]。在验证工具方面，HyTech [68, 66] 是第一个支持线性混成系统验证的工具。由法国 VERIMAG实验室[1]Goran Frehse 教授提出的 PHAVer [47, 48] 是继 HyTech 之后混成系统的又一形式化验证工具，其可支持连续变量导数分段有界，以及具有仿射动态行为的混成系统验证。为了实现精确的数值运算，PHAVer 采用了帕尔马多面体库（PPL [16]: Parma Polyhedra Library[2]，一个 C++ 语言编写的程序库，可以提供数值抽象，主要应用于复杂系统的分析和验证）。对于线性混成系统中的分段仿射动态特性可通过对系统的可达状态做即时的上逼近来处理。对于规模较大的系统，SpaceEx是一个可选的工具，该工具也由法国 VERIMAG 实验室的 Goran Frehse 等提出，目前可以支持的变量数目为 100 个以上 [49]。

此外，基于仿真的工具在工业界应用十分广泛。美国 MathWorks 公司的 Simulink/Stateflow 工具箱是其中使用较多的一个仿真工具。我们可以在 Stateflow 中对连续时间系统进行建模，也可以建立既包含连续行为又有离散状态迁移的系统，如混成系统，并在 Simulink 中对系统进行仿真。而且，我们可

[1] http://www-verimag.imag.fr

[2] PPL 源码和库文件等可以通过如下网址下载，http://bugseng.com/products/ppl/

以在 Simulink/Stateflow 中对模型进行调试，并设置执行断点，比如，我们可以在一个子系统的执行入口设置断点，也可以对事件广播和状态入口等进行断点设置和跟踪调试。在本章中，我们就是通过在对仿真模型进行调试的过程中发现屏蔽门控制系统的潜在问题的。调试的优势在于，开发者可以在分析过程中通过调试手段与模型进行交互，比如模拟某个模块的事件触发，修改某个变量的值等等，这些特性是形式化验证工具目前所无法提供的。

因此，我们提出反馈演进的验证过程以整合验证与仿真的优势来对地铁控制系统案例进行分析。使用 Simulink/Stateflow 工具箱，我们将得益于其丰富的组件库，以及对复杂行为提供的建模和仿真分析，比如 Stateflow 支持非线性方程，以及在状态迁移中使用非线性表达式，并支持 C 语言风格的内部函数定义等等。然而，仅仅使用仿真是不能保证系统的完全正确性的，利用形式化工具的验证却可以做到这一点。不过，形式化验证的结果往往不易得到，对于复杂的混成系统，我们在验证上需花费较多的时间才能得出结论。而仿真可以提供可视化的视图，开发者可以从具体的角度在分析过程中及时了解系统行为，因此，仿真结果可以对验证过程提供反馈信息，并及时修改验证参数和命令，改进具体的验证过程。所以，对仿真和验证进行合理整合是具有一定的积极意义的。

本章中我们使用的形式化验证工具是 SpaceEx/PHAVer，采用的仿真工具为 Simulink/Stateflow。对于地铁屏蔽门系统（PSDS：Platform Screen Doors System），首先在上述工具中建立初步的模型，并分别在验证和仿真工具中对有界的活性（Liveness）性质进行分析。在仿真过程中，我们发现了乘客会被夹于屏蔽门与车门间空隙的情况，在此场景中，乘客将无法返回站台，也无法进入列车，从而当列车启动后，该乘客将处于极其危险的境地。产生如此危险情形的原因在于，当列车门关闭后，屏蔽门即立刻开始关闭导致乘客无法及时返回站台。我们对模型进行修正，并进行仿真然后对验证工具所用模型也进行修改，并在 PHAVer 中进行验证，确保此类危险情形不再发生。对于列车防碰撞控制，我们在模型中加入 4 辆列车以及 4 个站点，通过仿真，我们并未发现碰撞情况，但通过 SpaceEx 的验证，我们发现模型存在两个列车同时处于相同位置的情况，如

此，验证结果可以对仿真进行反馈，从而可对仿真模型进行修正。

本章余下部分将组织如下：3.2节，我们介绍混成自动机概念，以及基本的可达性分析过程。Stateflow 建模语言将在3.3节中介绍。在3.4节中我们提出反馈演进验证过程。屏蔽门控制系统的需求在3.5.1节中阐述。接着，在3.5.2节中，我们提出系统的初步模型, 并在3.5.3节中对模型进行仿真和验证。模型的优化在3.5.4节中进行。基于此，我们建立列车防碰撞控制所需的模型，该工作在3.6节中做了详细的说明。

3.2 混成自动机及可达性分析

本节主要介绍混成自动机概念，包括线性混成自动机和仿射混成自动机的定义，并给出列车的自动机模型。一般而言，混成自动机是一个有向控制图，图中的节点表示控制模态，图中的边表示离散控制迁移，混成自动机可以用于表达实值变量随时间非确定性连续演化行为，以及控制模态间的离散迁移动作。

3.2.1 混成自动机

定义 3.2.1 (混成自动机). 一个混成自动机是一个元组结构 $A = \langle X, V, flow, inv, init, E, jump, \Sigma, syn\rangle$ *([10])*，其中：

1. **变量**：$X = \{x_1, x_2, \cdots, x_n\}$ 是一个由 n 个实值变量组成的有限集合，n 称为 A 的维度。

2. **控制模态**： V 是一个由控制模态（*Control Mode*，或称为 *Location*）组成的有限集合。对于每个控制模态 $v \in V$，初始条件 $init(v)$ 是作用在集合 X 中变量上的一个谓词。混成自动机的初始状态需满足相应的控制模态上设定的初始条件。

3. **连续流条件**： 对于每个控制模态 $v \in V$，谓词 $flow(v)$ 作用于集合 $X \cup \dot{X}$ 内的变量。其中，$\dot{X} = \{\dot{x_1}, \dot{x_2}, \cdots, \dot{x_n}\}$ 是变量 x_i 关于时间的一阶导数组

成的集合，$1 \leq i \leq n$。若自动机当前的执行处于控制模态 v 中，则 X 中的变量及其时间导数应满足连续流条件 $flow(v)$。

4. **不变式：** 对于每个控制模态 $v \in V$，存在不变性条件 $inv(v)$，该不变性条件称为不变式。$inv(v)$ 是一个对 X 中变量进行约束的谓词。当自动机当前执行所处的控制模态为 v 时，X 中变量的取值需要一直满足该不变式直到发生离散迁移。不变式一般用于指定连续变量的取值范围。

5. **离散控制迁移：** 对于两个控制模态 v 和 v'，如果自动机 A 有一个从模态 v 跳转到模态 v' 的离散动作，那么 $(v, v') \in E$ 称为一个离散控制迁移，v 为迁移的源模态，v' 为迁移的目标模态。需要注意的是，离散控制迁移集 E 是一个多重集（*Multi-Set*）。

6. **跳转条件：** 对于每个离散控制迁移 $e \in E$，存在一个跳转条件 $jump(e)$。跳转条件 $jump(e)$ 是一个作用于集合 $X \cup X'$ 的谓词。变量 $x \in X$ 用于表示迁移前变量的取值，变量 $x' \in X' = \{x' \mid x \in X\}$ 表示迁移后变量新的取值。

7. **事件：** 事件的集合用 Σ 来表示。对于每个离散控制迁移 $e \in E$，我们为 e 指定一个标签 $syn(e)$, 该标签即称为一个事件。其中，syn 为一个标签函数，$syn : E \to \Sigma$。事件主要在多个自动机并发时，用于协调和同步自动机之间的离散迁移行为的。通常，我们也使用同步标签（*Synchronization Label*）来指代事件。

如果 v 是一个控制模态，$r = (r_1, ..., r_n) \in \mathbb{R}^n$ 是 X 中变量的一个赋值，那么，我们称二元组 (v, r) 是混成系统 A 的一个状态。一个混成自动机的状态 (v, r) 是可接受的，当且仅当此时模态 v 的不变式 $inv(v)$ 为真。例如：假设 $X = \{x\}$，模态 v 的不变式为 $inv(v) = (x \geq 1 \wedge x \leq 2)$，变量赋值 $r = (1.72)$，则状态 (v, r) 是可满足的，因为将 r 代入 $inv(v)$ 中可知 $(1.72 \geq 1 \wedge 1.72 \leq 2)$ 为真。

混成系统往往由多个子系统（模块）构成，各个子系统可以并发执行，相互间可以通过通信机制来协调。下面我们定义混成自动机的并发组合，自动机之间的通信机制抽象为同步标签（事件）的方式来完成，即若两个自动机具有相同的同步标签，则二者需同步执行离散迁移。

定义 3.2.2 (混成自动机的并发组合). 给定两个混成自动机如下：

$$A_1 = \langle X_1, V_1, flow_1, inv_1, init_1, E_1, jump_1, \Sigma_1, syn_1 \rangle,$$

和

$$A_2 = \langle X_2, V_2, flow_2, inv_2, init_2, E_2, jump_2, \Sigma_2, syn_2 \rangle.$$

令 $A_{1||2} = \langle X, V, flow, inv, init, E, jump, \Sigma, syn \rangle$ 为 A_1 和 A_2 的并发组合，则有，

1. $V = V_1 \times V_2$，$X = X_1 \cup X_2$，$syn = syn_1 \cup syn_2$。控制模态的集合为 V_1 与 V_2 的笛卡尔积；变量的集合是 X_1 与 X_2 的并集；同步标签也为各自集合的并集。

2. 对于下标 $i = 1, 2$，离散控制迁移 $e_i = (v_i, v_i') \in E_i$，若相应的标签相等，即有 $syn_1(e_1) = syn_2(e_2)$，那么我们有离散控制迁移 $e = ((v_1, v_2), (v_1', v_2')) \in E$，同时，跳转条件为各自条件的逻辑合取 $jump(e) = jump_1(e_1) \wedge jump_2(e_2)$，而且同步标签不变 $syn(e) = syn_1(e_1) = syn_2(e_2)$。

3. 若有控制模态 $v_1 \in V_1$，$v_2 \in V_2$，则存在控制模态 $v = (v_1, v_2) \in V$，模态 v 的不变式为 $inv(v) = inv_1(v_1) \wedge inv_2(v_2)$。

4. 如果同步标签 $syn_1(e_1) \notin \Sigma_2$，那么若存在迁移 $e = ((v_1, v_2), (v_1', v_2)) \in E$，则须满足 $syn(e) = syn_1(e_1)$ 及 $jump(e) = jump_1(e_1)$；当 $syn_2(e_2) \notin \Sigma_1$ 时情况类似。

5. 对于初始条件，若模态 $(v_1, v_2) \in V$，则有 $init((v_1, v_2)) = init(v_1) \wedge init(v_2)$，其中 v_1 是混成自动机 A_1 的模态，v_2 为 A_2 的模态。

在此，需要注意的地方是，当控制迁移 $e_1 \in E_1$ 以及 $e_2 \in E_2$ 具有相同的同步标签 $syn(e) \in \Sigma$，$e \in E$ 时，控制迁移 e_1 和 e_2 需要同步执行。换句话说，当跳转条件 $jump_1(e_1)$ 和 $jump_2(e_2)$ 都为真，且 A_1 与 A_2 都可以执行相应的迁移时，控制迁移 e_1 和 e_2 需立即同时被触发。在本章中，我们经常会使用指令（*Command*）或信号（*Signal*）等词汇来表示同步标签。例如，控制器给列车发命令，即表示控制器与列车的混成自动机模型中具有相同的同步标签。

3.2.2　混成自动机的语义

混成自动机的行为可以描述为离散和连续迁移，以及这两者的组合。文献[65] 中给出了详细的语义，我们在此只阐述其关键的部分。在程序语言和时间相关系统方面的研究，对于系统行为的分析，研究者们采用的方法往往是将待定义的系统规约为迁移系统，然后再此基础上做形式化分析和验证。下面我们给出相关定义。

定义 3.2.3 (标号迁移系统). 一个标号迁移系统（*Labeled Transition System*）是一个四元组 $LTS = \langle S, S_{init}, L, \rightarrow_l \rangle$，其中，

1. S 为一个状态集合。

2. $S_{init} \subseteq S$ 是初始状态集合。

3. L 是标签集合。

4. $\rightarrow_l$ 为状态集合 S 上的一个带标签的二元关系，$l \in L$。

5. 设状态 $p, q \in S$，则称 $p \rightarrow_l q$ 为从 p 到 q 的一个迁移。一般地，我们称状态 q 为状态 p 的后继（*Successor*），相应地，p 称为 q 的前驱（*Predecessor*）。

现在，我们可以用标号迁移系统来定义混成自动机的行为如下。

定义 3.2.4 (混成自动机的迁移语义). 混成自动机 $A = \langle X, V, flow, inv, init, E, jump, \Sigma, syn\rangle$ 的语义可由标号迁移系统 $LTS_A = \langle S^A, S^A_{init}, L^A, \rightarrow^A_{\alpha|\delta}\rangle$ 的语义进行定义，其中：

1. 混成自动机 A 的维度为 n。状态集合是 $S^A \subseteq V \times \mathbb{R}^n$。初始状态集合 $S^A_{init} \subseteq V \times \mathbb{R}^n$。状态 $(v, r) \in S^A$ 当且仅当 $inv(v)[r/X] = true$，即当变量赋值为 r 时，不变式 $inv(v)$ 为真。同理，$(v, r) \in S^A_{init}$ 当且仅当 $inv(v)[r/X] = true$ 以及 $init(v)[r/X] = true$。表达式 $inv(v)[r/X]$ 表示将 $inv(v)$ 中出现的集合 X 中的变量用 r 中相应的变量取值进行替换所得到的谓词公式。

2. 标号迁移系统的标签集合包含事件与时间段，即有 $L^A = \Sigma \cup \mathbb{R}_{\geq 0}$。对于混成自动机中离散控制迁移，我们在标号迁移系统中有对应的迁移 $(v, r) \rightarrow^A_\alpha (v', r')$ 当且仅当 A 中存在迁移 $e \in E$，且跳转条件为真 $jump(e)[r, r'/X, X'] = true$，事件 $syn(e) = \alpha$，其中离散控制迁移 e 的源模态和目标模态分别为 v 和 v'。离散迁移可能改变模态也可能同时改变了变量的取值。

3. 对于混成自动机的连续行为，考虑一个时间段 $\delta \in \mathbb{R}_{\geq 0}$，则在标号迁移系统 LTS_A 中存在迁移 $(v, r) \rightarrow^A_\delta (v, r')$ 当且仅当存在函数 $f : [0, \delta] \rightarrow \mathbb{R}^n$，$f$ 是一个可微函数，其在初始时刻的取值 $f(0) = r$，在时段结束时刻取值 $f(\delta) = r'$。对于所有中间时刻 $\epsilon \in (0, \delta)$，不变式始终成立 $inv(v)[f(\epsilon)/X] = true$，且连续流条件也一直成立 $flow(v)[f(\epsilon), \dot{f}(\epsilon)/X, \dot{X}] = true$。需要注意的是，连续迁移没有改变模态，只可能改变变量的取值。

3.2.3 混成自动机示例

在本小节中，我们将对列车进行建模。在这个模型中，距离的单位为米（m），时间的单位是秒（s），因而可设定速度和加速度的单位分别为米每秒（m/s）和

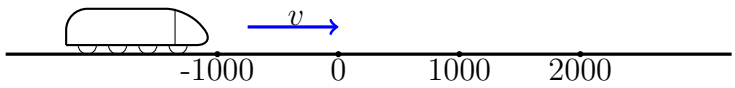


图 3.1: 行驶在地铁轨道上的列车，运行方向从左至右。变量 v 表示列车的速度，标注为 0 的位置是地铁站所在的位置

米每二次方秒（$\mathrm{m/s^2}$）。除非特别说明，在本章中所用到的度量单位均以此为准。

考虑如图3.1所示的列车，以向右为正方向，该列车初始位置为 -1000，即列车处于站点（标记为 0 的位置）的左边，且列车与站点之间的距离有 1000 米。此时，列车离站点较近，假设此时有传感器检测到该列车的位置，则我们可以发送信号给控制器以传递列车的位置信息。在这个例子中，我们假设列车在接近站点时，其速度是在一定范围内变化的，即对于速度我们有约束条件 $v \in [0, 16]$。列车的减速度 $\dot{v} = -0.128\ \mathrm{m/s^2}$，因为此时列车需要停靠在站点，在接近站点之前通过减速来达到顺利停靠的目的。当列车离站点只有 0.5 米的距离时，列车会发送一个 $near\_stop$ 信号给控制器，然后列车会停在站点。当列车停在站点时列车的速度和位置都被设置为 0，这是对列车具体行为的一个抽象，如当列车非常接近站点（距离小于 0.5 米）时，可能需要对列车的位置和速度进行微调，将列车门和屏蔽门对齐等，使得列车能够准确地停在合理的位置上，这个过程在我们的模型中没有细化，而是抽象为变量重置的操作。当列车从站点离开，列车将加速，加速度为 $0.128\ \mathrm{m/s^2}$。当列车与站点的距离达到 2000 米时，我们又设置其为接近下一个站点的运行模态中，如此便是列车的一个简单和抽象的运行过程。列车的模型如图3.2所示。

列车混成自动机的变量集合 $X = \{x, v\}$，其中 x 表示列车与站点间的距离，v 是列车的速度变量。该混成自动机有三个模态，控制模态的集合为 $V = \{near, stop, leave\}$。控制模态 $near$ 的连续流条件为 $flow(near) = (\dot{x} = v) \wedge (\dot{v} = -0.128)$，不变式为 $inv(near) = (-1000 \leq x \leq 0) \wedge (0 \leq v \leq 16)$。令 $r = (r_1, r_2)$，其中 $r_1, r_2 \in \mathbb{R}$，r_1 是变量 x 的一个取值情况，r_2 是变量 v 的取值，如果 $r = (-1000, 16)$ 且状态 $(near, r)$ 是列车自动机的初始状态，那么状态 $(near, r)$ 对于模态 $near$ 来说是可接受的，因为 r 满足不变式条件 $inv(near)$，即

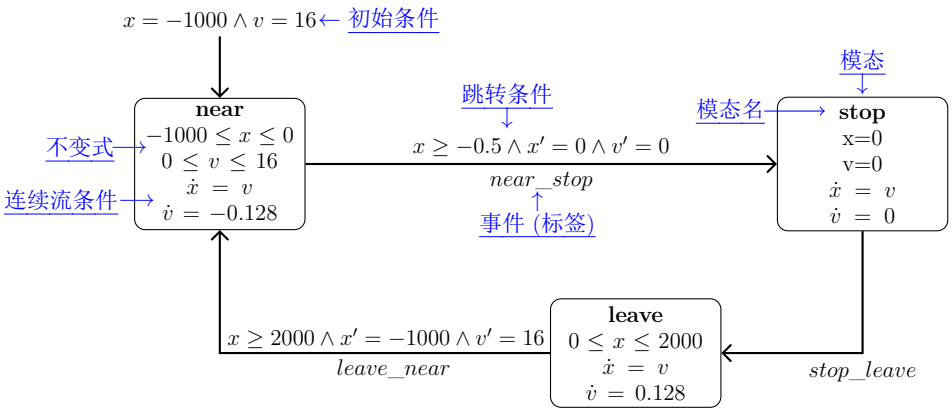


图 3.2: 列车混成自动机，初始模态为 $near$，变量 x 表示列车的位置，初始位置为 -1000, 初始速度 $v = 16$

$(-1000 \leq -1000 \leq 0) \wedge (0 \leq 16 \leq 16) \equiv true$。控制迁移 $cs = (stop, leave)$ 的相关事件为 $stop\_leave$，其跳转条件未指定，此时若事件 $stop\_leave$ 不需和其他系统组件同步，则 cs 的跳转条件即为 true，是立即满足的。

若事件 $stop\_leave$ 需要与其他组件中的事件同步，比如控制器，则需要等待控制器可以执行相应的迁移时方可一起触发。鉴于模态 $stop$ 中变量 x 和 v 始终为 0，我们可以认为，迁移 cs 的跳转条件可以写为：$x = 0 \wedge v = 0 \wedge x' = 0 \wedge v' = 0$。

3.2.4　线性与仿射混成自动机

线性混成自动机（LHA：Linear Hybrid Automata [10]）是混成自动机的一个子类。要理解线性混成自动机，首先需要了解线性表达式（Linear Expression），线性约束（Linear Constraint）和线性谓词（Linear Predicate）等 [65, 46]。下面，我们将对这些概念依次说明。

1. **线性表达式：** 关于变量集合 $Vars$ 的线性表达式具有如下形式：

$$s_1 x_1 + ... + s_n x_n + t$$

其中常数 $s_1, ..., s_n, t \in \mathbb{R}$，变量 $x_1, ..., x_n \in Vars$，$\mathbb{R}$ 是实数的集合。

2. **线性约束：** 线性约束是具有如下形式的表达式：

$$le \bowtie 0$$

其中，le 是一个线性表达式，关系符号 $\bowtie \in \{<, \leq, =\}$。

3. **凸线性谓词：** 一个凸线性谓词是若干线性约束的有限合取式：

$$(le_1 \bowtie 0) \wedge (le_2 \bowtie 0) \wedge \cdots \wedge (le_m \bowtie 0)$$

其中，$m \in \mathbb{N}$ 是一个大于或等于 1 的自然数，$(le_i \bowtie 0)$ 为线性约束，$1 \leq i \leq m$。

4. **线性谓词（或称非凸线性谓词）：** 凸线性谓词的有限析取即构成一个线性谓词。形式如下所示：

$$\bigwedge_{i=1}^{n_1}(le_i^1 \bowtie 0) \vee \bigwedge_{i=1}^{n_2}(le_i^2 \bowtie 0) \vee \cdots \vee \bigwedge_{i=1}^{n_m}(le_i^m \bowtie 0)$$

其中，$\bigwedge_{i=1}^{n_k}(le_i^k \bowtie 0)$ 为凸线性谓词，且 $m \geq 1$，$1 \leq k \leq m$，$n_k \geq 1$。当 $m = 1$ 时，线性谓词就是凸线性谓词。

我们可以定义线性混成自动机 $LHA = \langle X, V, flow, inv, init, E, jump, \Sigma, syn \rangle$ 如下。线性混成自动机是满足如下条件的混成自动机：

- 对于混成自动机中的每个控制模态 $v \in V$，其不变式 $inv(v)$ 和初始条件 $init(v)$ 均为线性谓词。
- 对于混成自动机中的每个控制迁移 $e \in E$，其跳转条件 $jump(e)$ 是线性谓词。如：$x = 0 \wedge x' = 1$。
- 对于混成自动机中的每个控制模态 $v \in V$，其连续流条件 $flow(v)$ 是只涉及变量集合 $\dot{X}$ 中的变量和 $\mathbb{R}$ 中常数的线性谓词。如：$\dot{x} = 1.5$。

仿射混成自动机（AHA：Affine Hybrid Automata[44]）也是混成自动机的一个子类。其与线性混成自动机的主要区别在于，在仿射混成自动机中，对于每个

模态 $v \in V$，连续流条件 $flow(v)$ 是一个仿射动态谓词。仿射动态谓词定义于变量集合 $X \cup \dot{X}$ 的变量之上，其具有如下形式：

$$\dot{x} = le$$

其中，le 是一个线性表达式。如：$\dot{x_1} = x_2 + 1 \wedge \dot{x_2} = x_1 - 1$。

图3.2中流条件 $\dot{x} = v$ 是一个仿射动态谓词。列车混成自动机即为仿射混成自动机的一个例子。在形式化分析工具方面，HyTech 是一个用于线性混成自动机形式化验证的工具。SpaceEx/PHAVer 可支持仿射混成自动机的形式化验证。

3.2.5 可达性分析

通常，一个混成系统是否满足某个安全性质，可以通过分析系统的可达状态集来决定。对于一个混成自动机 $A = \langle X, V, flow, inv, init, E, jump, \Sigma, syn \rangle$，假设存在一个连续时间迁移：

$$(v, r) \rightarrow_{\delta} (v, r')$$

其中，(v, r) 和 (v, r') 是 A 的两个状态，$v \in V$ 是 A 的一个控制模态，$\delta \in \mathbb{R}_{\geq 0}$ 是一个时段。令可微函数 $f : [0, \delta] \rightarrow \mathbb{R}^n$ 表示上述时间迁移所对应的连续流曲线，则对于时间区间 $[0, \delta]$ 的两个端点，我们有 $f(0) = r$，$f(\delta) = r'$。所有状态 $(v, f(t))$ 在时刻点 $t \in [0, \delta]$ 是可接受的，连续流条件 $flow(v)$ 和不变式 $inv(v)$ 在区间 $[0, \delta]$ 上始终成立。对于离散迁移，我们有从状态 (v, r) 到状态 (v', r') 的迁移：

$$(v, r) \rightarrow_{\alpha} (v', r')$$

其中，离散控制迁移 $e = (v, v') \in E$，事件（标签）$\alpha = syn(e)$。

由初始状态开始，通过反复地计算连续时间和离散迁移中的后继状态，我们可以得到混成自动机 A 的可达状态集合。令 S 为混成自动机 A 的一个状态集合，其连续时间和离散迁移后继分别表示如下：

1. **连续后继：**

$$Post_t(S) = \{(v, r') | \exists \delta \in \mathbb{R}_{\geq 0}, (v, r) \in S : (v, r) \rightarrow_\delta (v, r')\};$$

2. **离散后继：**

$$Post_d(S) = \{(v', r') | \exists \alpha, (v, r) \in S : (v, r) \rightarrow_\alpha (v', r')\}.$$

假设 $Init$ 为初始状态集合，则从该集合出发，其所有可达状态组成的集合记为 $Reach(Init)$, 是如下方程的最小不动点：

$$S = Post_t(Post_d(S)),\ Post_t(Init) \subseteq S$$

同样地，也可以从状态前驱的角度定义能够到达 S 的所有前驱状态集合，用符号表示为 $Reach^{-1}(S)$，更多相关论述可以查看 [10]。

可达状态 $Reach(Init)$ 的计算过程可用于前向分析，即：从初始状态 $Init$ 出发，反复地执行 $Post_t$ 和 $Post_d$ 计算后继状态，直到不动点产生，然后检查不安全的状态集合与 $Reach(Init)$ 的交集是否为空集。若为空集，则可知不安全状态不可达，系统是安全的，反之则可达，系统是不安全的。但是，对于很多复杂的混成系统，不动点的计算是不一定会终止的，这也是目前验证技术的不足之处。

类似地，后向分析的基本过程是：从不安全状态 Unsafe 出发，反复计算其连续前驱和离散前驱状态，直到发生不动点，然后检查初始状态集合与 $Reach^{-1}(Unsafe)$ 的交集是否是空集，若为空集，则系统是安全的，反之为不安全的。

3.3　仿真建模语言

这一节我们主要列出 Stateflow（状态图）建模的基本语法元素。Stateflow 是一种图形语言，其主要的组件是状态（State），外观类似我们在混成自动机中所用的矩形（模态）。为免引起混淆，我们将 Stateflow 中的状态也称为模态。

在 Stateflow 中每个模态有一个标签，用于标记其名称，名称之后可以接一

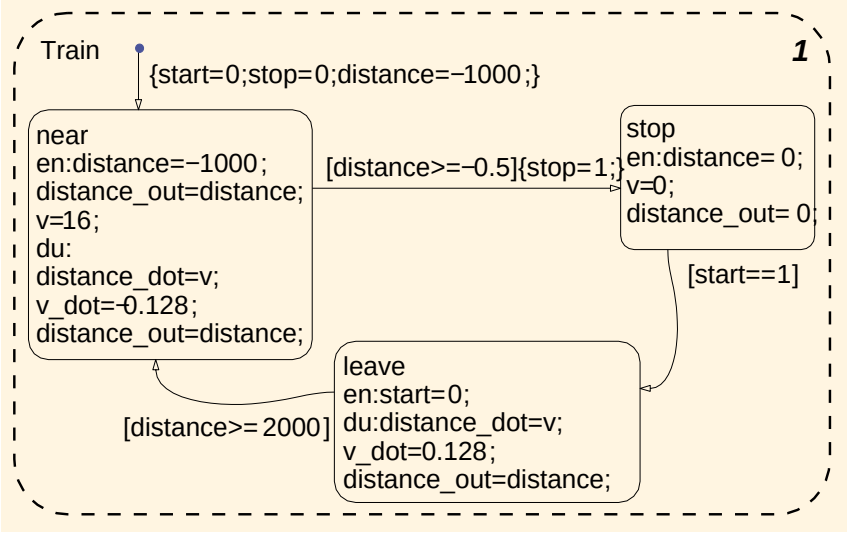


图 3.3: Stateflow 中的列车状态图。变量 *start* 和 *stop* 是列车与控制器状态图间的共享变量，是分别用于控制列车离站和到站的辅助变量，在控制领域通常也称这些变量为信号

个可选的斜线（/）。如果模态名之后有回车符，我们就可以省略这个斜线。例如，图3.3表示列车的状态图，初始模态为 near。在模态名下，声明了一系列动作（Action），每个动作前有一个关键字标签，如：en，du，其后由冒号作为分隔符。在 Stateflow 中，总共有 5 种类型的动作，我们依次介绍如下。

1. **Entry Action：** 关键字为 'entry' 或其简写为 'en'。如图3.3所示，在模态 near 中，有 3 个 entry 动作语句，分别是 '`distance = −1000`', '`distance_out = distance`' 和 '`v = 16`'，语句之间以分号相隔。每次进入 near 模态时，变量 *distance* 赋值为 −1000，变量 *distance_out* 赋值为 *distance*，以及将变量 v 置为 16。

2. **During Action：** 关键字为 'during' 或简写为 'du'。如图3.3所示，模态 near 有 3 个 during 动作语句。第一条语句 '`distance_dot = v`' 表示一个微分方程，即变量 *distance* 的一阶导数等于 v。第二条语句 '`v_dot = −0.128`' 也是一个微分方程，表示变量 v 的一阶导数为 −0.128。第三条语句 '`distance_out = distance`' 表示变量 *distance_out* 会一直等于变量 *distance* 的值，即随着 *distance* 变化而变化。当进入模态后，先执行 entry

动作，然后会执行 during 动作。during 动作的执行频率和 Simulink 的采样时间以及系统中的事件有关。during 动作一般用于对系统连续行为的建模和仿真。

3. **Exit Action：** 关键字为 'exit' 或简写为 'ex'。当系统的执行进入一个模态之后，在某一时刻需要退出该模态，则退出的时候 exit 动作会被执行。exit 动作声明与 entry 动作所用语法一致。

4. **On Event_Name Action：** 关键字中包含 'on' 和事件名（$event\_name$）。其主要用于表示当相应的事件发生时，所需要执行的动作语句。这类动作在本章中没有用到，因为 Stateflow 在连续时间状态图中并不支持此类动作，但可以用于离散时间的模型。

5. **Bind Action：** 关键字为 'bind'。主要用于变量写权限的指定（也可绑定事件）。当在一个模态中声明了该动作，则该模态及其子模态（子状态图，Stateflow 支持层次模型）可以对相应的变量进行修改，但其他模态只能进行读，不能进行写（赋值）操作。

在 Stateflow 中，模态之间存在迁移，每个迁移有一个标签字符串指定迁移的条件、动作等。一般地，迁移可能包含的信息有：事件、迁移条件、条件动作以及迁移动作。迁移的一般形式如下所示：

$$event[condition]\{condition\_action\}/transition\_action$$

其中，若在迁移中指定了事件（event），则当指定的事件触发时，迁移就可以执行，否则就不能执行而需等待该事件。若没有指定事件，则当迁移条件 *condition* 为真时，就可以执行迁移，否则就需等待该条件成立。当事件和迁移条件都被指定，则需要同时满足事件触发和迁移条件成立的要求才能执行迁移。条件动作 *condition_action* 在条件满足时被执行。迁移动作 *transition_action* 是在迁移的目标模态有效时，即当迁移可以执行且目标模态可以被激活时执行。

另外，初始迁移是一个比较特殊的迁移，如图3.3所示，在模态 near 上有一

个初始迁移，其从一个小点出发到达 near 的边缘，只包含条件动作（迁移条件默认为 *true*）：

$$\{\texttt{start} = 0; \texttt{stop} = 0; \texttt{distance} = -1000;\}$$

其中，变量 *start* 和 *stop* 均被初始化为 0，变量 *distance* 赋为 -1000，即列车的初始位置为站点左侧 1000 米的位置。

此外需要注意的是，在迁移条件中，逻辑合取运算符用两个 '&' 符号表示，即 $(x > 1)\&\&(x < 2) \equiv (x > 1) \wedge (x < 2)$，析取用 '||' 表示，即 $(x > 1)\ ||\ (x < -1) \equiv (x > 1) \vee (x < -1)$。更多细节可以查看 Stateflow 的用户手册[3]。

3.4 协同验证过程

本章中，混成系统的形式化验证是基于混成自动机的可达性分析来完成的。经典的基于形式化方法的工具（如：HyTech，PHAVer）可以提供参数分析以及丰富的分析命令，并生成错误跟踪信息。另一方面，使用 Simulink/Stateflow，我们可以受益于其丰富的原子组件库。Matlab 工具箱可以提供灵活友好的工具对仿真数据进行浏览和加工。其交互式仿真的特性，能够胜任复杂系统的建模和仿真任务。

然而，通过仿真，我们无法预知系统组件之间，以及系统与环境间所有可能的交互行为，同时混成系统的可能状态也是无限的，因而无法对所有可能的系统状态和行为进行仿真，进而无法确保系统的完全正确性。相反地，基于形式化方法的验证工具在一定程度上可以避免仿真的局限性，能够提供系统正确性的保证。可惜的是，在使用形式化工具过程中，我们发现作为工具输入的命令并不容易构造，若非工具的开发者，一般的使用者需要进行多次实验才能确定合适的命令和对应的形式化模型，因此，用形式化工具来对复杂的混成系统进行分析是一件比较耗时的工作。在我们的案例中，Simulink/Stateflow 中所用的仿真模型可以看作是 SpaceEx/PHAVer 中所用形式化模型的精化版本。仿真模型是较为接

[3] http://www.mathworks.com/help/pdf_doc/stateflow/sf_ug.pdf

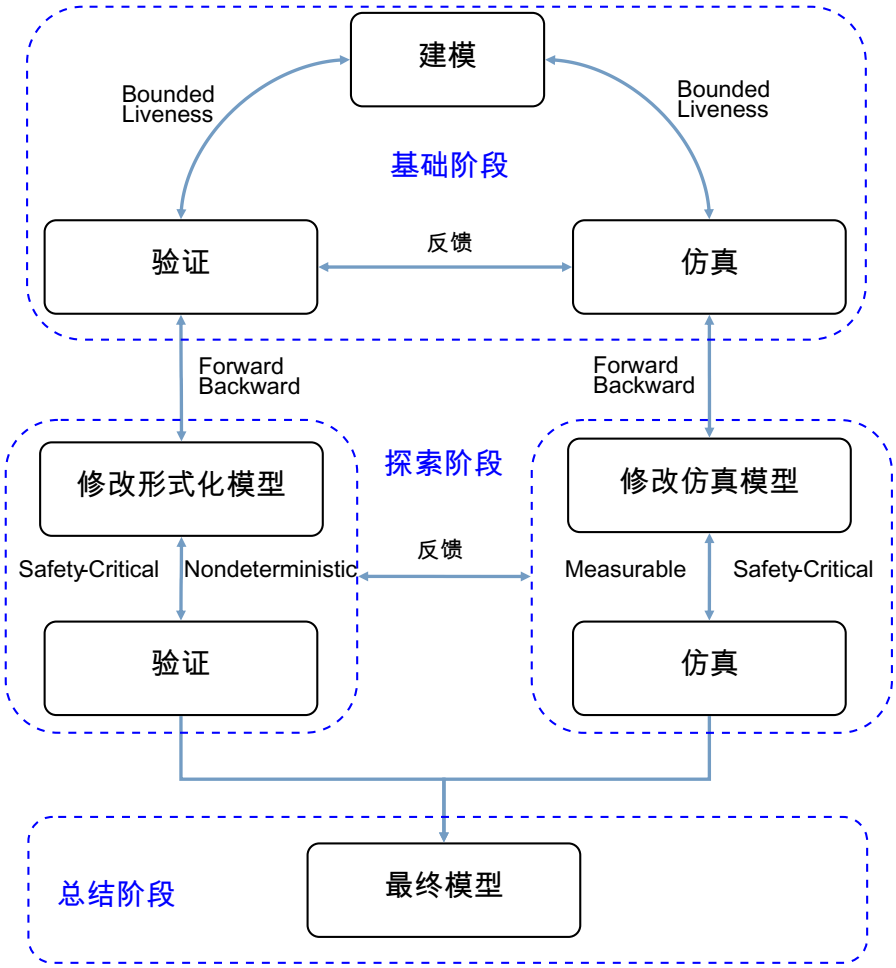


图 3.4: 反馈演进验证过程（FAV：Feedback-Advancement Verification）。包括：基础，探索和总结三个过程

近最终的系统实现的，而仿真模型与形式化模型间的关系可以通过构造合适的精化关系得以证明（如：时间仿真或时间轨迹包含等 [65, 94]）。此外，在某些特性上，两种模型是互补的（如：仿真的可交互性，结果的可视化与正确性的证明，参数求解等），它们可以从系统不同侧面对总体系统进行强化，并提供互补的分析手段，即验证与仿真。因而，通过整合形式化验证与仿真技术来对混成系统进行分析验证是可行的。在此，我们提出一个协同验证过程，称为反馈演进验证过程，如图3.4所示。这个验证过程包含三个阶段：基础阶段，探索阶段，以及总结阶段。下面，我们将对这三个阶段进行详细阐述。

基础阶段

以混成自动机和 Simulink/Stateflow 分别构建模型。在这个阶段，主要分析有界活性性质（如限定时间和验证工具迭代步骤等）和非安全攸关性质，并根据仿真和验证的结果对模型进行修正。由于混成自动机与 Stateflow 均从有限状态机演化而来，虽然两者所用语言不尽相同，但在某些结构和语法上是一致的。例如：两者都有模态的结构（从有限自动机中的状态演变而来，在 Stateflow 中也称为状态），还有两者在模态之间都设置了离散迁移动作。在连续行为描述方面，混成自动机中的连续流条件可以对应到 Stateflow 中的 during 动作，而且都以微分方程的形式来定义系统中的连续动态行为。比如，在混成自动机中，流条件 $\dot{v} = 1$，可以在 Stateflow 中用 during 动作 $\texttt{du} : \texttt{v\_dot} = 1$ 来描述。混成自动机中的连续流条件 $\dot{v} = dv$ 和不变式约束 $(-1 \leq dv \leq 0)$ 可以在 Stateflow 中用 during 动作 $\texttt{du} : \texttt{v\_dot} = \texttt{v\_shut}$ 表示，并且用一个开关组件来控制变量 v_shut 的值（参考图3.13中的 Simulink 模型）。

在这个阶段，我们的主要目标是以粗粒度的方式检查模型是否正确。我们关注的性质比较简单，比如：给定初始状态和最大时间，检查列车是否会停在一个站点，或者是否能够离开站点。安全攸关性质和其他复杂性质将在探索阶段分析。

探索阶段

在这个阶段，我们将结合形式化验证和仿真技术探究安全攸关性质或系统中复杂的交互场景。在基础阶段中，若在验证中发现问题我们可以修改模型，然后继续分析和验证，最终使得模型满足相关性质，或者因验证工具、计算资源等方面的限制而无法继续验证。如此，需要尝试在仿真模型中对所关心的性质进行分析。

对于仿真，因为正确性要求无法得以满足，因而最后也需要通过验证工具来进行检验。因此，系统模型的修正需要基于两类模型的分析结果，仿真和验证之间可以互相反馈分析结果和问题，从而优化各自的模型。比如，在我们的案例分析中，对两种模型（图3.15和3.16）的一致修正是通过引导图（如图3.5）来完成

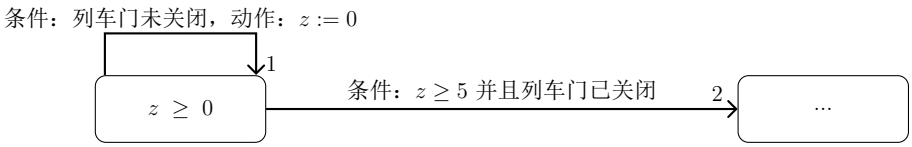


图 3.5: 引导图。

的。图3.5中含有约束条件 $z \geq 0$ 的模态是图3.15中模态 $about\_to\_close2$ 的一个抽象表示。引导图中的有向边表示离散迁移，类似地也包含有迁移条件和迁移动作的声明。当迁移条件成立时，会执行迁移动作，从而完成模态间的迁移。迁移有优先级，用一个数字表示，如图3.5中的 1 和 2，数字越小，相应迁移的优先级越高。例如，图中级别为 1 的迁移会先被检查，即每次检查时若发现列车门未关闭，则将变量 z 赋值为 0。若条件不成立，则检查优先级 2 的迁移是否可行，若此时变量 z 的值大于或等于 5 而且列车门是关闭的，则执行该迁移，系统的执行切换为新的模态。根据引导图，我们在图3.16中引入变量 c_1，只有当同步事件 $shut\_closed_1$（意味着列车门已关闭）触发后，c_1 才被设置为 0，如此，事件 $close_2$（关闭屏蔽门）只能在事件 $shut\_closed_1$ 后才能被触发。这样就实现了列车门与关闭屏蔽门这两个事件的优先关系。同时，通过 $z \geq 5$ 的条件，可以保证这两个事件的时间间隔是大于等于 5 的。总之，引导图是对模型修改部分的一种简单抽象，可以辅助我们在探索阶段实现仿真模型和验证模型的一致修改。

总结阶段

到这一阶段，仿真和验证模型都已分析完毕，那么就可以选择其中合适的模型进行下一步的开发。一般情况下，可能会选择仿真模型，然后做控制器代码生成、优化等相关工作。具体如何选择，就要看具体的需求，在此我们并不深入展开讨论。

总之，探索阶段主要分析安全攸关的性质并提供一个正确的模型。在整个过程中，形式化验证和仿真可以通过分析结果的反馈而互相影响，使得模型得以逐步优化。此外，两类模型在表达能力和语义方面存有一定的分歧，因而维护两种模型的一致性对于复杂的混成系统来说是一个挑战。在反馈演进分析过程中，我

们倾向于以循序渐进的方式来构造和优化模型。每次模型修正，我们只专注于模型中很小的一个部分，如：只涉及一个模态，一个控制迁移或者仅仅是一个条件表达式。比较乐观的是，Simulink/Stateflow 可以提供快速和可视化的仿真结果反馈，以支持我们对模型的微调。

在后续章节中，我们将遵循反馈演进验证过程来对地铁控制系统进行建模和分析验证。首先，我们阐述本案例系统的相关需求，然后，建立仿真和验证的模型并在 SpaceEx/PHAVer 和 Matlab 工具箱中进行验证和仿真。在仿真中，我们发现模型无法满足安全性质，修改模型之后我们通过在验证工具中分析，最终使得模型满足安全性质。最后，我们对防碰撞系统进行分析，通过 SpaceEx/PHAVer 的验证，我们发现了在仿真中未揭示的列车碰撞场景，并对结果进行了相应分析。

3.5 屏蔽门系统分析

3.5.1 系统需求

在这一节中，我们主要关注的是屏蔽门系统。屏蔽门在世界各地的地铁站中应用广泛，屏蔽门可以在站台与列车之间建立一道防线，利用屏蔽门可以防止乘客因自杀或者活塞效应 [22] 而导致的从站点意外跌落等类似情况的发生。

屏蔽门系统属于安全攸关系统。系统中一个很小的设计缺陷可能导致严重事故。例如，2007 年 7 月 15 日下午，上海某地铁站，有一乘客极力想挤入拥挤的地铁车厢，由于车厢内人太多，最后未能进入车厢。当时列车门和屏蔽门几乎同时关闭，导致该乘客处于列车门与屏蔽门之间的狭小空间内，列车启动后导致其因列车挤压而跌落站台，最终发生意外。现在，我们考虑如下三种屏蔽门关门方案：

1. **同时关闭：** 屏蔽门和列车门同时关闭。这种方案会导致乘客被卡在列车和屏蔽门之间的情况，因此，并不适合在屏蔽门系统中采用。

2. **先关闭屏蔽门：** 先关闭屏蔽门，然后再关闭列车门。当车厢内人过多时，会存在部分乘客无法挤入车厢情况，因此，关闭屏蔽门后，乘客无法回到站点，这也会导致危险事故发生。

3. **先关闭列车门：** 先关闭列车门，然后关闭屏蔽门。而且，在列车门关闭后，等待一定时间，然后关闭屏蔽门，这种方案可以让乘客返回站台，相对前两种方案较为安全。

当然，还可以采取在列车与屏蔽门之间的空隙处部署传感器，通过感知乘客位置来提升屏蔽门系统的安全性，但在本章中，我们只专注于第三种方案的分析。

因此，在我们的案例中，最重要的需求是乘客在列车启动之时，不会被夹于列车门和屏蔽门之间。在这里，我们将此需求细化为：当列车门关闭后，会有一段等待时间，在这段时间内乘客可以返回站台，当等待时间到达时，关闭屏蔽门。

需要注意的是，我们无法强制乘客在等待时间内必须返回站台，因此，我们只预留相应的时间，这样虽无法绝对地避免事故，但在未采用其他设施和技术的前提下，相比其他两种方案来说是较为安全的。现在，令 T_1 为列车门关闭的时刻点，T_2 为屏蔽门开始进行关闭的时刻点，则上述需求可以用如下简单谓词表示：

$$(T_2 - T_1) \geq C \ \wedge \ C > 0$$

其中 C 是一个常量，代表等待时间，在我们的案例中，C 被设置为 5 秒钟。

3.5.2　初步模型

整个系统模型将包含四个部分：列车、列车门、屏蔽门以及控制器。首先，我们会介绍系统的形式化模型，即混成自动机模型，然后介绍用于仿真的模型，即 Simulink/Stateflow 模型。

混成自动机模型

我们使用四个混成自动机来对屏蔽门系统进行形式化建模。列车自动机（图3.2）主要用于表示列车的运动特性。列车门和屏蔽门自动机（图3.7和3.8）分别表示各自的开启和关闭行为。控制器自动机（图3.9）在上述三者间起到仲裁协调的作用。

如图3.7有四个模态。模态 $open$ 和 $closed$ 分别表示列车门处于完全开启和关闭的状态。模态 $part$ 和 $shut$ 分别表示列车门正在打开和关闭。每个列车门都有两块门扉如图3.6所示（当然，对于屏蔽门也一样，在此恕不赘述）。变量 y_1 表示左侧门扉与右侧门扉的距离，在此案例中，我们设定最大距离为 2 米。如此，当 $y_1 = 2$ 时，我们可以知道门处于完全打开的状态。当 $y_1 = 0$ 时，门是完全关闭的。最初，列车门是关闭的，如果列车门收到 $open_1$ 命令，则列车门会跳转到模态 $part$，门会以 1 m/s 的速度打开，直到 $y_1 = 2$。当列车门完全打开后，列车门自动机会跳转到模态 $open$，在此模态，y_1 的值不会改变，若此时收到 $close_1$ 命令，则自动机跳转到模态 $shut$，列车门会被关闭。由于门在关闭过程中可能会受乘客或者异物影响而卡住，因此，关门的速度是非确定的，在模态 $shut$ 中，关门速度 dy_1 在区间 $[-1, 0]$ 内变化。当速度 $dy_1 = 0$ 但门未完全关闭（$y_1 > 0$）时，需要重新打开门，然后再尝试关门。变量 c_1 在收到 $close_1$ 命令时被重置为 1，因此，可以通过判断 c_1 的值来决定在门重新打开后是否继续关门操作。如果列车门完全关闭了，则自动机跳转到初始模态 $closed$ 中，如此，完成了一次循环操作。

屏蔽门自动机与列车门自动机类似。模型的区别在于二者使用的变量名称和同步标签名不同。因此，我们不再详细解释。图3.9是控制器的模型。控制器是列车与列车门以及屏蔽门的协调者。刚开始时，控制器处于模态 $idle$，当收到列车发送的 $near\_stop$ 信号（表示列车停在了站点）时，控制器跳转到模态 $about\_to\_open_2$ 中。等待 5 秒钟后，控制器发送 $open_2$ 命令给屏蔽门，如此，屏蔽门进入模态 $part$，执行打开屏蔽门的任务，同时控制器跳转到模态

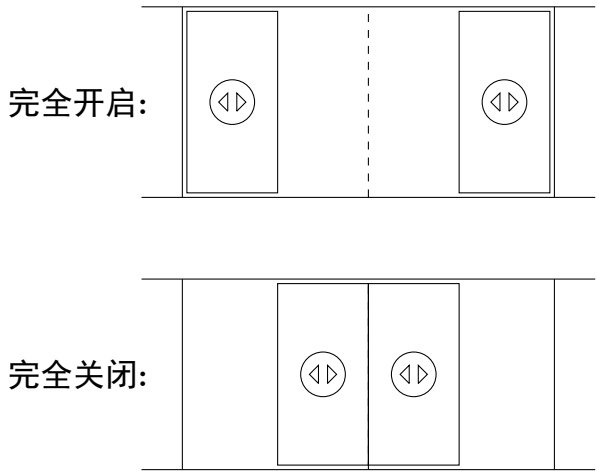


图 3.6: 屏蔽门示意图。两侧门扉可以左右滑动，第一个子图表示完全开启的情况，即左侧门扉向左移动到最左边，而右侧门扉移动到最右边；第二个子图表示完全关闭的情况，即两侧门扉相向移动最后在中间相遇

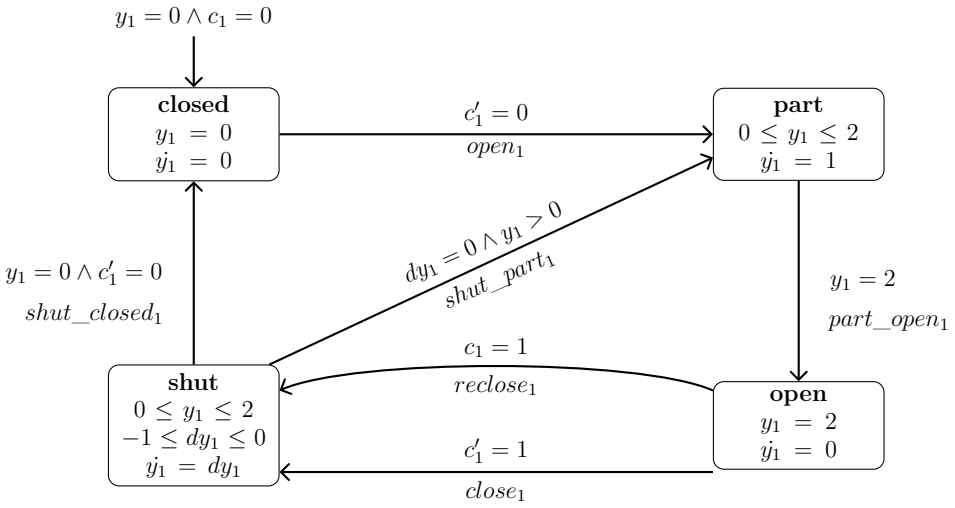


图 3.7: 列车门自动机。起初，列车门是关闭的（模态 *closed*）。连续流条件 $\dot{y}_1 = 1$ 表示开门的速度为 1 m/s。在模态 *part* 中，我们以 1 m/s 的速度打开列车门。当变量 y_1 等于 2 时，列车门完全开启

$about\_to\_open_1$，同样等待 5 秒后，发送 $open_1$ 命令给列车门，列车门也被打开。控制器跳转到模态 $open_1$ 中，等待 6 秒后，进入模态 $ring$，即执行响铃操作（我们在此没有细化该操作），3 秒后跳转到模态 $about\_to\_close_1$。又等待 5 秒，控制器发送 $close_1$ 命令给列车门，列车执行关门动作，同时控制器跳转到模态 $about\_to\_close_2$，在等待 5 秒后发送 $close_2$ 命令给屏蔽门，屏蔽门执行关门操作。此时，控制器跳转到模态 $start$ 中，在 5 秒后发送 $stop\_leave$ 命令给列

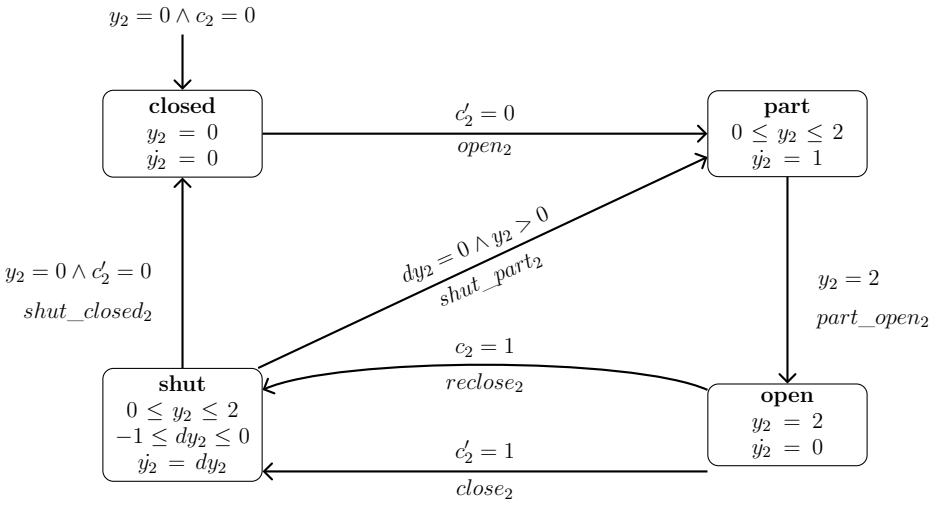


图 3.8: 屏蔽门自动机。起初，屏蔽门是关闭的（模态 *closed*）。连续流条件 $\dot{y}_2 = 1$ 表示开门的速度为 1 m/s。在模态 *part* 中，我们以 1 m/s 的速度打开屏蔽门。当变量 y_2 等于 2 时，屏蔽门完全开启

车，列车即开始离开站点。最终，控制器回到初始模态 *idle* 中，一个基本的控制过程完成。

状态图模型

在这小节中，我们使用 Simulink/Stateflow 对屏蔽门系统进行建模。与混成自动机模型相对应，我们的模型也由四个部分组成。图3.3展示的是列车的状态图。列车门和屏蔽门的状态图分别如图3.10和3.11所示。控制器的状态图如图3.12所示。

在图3.13中，我们使用两个手动开关模拟关门速度，其中一个开关用于控制列车门的关门速度 *v_shut*1，另一个用于控制屏蔽门的关门速度 *v_shut*2。两个开关均可以在 -1 和 0 之间切换，每次切换，都会改变相应变量的取值，从而影响模型的仿真结果。变量 *v_shut*1 和 *v_shut*2 是 Stateflow 模型（作为 Simulink 模型的子系统，如图3.13所示）的输入变量，分别被列车门和屏蔽门状态图所用。此外，Stateflow 模型有三个输出变量，分别是 *y*1*_out*，*y*2*_out* 以及 *distance_out*，分别表示列车门、屏蔽门的状态以及列车的位置。示波器（scope）用于显示仿真结果。

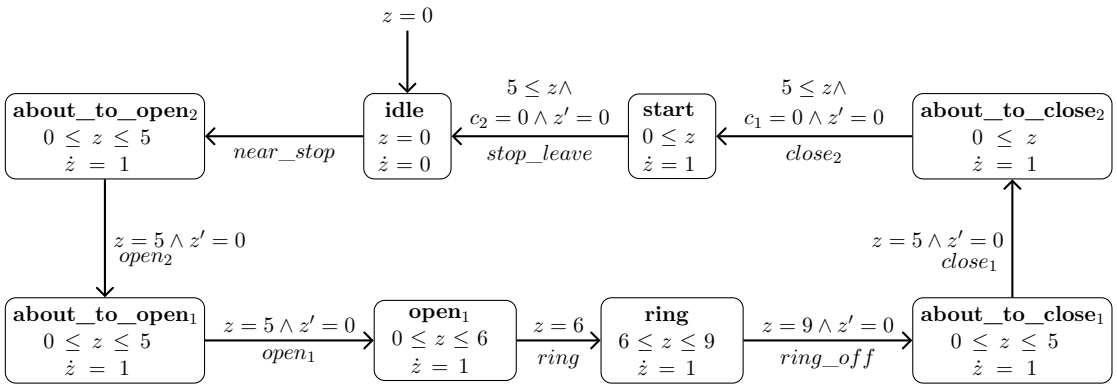


图 3.9: 控制器自动机。其中变量 z 是一个时钟变量，其时间导数为 1（$\dot{z} = 1$）。变量 c_1 是控制器与列车门之间的共享变量，c_2 是控制器与屏蔽门之间的共享变量。同步标签 *near_stop* 是控制器与列车自动机间的共享标签

在图3.3中，*distance* 是一个局部变量表示列车与站点间的距离。变量 v 也是局部变量，表示列车的速度。表达式 *distance_dot* 表示变量 *distance* 的一阶时间导数，因此其值与变量 v 是相等的。

类似地，*v_dot* 表示 v 的导数，也就是列车的加速度。此外，‘*en* :’ 是 entry 动作的前缀，其后紧接的语句在进入模态时首先会被执行。‘*du* :’ 是 during 动作的前缀，其后紧接的语句用于声明变量的导数和更新约束，在每一次采样时，如果没有迁移可以发生，则相应的变量更新操作会被执行（基于微分方程求解或者直接赋值操作）。

在 Stateflow 中，一个状态图的局部变量可以被其子状态图读取和修改，因此，局部变量可以在子状态图间共享访问。例如，变量 *stop* 可以在列车状态图（图3.3）和控制器状态图（图3.12）中共享访问。

当列车状态图中从模态 *near* 到模态 *stop* 的迁移发生时，变量 *stop* 会被赋为 1。如此，控制器状态图中的条件 $[stop == 1]$ 就会成立，从模态 *idle* 到模态 *about_to_open*2 的迁移就会被激活。类似地，变量 *start* 也是列车状态图与控制器状态图的共享变量。变量 *open*1 和 *close*1 是列车门状态图与控制器状态图之间的共享变量。

变量 *close*1 与列车门自动机（图3.7）中的变量 c_1 用途相同，即二者均是为了控制列车门的关闭而设置的辅助变量。变量 *open*1 用于控制列车门的开启，当

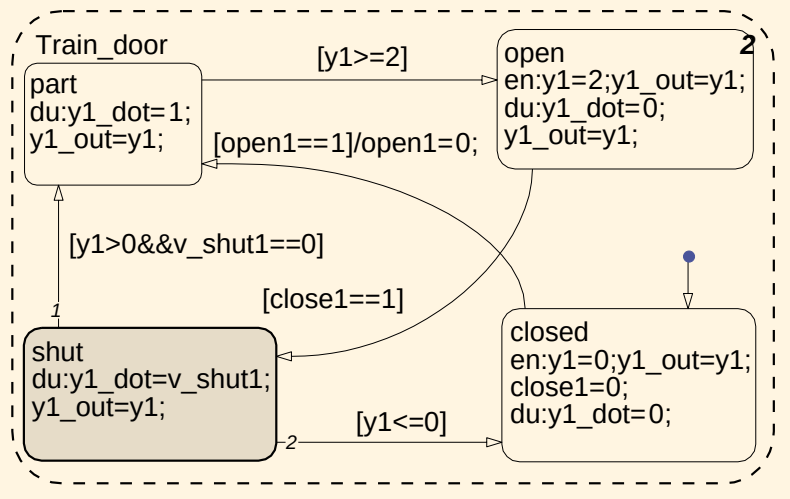


图 3.10: 列车门状态图。变量 $open1$ 和 $close1$ 是列车门和控制间的共享变量

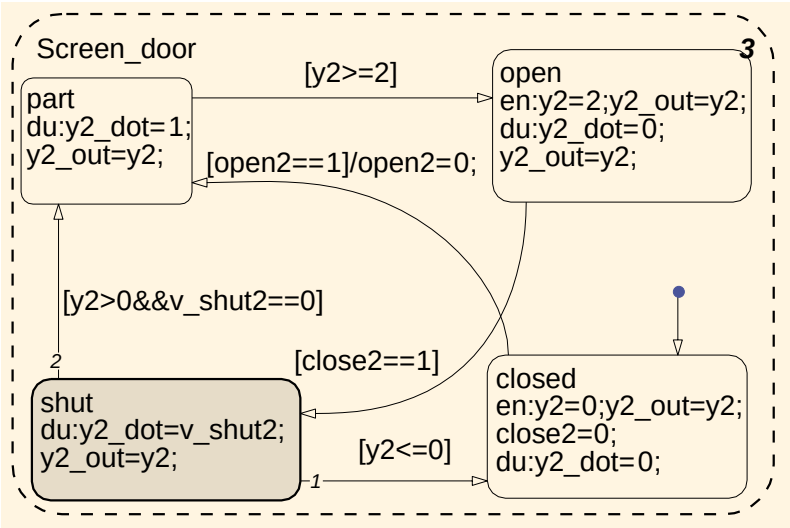


图 3.11: 屏蔽门状态图。变量 $open2$ 和 $close2$ 是屏蔽门和控制器间的共享变量

其值为 1 时，我们需要打开列车门。变量 $open2$ 和 $close2$ 有上述类似功能，区别在于，它们是用于控制屏蔽门的开启和关闭的辅助变量。

值得注意的是，我们在控制器状态图中使用了如下谓词，用于判断列车门是否处于关闭模态：

$$in(Train\_door.closed) == 0$$

其中，$Train\_door$ 是列车门状态图（图3.10）的名称，$closed$ 是列车门状态图中的一个模态（表示门完全关闭）的名称。

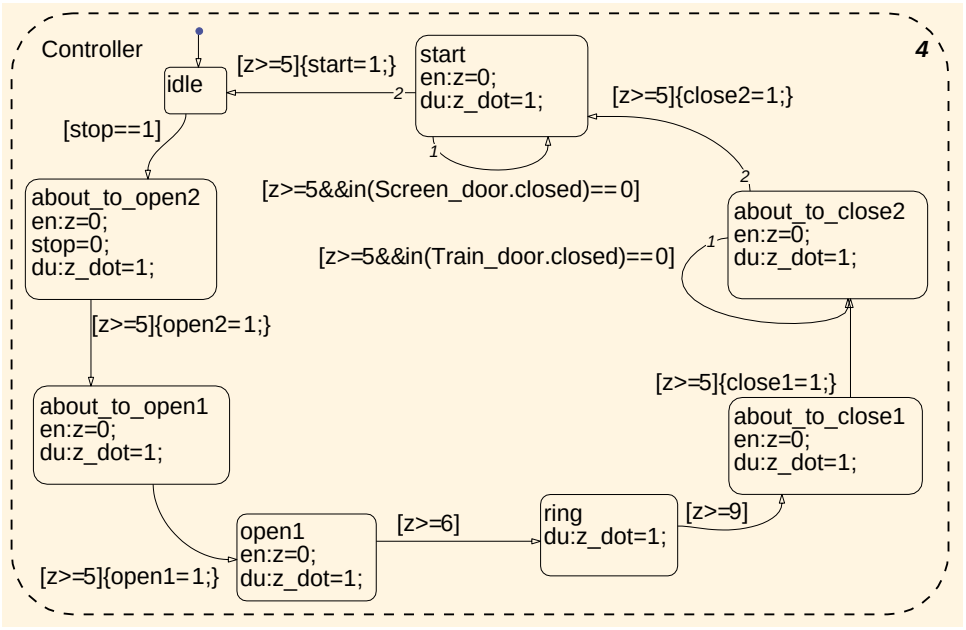


图 3.12: 控制器状态图。变量 z 表示时钟，谓词 $in(\cdot)$ 用于检查屏蔽门或列车门是否已关闭

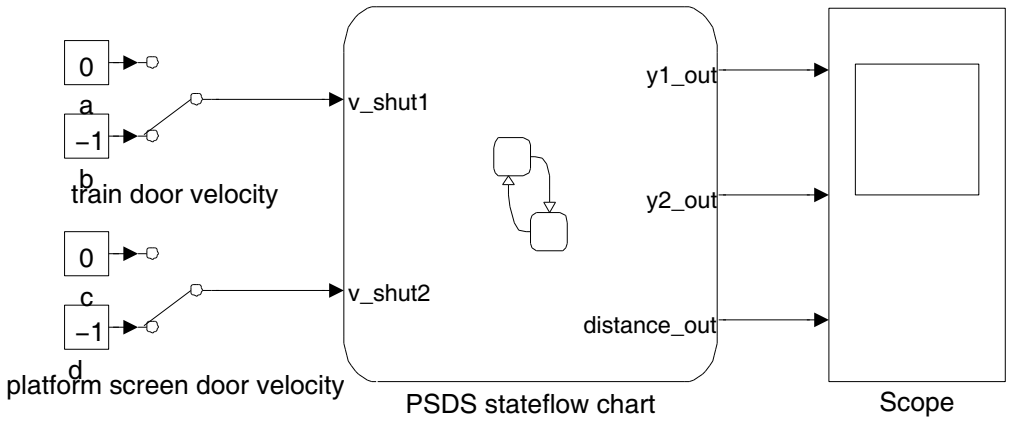


图 3.13: 屏蔽门控制系统的 Simulink 模型。左边有两个手动控制开关（Manual Switch），用于控制列车门和屏蔽门的关门速度。中间是一个子系统，表示屏蔽门系统的 Stateflow 模型。$y1\_out$ 是一个输出变量，表示列车门的状态，即列车门左右两边的距离，$y1\_out$ 表示屏蔽门的状态，$distance\_out$ 表示车的位置

若上述谓词为真，则列车门未处于完全关闭模态，因此，我们在模型中加入了自反迁移，如图3.12所示，模态 $about\_to\_close2$ 和 $start$ 有自反迁移。所以，在等待 5 秒时间到达时，若相应的门没有关闭，控制器将重置时钟（变量 z）并继续等待直到列车门或屏蔽门完全关闭。

3.5.3 验证与仿真

对于一个混成系统 H，给定性质 P，形式化验证回答的问题是，系统 H 是否满足性质 P。在对安全攸关系统的分析中，我们想要检查的是，系统不会到达不安全的状态。然而，对于一个系统，我们不会一开始就验证安全性质。相反地，我们首先会检查有界活性和一些简单的性质，然后才考虑安全性质。下面，我们将通过形式验证和仿真来分析屏蔽门系统的四个性质。

验证

我们使用验证工具 PHAVer 来对混成自动机模型进行检验。PHAVer 支持在混成系统中使用导数分段有界的约束，并提供了对仿射连续行为的形式化验证，对于可达状态集的计算，提出动态上逼近算法以实现精确计算。在技术层面，通过使用 PPL [16] 和 GMP（GNU multiple precision arithmetic library [57]），PHAVer 在验证过程中避免了运算溢出错误的出现并提供准确的精度控制，在一定程度上保证了验证工具的鲁棒性和正确性。下面我们对验证过程进行详细阐述。

在 PHAVer 中，我们可以对多个混成自动机的并发组合进行声明。下面这个语句表示四个混成自动机的并发组合，符号 & 是组合连接符：

```
sys = controller & traindoor & screendoor & train.
```

其中，*sys* 代表组合之后的混成自动机。因此，屏蔽门系统就可用 *sys* 来指代。语句中的 *controller* 等依次代表控制器自动机（图3.9）、列车门自动机（图3.7）、屏蔽门自动机（图3.8）和列车自动机（图3.2）。

PHAVer 中的符号状态由自动机中的控制模态和线性公式组合而成，例如，符号状态 $stop\ \&\ x == 0$ 表示列车处于模态 *stop*，同时变量 x 的值等于 0。符号 & 等价于逻辑与操作符（$\wedge$）。

对于我们的屏蔽门系统，其初始状态可以用如下符号状态表示：

```
idle ∼ closed ∼ closed ∼ near &
z==0 & x==-1000 & v==16 &  y1==0 &
y2==0 & close1_flag==0 & close2_flag==0
```

其中，第一行分别是四个自动机的模态名称，*idle* 是控制器自动机的初始模态，其后的两个 *closed* 分别是列车门和屏蔽门自动机的中的初始模态，*near* 是列车自动机的初始模态。此外 $close1\_flag$ 代表列车门自动机中的变量 c_1，$close2\_flag$ 代表屏蔽门自动机中的变量 c_2。下面，我们关注如下四个性质是否可满足，即检查性质对应的状态是否可达。

1. **停靠和出发：** 列车能够在站点停靠，也可以顺利地离开站点。如下符号状态表示列车处于停靠或离开站点的模态：

```
sys.{
  $ ∼ $ ∼ $ ∼ leave & True,
  $ ∼ $ ∼ $ ∼ stop & True
}.
```

其中，表达式 $ ∼ $ ∼ $ ∼ leave 表示所有可能的模态组合情况，但指定列车自动机的模态为 *leave*（表示列车离开站点）；同理， $ ∼ $ ∼ $ ∼ stop 表示列车处于模态 *stop*，但其他自动机可以取任意模态。标识符 True 表示所有可能的变量取值情况。

在 PHAVer 中，性质是通过可达性分析来验证的。PHAVer 提供了相应的命令支持可达性分析。例如，命令 sys.reachable 返回系统 sys 所有可达状态的集合。如果，s1 和 s2 分别代表两个状态集合，则命令：

```
s1.intersection_assign(s2)
```

表示取 s1 与 s2 的交集，并把结果赋给 s1。下面是检查状态集合 s 中的状态是否可达所用到的命令：

```
as=sys.reachable;
as.intersection_assign(s);
echo "Reachable states in s:";
as.print;
echo "Reachable states of s empty?";
as.is_empty;
```

其中，第一行 as 指向系统所有可达状态的集合，第二行计算 as 与状态集合 s 的交集，第三行在命令行中打印一行字符串（“Reachable states in s:"），第四行打印 s 中所有可达状态（以符号状态的形式），第六行打印交集是否为空的情况。

利用 PHAVer 的可达性分析，我们得出的结论是列车可以停靠在站点，也可以顺利地离开站点。

2. **响铃：** 在响铃时，列车门和屏蔽门都是处于开启模态。考虑如下符号状态：

```
sys.{ ring ~ $ ~ $ ~ $ & True }.
```

其中，表达式 ring ~ $ ~ $ ~ $ 表示控制器自动机处于模态 *ring* 中，但其他自动机可以处于任何模态。通过上述状态与系统所有可达状态的交集，我们可以检查，响铃时列车门和屏蔽门是否都已开启。PHAVer 给出的结果以符号状态形式表示如下：

```
controller ~ traindoor ~ screendoor ~ train.{
  ring ~ open ~ open ~ stop &
  6<=z<=9 & x==0 & y2==2  & y1==2 &
  close2_flag==0 & close1_flag==0
}.
```

可以看出，列车门和屏蔽门都是在模态 *open* 中。因此，我们可以确信列车门和屏蔽门在响铃时均是完全开启的。

3. **操作有序：** 列车门需要先关闭，然后关闭屏蔽门。在列车门进行关闭时，屏蔽门需要保持开启状态。现在考虑如下符号状态：

```
sys.{ about_to_close2 ~ $ ~ $ ~ $ & True }.
```

其中，表达式 about_to_close2 ~ $ ~ $ ~ $ 表示控制器所处的模态为 *about_to_close*2，而其他自动机可以处于任意模态中。当控制器处于模态 *about_to_close*2 中时，控制器已经发出关闭列车门的命令 $close_1$。通过将系统所有可达状态的集合与上述符号状态做交集，可以知道此时屏蔽门会处于何模态中。PHAVer 给出的验证结果为：

```
1    controller ~ traindoor ~ screendoor ~ train.{
2        about_to_close2 ~ shut ~  open ~ stop &
3        -1 <= dy1 <= 0 & 0 <= y1 <= 2 & x == 0 &
4        y2 == 2 & close2_flag == 0 &
5        close1_flag == 1 & z + y1 >= 2,
6        about_to_close2 ~ open ~  open ~ stop &
7        x == 0 & y2 == 2 & y1 == 2 &
8        close2_flag == 0 & close1_flag == 1 & z >= 0,
9        about_to_close2 ~ closed ~  open ~ stop   &
10       x == 0 & y2 == 2 & y1 == 0 &
11       close2_flag == 0 & close1_flag == 0 & z >= 2,
12       about_to_close2 ~ part ~  open ~ stop &
13       0 < y1 <= 2 & x == 0 & y2 == 2 &
14       close2_flag == 0 & close1_flag == 1 &
15       z + y1 >= 2
16   }.
```

从上述结果可以看出，屏蔽门始终是处于开启模态 *open* 中的。

此外，当控制器处于模态 *start* 中时，列车门必须是关闭的。控制器处于模态 *start* 的相应的符号状态可以表示为：

```
sys.{ start ~ $ ~ $ ~ $ & True }.
```

同样地，通过可达性分析，我们也可以得出相应的结果，即列车门的确是

关闭的。

4. **操作限制：** 在列车未停靠在站点时，不应该有列车门的操作。因而，如下符号状态是不可达的：

```
sys.{
    $ ~ open ~ $ ~ $  & x>0,
    $ ~ part ~ $ ~ $  & x>0,
    $ ~ shut ~ $ ~ $  & x>0,
    $ ~ open ~ $ ~ $  & x<0,
    $ ~ part ~ $ ~ $  & x<0,
    $ ~ shut ~ $ ~ $  & x<0
}.
```

通过 PHAVer 的可达性分析结果可知，列车门始终是处于模态 *closed* 中的。

总之，对于本文讨论的屏蔽门系统模型，通过 PHAVer 的验证，上述四个性质均得以检验，期望的性质得以满足。

仿真

现在，我们对在3.5.2节中建立的屏蔽门系统状态图模型进行仿真分析。仿真的结果可以用 Simulink 的示波器（scope）组件来查看。图3.14a显示的是列车从模态 *near* 到模态 *leave* 的一趟常规运行，包含列车门和屏蔽门的开启和关闭过程。横坐标表示时间，以秒为单位，纵坐标表示相应变量的取值。

图3.14a中，$y1$ 代表列车门的状态变化情况，$y2$ 代表屏蔽门的状态变化情况。变量 *distance* 的变化表示列车的运行情况，*distance* 的值等于列车和地铁站的距离，以米为单位。变量 *distance* 的取值范围可用区间 $[-1000, 2000]$ 来表示。当列车停在站点时，$distance = 0$。列车到站后，变量 $y2$ 的值会变为 2，这表明屏蔽门被打开。然后，$y1$ 的值也变为 2，这样列车门也被打开。一段时间后，列车门关闭，$y1$ 的值回到 0，之后屏蔽门也关闭，$y2$ 的值也回到 0。可以看出，列车门在屏蔽门之前关闭，我们关心的第三个性质是满足的。

在图3.14a中，我们选择的仿真时间总共是 350 秒：从仿真结果可以知道，列车可以停靠在站点也可以离开站点，因此我们所关心的第一个性质得以满足。此外，变量 $y1$ 和 $y2$ 只在列车停在站点时才有变化，在当列车离开或者未到站点时，没有发生变化，所以第四个性质是满足的。对于第二个性质，可以在 Simulink 调试时设置断点然后检查状态图的模态情况得以确认。

前面提到过乘客可能会被夹在列车和屏蔽门之间的情况，在此，通过仿真技术，我们尝试重现该场景：首先，我们在 Simulink 中启动调试器，在关闭列车门时，$y1$ 的值会逐渐减小。此时，我们触发手动开关，使得列车门的关闭速度从 -1 变成 0，这样列车门会被重新打开，然后再关闭。图3.14c展现的是上述仿真调试的结果。可以看到，屏蔽门几乎是在列车门关闭之后就开始关闭的。对比图3.14b和图3.14c，可以发现，图3.14b中列车门和屏蔽门关闭操作之间存在时延，而图3.14c中，屏蔽门却没有相应的等待时间。

因此，若当屏蔽门关闭时还有乘客处于列车和屏蔽门之间，则乘客将被夹于列车和屏蔽门之间而无法返回站台。所以，在模型中应该考虑列车门和屏蔽门关闭操作的时延条件的满足，使屏蔽门在列车门关闭之后必须等待一定时间然后才能关闭。因此，需要对模型进行优化。

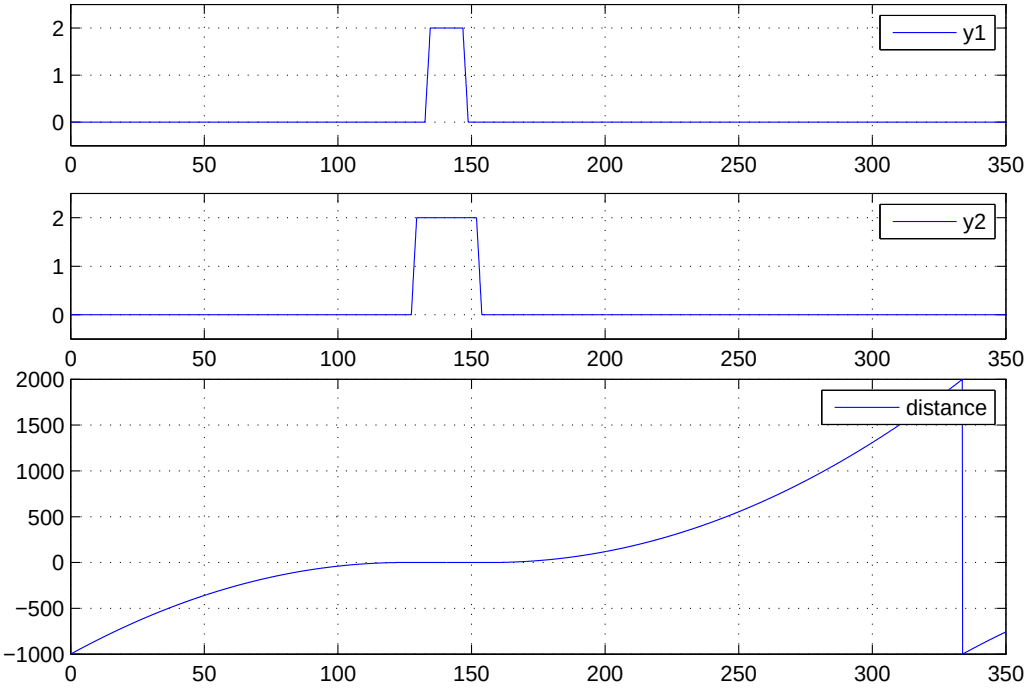
3.5.4　模型优化

在这一节，我们将对控制器模型进行修正，以满足时延条件。首先，我们在状态图模型中进行修改并重新仿真，然后对混成自动机模型进行相应的优化。

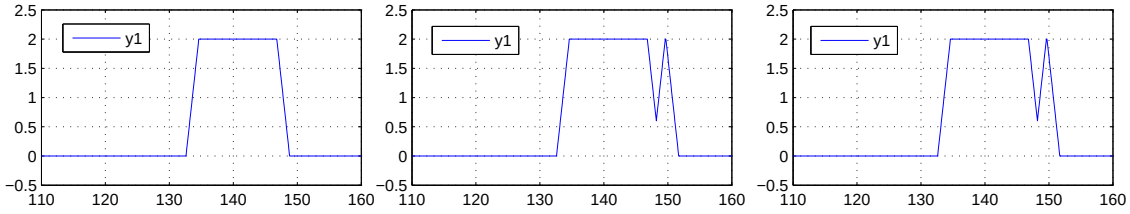
仿真模型优化

首先，让我们更深入地分析下导致乘客被夹于列车和屏蔽门之间的原因。我们注意到控制器状态图3.12的模态 $about\_to\_close2$ 的自反迁移含有如下迁移条件：

```
z>=5 && in(Train_door.closed)==0.
```



(a) 屏蔽门系统常规运行情况



(b) 正常仿真结果

(c) 异常仿真结果

(d) 模型修正后仿真结果

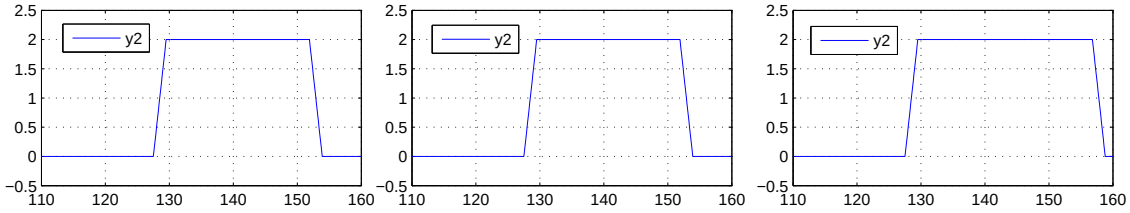


图 3.14: 屏蔽门系统仿真结果。图中变量 $y1$ 表示列车门左右两边门扉的距离变化情况，当 $y1 = 2$ 时，列车门完全开启；$y1 = 0$，则列车门是关闭的。屏蔽门的状态由变量 $y2$ 体现，也是表示其左右两边门扉的距离。变量 $distance$ 表示列车与站点间距离的变化情况，若 $distance = 0$，则列车停止在站点位置。从图 b 可以看出，正常情况下，屏蔽门是在列车门关闭之后，等待一定的时间后才关闭屏蔽门的，但在图 c 中，发生了屏蔽门在列车门关闭之后立即开始关闭的情况。图 d 是对模型进行优化后的仿真结果，屏蔽门会等待一定时间然后再关门

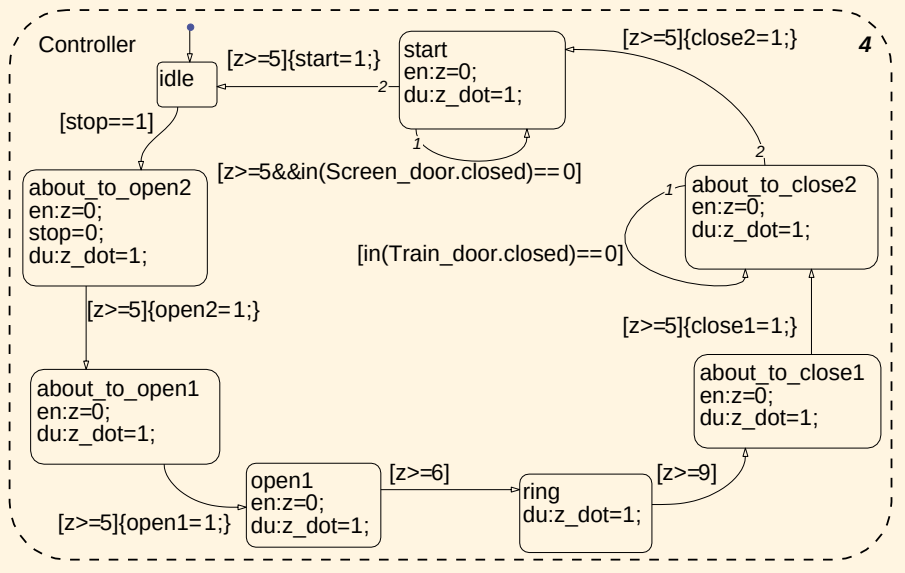


图 3.15: 修正后正确的控制器状态图

其中，表达式 $in(\cdot)$ 可以取值为 1 或者 0。如果列车门处于关闭状态，则表达式 $in(Train\_door.closed)$ 取值为 1，谓词 $(in(Train\_door.closed) == 0)$ 为假，否则若列车门未关闭的，则 $in(Train\_door.closed) = 0$，谓词 $(in(Train\_door.closed) == 0)$ 为真。

考虑如下情形：如果列车门在时钟 $z = 4.9$ 的时候刚好关闭，此时谓词 $(in(Train\_door.closed) == 0)$ 为假。那么，当 $z = 5$ 时，从模态 $about\_to\_close2$ 到模态 $start$ 的迁移会触发，同时屏蔽门会开始关闭，这样在列车门完全关闭的时刻点与屏蔽门开始关闭的时刻点相差只有 0.1 秒，这个时延远小于我们预设的 5 秒钟。若将条件 $z >= 5$ 从自反迁移的条件中删除，则在迁移条件中仅仅检查列车门是否关闭，只要列车门没有关闭，就触发自反迁移，然后重置时钟变量 z 为 0，如此可以使得时延足够大。新的状态图如图3.15所示。重新仿真的结果（图3.14d）表明，时延条件在新的模型中是满足的。当然，如果考虑 Simulink 的采样时间因素，实际的时延可能是一个非常接近 5 但会略小于 5 的时间。这种情况，可以通过修改从模态 $about\_to\_close2$ 到模态 $start$ 的迁移条件就可以避免，比如将 $z >= 5$ 改为 $z >= 6$。

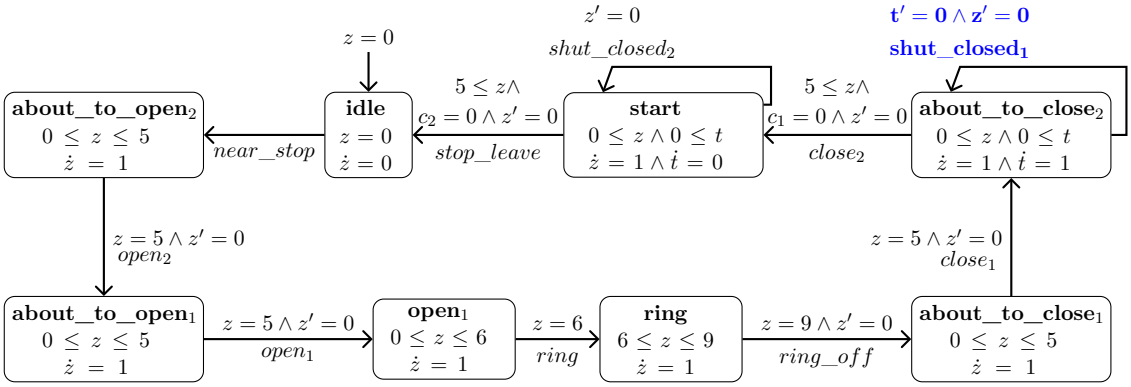


图 3.16: 修正后正确的控制器自动机。同步标签 $shut\_closed_1$ 和模态 $about\_to\_close_2$ 中的自反迁移是为了时间间隔需求而加入的

基于仿真的分析无法保证模型的绝对正确，因而，我们在下一节中使用验证工具 PHAVer 来检验修改后的混成自动机模型是否能够满足时延需求。

验证模型优化

对于控制器自动机（图3.9），也存在时延问题。同样地，假设 $z = 4.9$ 时，列车门完全关闭，则变量 c_1 会被重置为 0，那么当 $z = 5$ 时，从模态 $about\_to\_close2$ 到模态 $start$ 的迁移条件会成立，因此，屏蔽门就开始关闭，这就导致了与仿真模型相同的问题。为了方便地检验时延性质，我们在模态 $about\_to\_close2$（图3.16）中引入时钟变量 t，当模态从 $about\_to\_close2$ 迁移到 $start$ 时，我们不改变 t 的值，因此，我们可以通过 t 的值来检查时延条件是否满足。

同时，我们还在模态 $about\_to\_close2$ 上加入了与列车门（图3.7）同步的自反迁移，当列车门完全关闭时，通过该同步迁移将 t 和 z 重置为 0，这样时钟重新开始计时，直到 $z \ge 5$，如此便保证了 t 也大于或等于 5。类似地，对于模态 $start$，我们也加入了一个自反迁移，以保证屏蔽门关闭后会等待 5 秒钟，然后列车离开站点。

在 PHAVer 中，我们考虑符号状态 $sys.\{start \sim \$ \sim \$ \sim \$ \& t < 5\}$ 是否可达，即是否存在时延 $t < 5$ 的情况。PHAVer 的验证结果表明该状态是不可达的。同时，我们也考虑了如下符号状态：

```
sys.{ start ∼ $ ∼ $ ∼ $ & t >= 5 }.
```

经 PHAVer 验证，上述状态是可达的，可达状态以符号状态形式显示如下：

```
controller ∼ traindoor ∼ screendoor ∼ train.{
    start ∼ closed ∼ shut ∼ stop & -1 <= dy2 <= 0
    & 0 <= y2 <= 2 & x == 0 & y1 == 0 & close2_flag == 1
    & close1_flag == 0 & t >= 5 & z + y2 >= 2,
    start ∼ closed ∼ open ∼ stop & x == 0 &
    y2 == 2 & y1 == 0 & close2_flag == 1 &
    close1_flag == 0 & t >= 5 & z>=0,
    start ∼ closed ∼ closed ∼ stop &
    x == 0 & y2 == 0 & y1 == 0 & close2_flag == 0 &
    close1_flag == 0 & t >= 5 & z>=0,
    start ∼ closed ∼ part ∼ stop & 0 < y2 <= 2
    & x == 0 & y1 == 0 & close2_flag == 1 &
    close1_flag == 0 & t >= 5 & z + y2 >= 2
}.
```

根据验证结果可知，当控制器处于模态 $start$ 时，列车门是关闭的，但屏蔽门可能处于所有可能的四种模态，而且时延条件 $t \geq 5$ 始终是满足的。

3.6　列车防碰撞分析

在这一节里，我们对屏蔽门系统模型进行扩展，在模型中加入 4 辆列车，并设置 4 个地铁站点，每个站点有两个站台，分别用 A 和 B 表示，如图3.17所示。地铁轨道线与站台 A（或 B）侧相关的，称为轨道 A（或 B），相应地，列车行驶方向也称为方向 A（或 B）。起初，列车从站点 1 出发，以方向 A 行驶，在途中若遇到站点则在站点停留一段时间以便乘客上下车，然后继续行驶直到第四个站点，即站点 4，当乘客上下车后，列车会切换行驶方向，并通过额外的轨道停靠在站点 4 的站台 B，这额外的轨道部分，我们在本案例中没有进行细化，而是用列车等待一段时间然后到达站点 4 的站台 B 来进行简单抽象。类似地，当列车行驶到站点 1 的站台 B 时，也会进行行驶方向的切换然后回到站点 1 的站台 A。

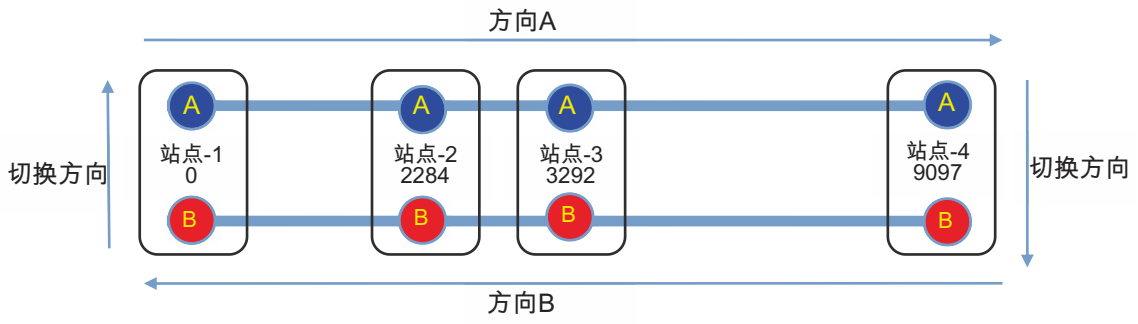


图 3.17: 地铁线路图

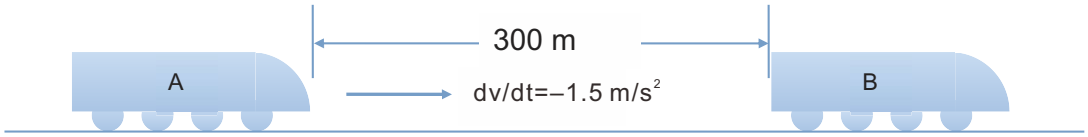


图 3.18: 列车间距控制

各站点的位置用其与站点 1 间的距离来表示，单位为米（m），如站点 1 的位置设为 0，站点 2 的位置为 2284，则站点 2 距离站点 1 为 2284 米，站点 3 和 4 的位置分别为 3292 和 9097。每个列车都有一组列车门，每个站台也有一组屏蔽门，因此，在这个模型中，我们有 8 组屏蔽门。屏蔽门和列车门的模型和之前建立的模型一致。

3.6.1　需求

在这个新系统中，我们将关注列车间安全距离的控制。我们的目标是，确保系统中没有列车碰撞情况发生。为了保持安全距离，在本案例中，我们对列车间的距离进行监控，当一个列车 A 与其前方的列车 B 之间的距离小于或等于 300 m（如图3.18所示）时，为免碰撞发生，列车 A 应立即制动，在该模型中，我们设置相应的减速度为 $-1.5\ \mathrm{m/s^2}$。

令 θ 为紧急距离，在我们的案例中，$\theta = 300$，假设对于列车 T_i 和 T_j，列车 T_j 处于 T_i 的正前方，则系统需求可以描述如下：

$$(Distance(T_i, T_j) \leq \theta \implies Urgent(T_i)) \wedge Distance(T_i, T_j) > 0$$

其中，函数 $Distance$ 返回列车 T_i 与 T_j 之间的距离，谓词 $Urgent(T_i)$ 表示列车

T_i 处于紧急制动状态。简单地说，当两列车间距小于或等于紧急距离时，处于后方的列车需要进入紧急制动状态，并且两列车的间距始终大于 0，即不会发生碰撞。

3.6.2 仿真

考虑到新系统的规模和复杂度，以及 Simulink/Stateflow 在表达能力和仿真结果快速反馈方面的优势，我们优先进行模型的仿真。Stateflow 支持 Matlab 函数声明和调用，而且我们还可以利用其丰富的控制结构，比如：条件和循环语句等。此外，在 Simulink/Stateflow 模型中，我们可以使用数组变量。因此，我们可以通过下标来对变量进行读写操作（表3.1列出了本模型中用到的部分变量）。这些特性对于建模是十分有利的。比如，在列车防碰撞模型中，我们有 4 个列车，每个列车的状态图结构和变量取名方式是一样的，因此，在控制中通过使用下标来对相应的变量进行存取操作既简单又高效。此外，在状态图中可以使用非线性表达式和内置的数学函数，比如：图3.19中模态 $urgent\_stop$ 的 exit 动作 $maxv = sqrt(0.5 * abs(S[j] - distance))$，其中 $sqrt$ 是平方根函数，abs 是取绝对值函数。

表3.1中，作用域为 $Output$ 的变量用于仿真结果的展现，可以作为示波器的输入。作用域为 $Input$ 的变量用于从 Stateflow 模型的外部获取值，如从 Simulink 组件接收信号（或者值）。类型为 $double$ 的变量是连续变量，同时也可以作为普通的浮点数变量。类型为 $int32$ 的变量为离散变量，存储整数值。类型为 $Inherit$ 的变量，表示其类型继承自 Stateflow 外部环境，即与相应的 Simulink 组件的类型一致。此外，局部变量和全局变量是一对相对概念。例如：若状态图 A 中声明有局部变量 x，而如果 A 有四个字图，则变量 x 在四个子图间可作为共享变量。但是，如果局部变量 y 只在某子图中声明，则其他子图将无法对 y 进行读写。

现在，我们介绍图3.19中用到的变量。变量 j 是一个索引变量，用于指示

表 3.1: 列车防碰撞的仿真模型使用的部分变量

变量名	作用域	类型	大小	初值
v_shutTD	Input	Inherit	[4 1]	
v_shutPSD	Input	Inherit	[4 2]	
openPSD	Local	double	[4 2]	
closePSD	Local	double	[4 2]	
Delay	Local	double	[1 4]	[0 50 100 150]
openTD	Local	double	[4 1]	
closeTD	Local	double	[4 1]	
S	Local	double	[1 4]	[0 2284 3292 9097]
N	Local	int32		4
urstopped	Local	double	[4 1]	
dir	Local	int32	[4 1]	
position	Local	double	[4 1]	
distance_out	Output	double	[4 1]	
y1_out	Output	double	[4 1]	
y2_out	Output	double	[4 2]	

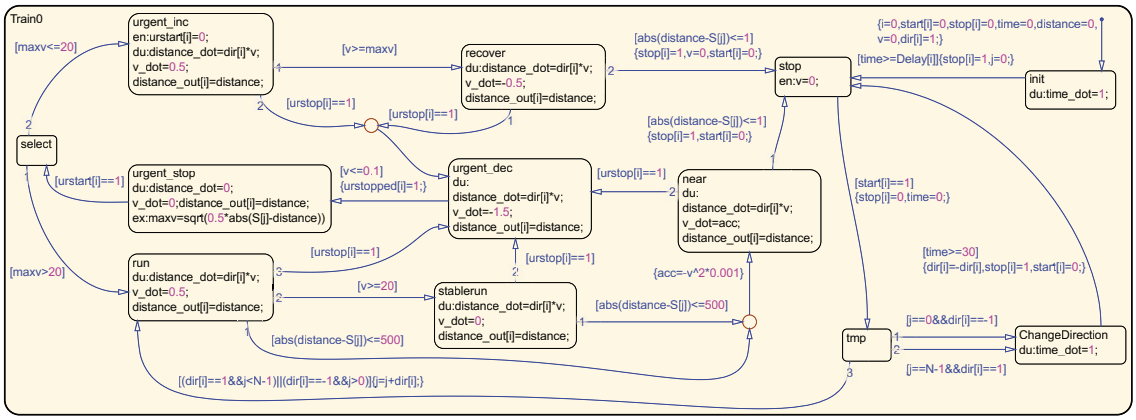


图 3.19: 列车防碰撞系统中的列车状态图

站点，j 的取值范围为 $\{0,1,2,3\}$。变量 N 代表站点的数目，在我们的案例中 $N=4$。变量 i 作为列车的索引，对于第一个列车，我们设置 i 的值为 0。对于其他列车，$i=1$，2 或者 3。列车 i 的行驶方向用数组元素 $dir[i]$ 表示。类似地，站点 j 的位置可以用数组元素 $S[j]$ 表示，例如，第一个站点的位置 $S[0]=0$。起初，列车在开始运行前会等待一段时间，$Delay[i]$ 表示列车 i 的等待时间，当列

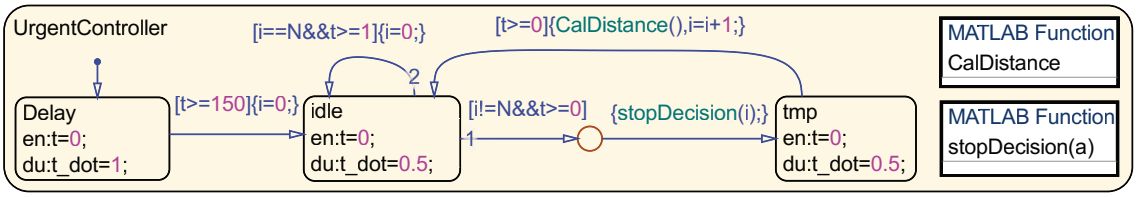


图 3.20: 紧急控制器。变量 i 是列车的索引。变量 N 表示列车的总数量。函数 $stopDecision$ 的参数 a 为列车的索引值

车的等待时间结束时，列车会停靠在第一个站点，从此开始运行。列车与第一个站点间的距离用变量 $distance$ 表示，列车的速度由变量 v 表示。当列车在行驶过程中收到制动命令，则其进入紧急制动，当列车需要重新启动时，我们根据列车与前方站点的距离和正常的加速度 0.5 $\mathrm{m/s^2}$ 计算列车可能的最大速度 $maxv$，若 $maxv > 20$ m/s，则列车会跳转到模态 run，否则跳转到模态 $urgent\_inc$。

图3.19中的小圆圈是连接接头（connective junction），用于当存在多种迁移路径可以选择时起到连接作用，其可以用于迁移动作的共享，实现选择分支或循环的功能，也可用于单路径情况。

图3.19的状态图是对先前模型的改进。所有列车不能同时从第一个站点的站台 A 开始运行，因为我们只有一条轨道。当列车从站点离开后，列车速度逐渐加快，当速度达到 20 m/s 时，列车将从模态 run 迁移到模态 $stablerun$ 中，在模态 $stablerun$ 中，其速度保持不变。如果列车在行驶过程中，其与前方站点的距离小于或等于 500 m 时，则列车需要减速（模态 $near$）。在运行过程中，每辆列车都需要与紧急控制器（图3.20）交互。

共享变量 $urstop[i]$ 用于指示是否需要对列车 i 进行紧急控制，变量 $urstart[i]$ 用于列车重启控制。我们将控制逻辑在 Matlab 函数 $stopDecision$ 中实现，判断是否将列车紧急制动。在 Matlab 函数 $CalDistance$ 中，对列车间的距离进行计算，判断被紧急制动的列车是否能够重启，若可以，则将相应的列车重启（通过设置变量 $urstart[i]$ 的值为 1）。

仿真结果如图3.21和3.22所示。图3.21展示的是 4 辆列车的一次常规运行情况。在第一个子图中，从左至右分别是列车 0、列车 1、列车 2 和列车 3 的位置

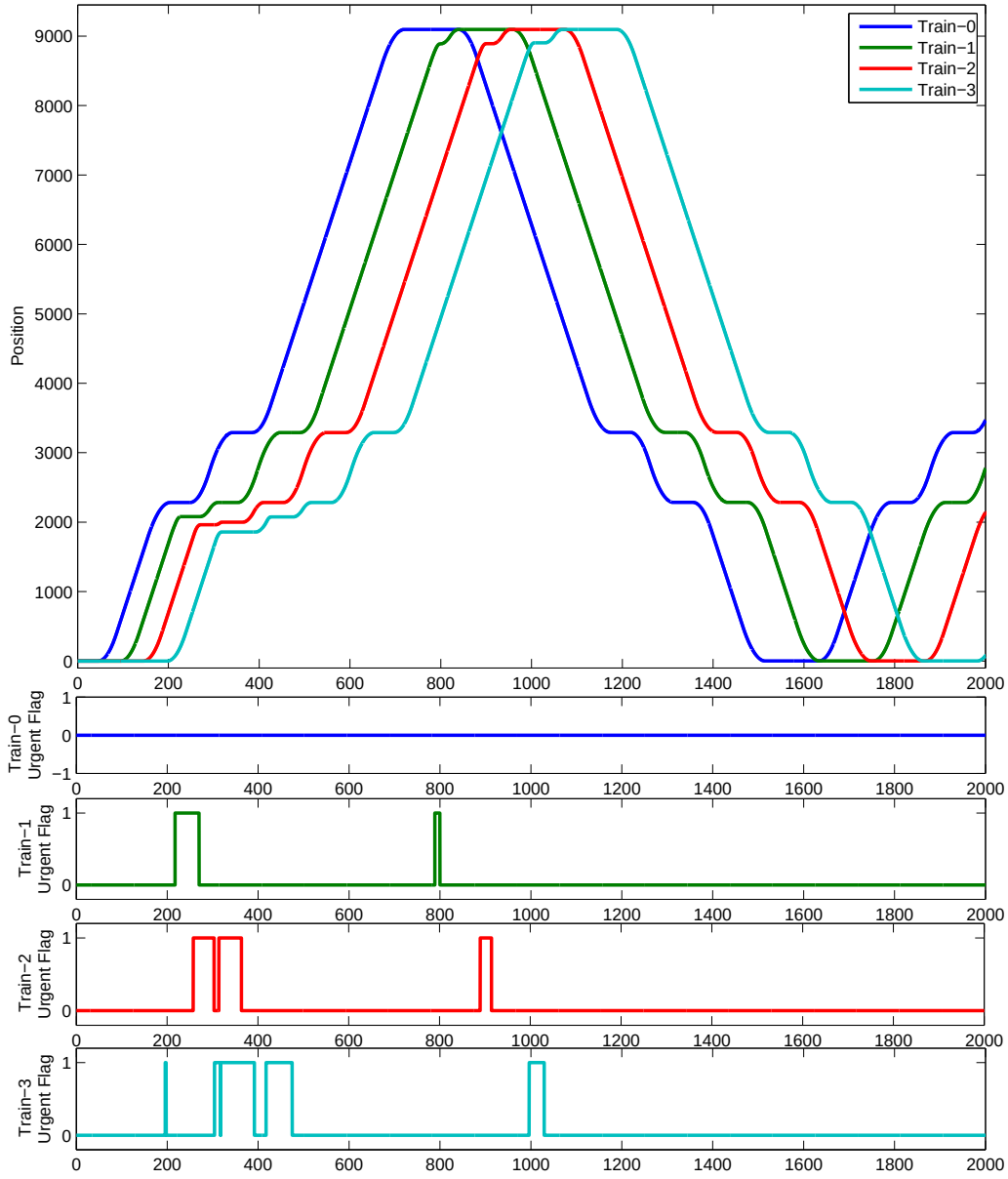


图 3.21: 列车防碰撞系统仿真结果。列车位置子图之下的四个子图显示的是列车紧急制动标志，若标志的值从 0 变为 1 则表示该列车加入紧急制动状态，若标记的值从 1 变为 0，则表示列车开始重启运行

随时间变化情况。其他四个子图表示列车紧急制动的情况，可以知道，第一个列车（即列车 0）没有进行过紧急制动，因为在它前面没有列车运行，而其他列车在到达最后一个站点（位置为 9097）之前均有紧急制动过程。

第二辆列车（即列车 1）在靠近站点 2（位置为 2284）时有一次紧急制动过程，因为第一辆列车已经停靠在了站点 2。这样就产生了连锁反应，导致列车 2

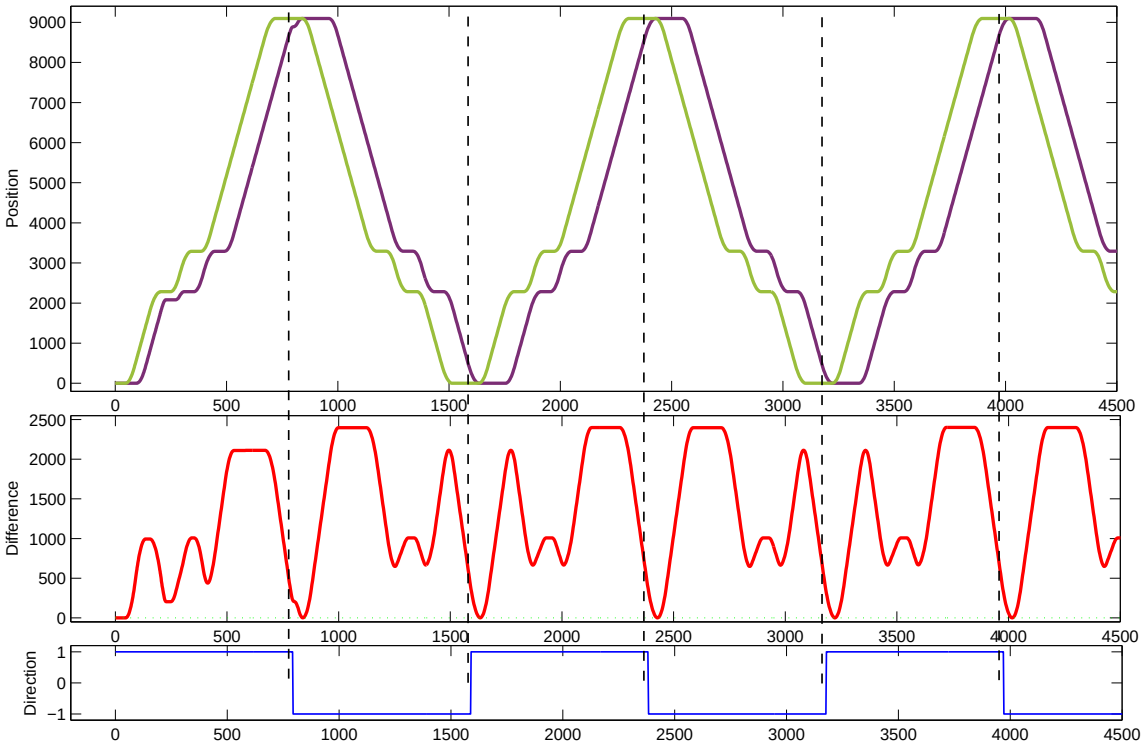


图 3.22: 两列车间的距离

和列车 3 也进行了相应的紧急制动操作。在最后一个站点（位置为 9097），除了列车 0 外，其他列车都有紧急制动情况，因为当他们靠近站点 4 时，已有列车停靠在站点，只有当前面列车切换方向且停靠在站台 B 后，后面的列车才能停靠在站台 A。

在列车切换方向后，在站台 B 侧的轨道线上行驶过程中，我们发现并没有发生紧急制动过程，这是由于在之前，通过距离控制，各列车间的距离已经是安全的了。

图3.22展示了列车 0 与列车 1 的位置差距。第一个子图表示的是两列车的位置，第二个子图是位置的差值，第三个子图是列车 0 的行驶方向，值 1 代表方向 A，值 -1 代表方向 B。虚线与时间轴相交的点表示列车 0 切换行驶方向的时刻点。

需要注意的是，当列车 0 切换方向后，两车的位置差值会逐渐变为 0，这是由于两车分别处于站台 A 和站台 B，所以会产生位置差值为 0 的情况，这是安全的，因为两个列车并不处于同一轨道。

3.6.3 验证

对于列车防碰撞系统，我们使用 SpaceEx[49] 进行建模和验证。SpaceEx 提供图形化建模界面，支持组件方式建模和仿射混成自动机的形式化验证。对于规模较大的混成系统，使用 SpaceEx 进行建模和分析是比较合适的。

由图3.21中的仿真结果可以知道，列车之间可以保持严格的运行顺序。通过仿真结果分析，可以简化混成自动机模型的构建和验证，而且，列车防碰撞模型虽然从实际应用的角度来看规模是很小的，但对于目前的混成系统验证工具来说还是无法顺利地形式化验证。

因此，我们在 SpaceEx 中只对两个列车进行建模和分析，这对于列车防碰撞分析已足够。列车混成自动机和紧急距离控制器自动机模型如图3.23和3.24所示，相关变量如表3.2所列。

通过 SpaceEx 的验证，我们找到一个在仿真中未发现的漏洞，验证结果如图3.25所示。其中，谓词 $loc(trainsys\_1.train)$ 表示第一个列车的模态，谓词 $loc(trainsys\_2.train)$ 表示第二个列车的模态，变量 $pos1$ 和 $pos2$ 分别表示两列车的位置，方向变量 $dir1$ 和 $dir2$ 分别表示两列车的行驶方向，值为 1 表示方向 A，值为 -1 表示方向 B。

从验证结果可知，两列车（分别处于组件 $trainsys\_1$ 和 $trainsys\_2$ 中）在同一个时刻停靠在第一个站点中。在屏蔽门系统的仿真和验证过程中，我们知道屏蔽门（或列车门）在关闭过程中可能会遇到异物或者因乘客过多而需要重新开启，这样在上下班高峰期就可能会出现屏蔽门（或列车门）需要耗费相当多的时间才能关闭的情况，这就使得列车在站点停留的时间较长。如此，在模型中，若列车在第一个站点之前的等待时间是固定的，而列车在站点的停留时间是非确定的，则我们将无法保障列车的有序运行。

模型的缺陷似乎很 ‘‘小’’，大部分开发者可能觉得他们可以很容易地在系统设计初期发现这样的错误。现实中，一个极其微小的软硬件错误往往导致巨大的灾难。在本案例中，我们对这个 ‘‘小’’ 缺陷进行了仔细分析，发现紧急距离控制

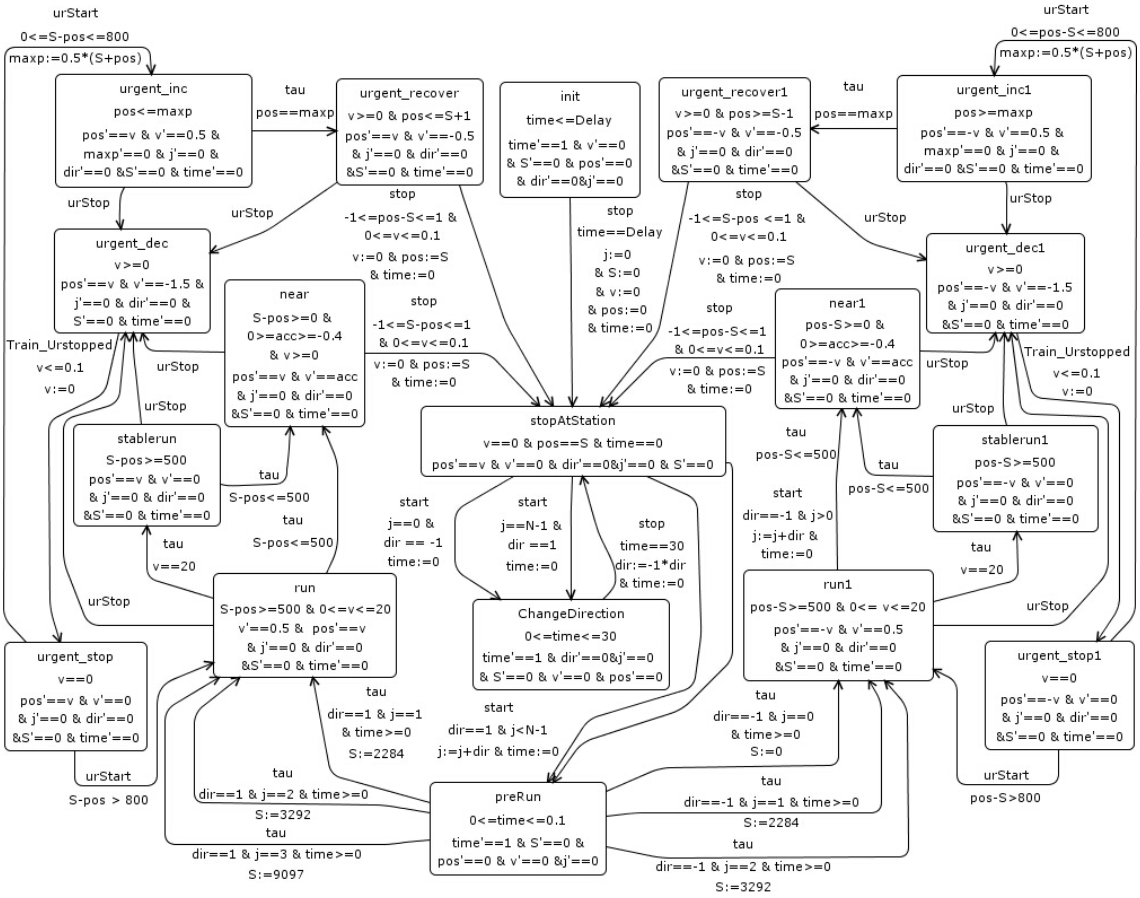


图 3.23: SpaceEx 中的列车自动机。SpaceEx 中的混成自动机在表达上与之前的形式化定义略有不同。例如，对于变量 x 的一阶导数，使用 x' 来表示。离散控制迁移中对于变量的更新情况，使用赋值语句来表示，如：$x := 0$。此外，符号 & 表示逻辑与操作符（$\wedge$）

器模型中还存在着另一个与此相似的设计失误。

在设计控制器时，我们认为行驶方向不同的两个列车停靠在同一个站点是安全的，但是，当两个列车停靠在最后一个站点时，假设列车 1 停靠在站台 A，列车 2 停靠在站台 B，那么，当列车 1 切换方向后就有可能会与停靠在站台 B 的列车 2 相碰撞，因为我们在模型中只设定了固定的等待时间，而对站点是否有列车停靠未做判断，因此，这也是一个潜在的设计缺陷。

这在我们的仿真分析中没有发现类似问题，而验证却可以检测到模型中的设计缺陷。验证可以克服仿真技术无法对系统所有场景进行分析的缺陷。这也是我们在混成系统建模和分析过程中使用协同验证的原因所在。

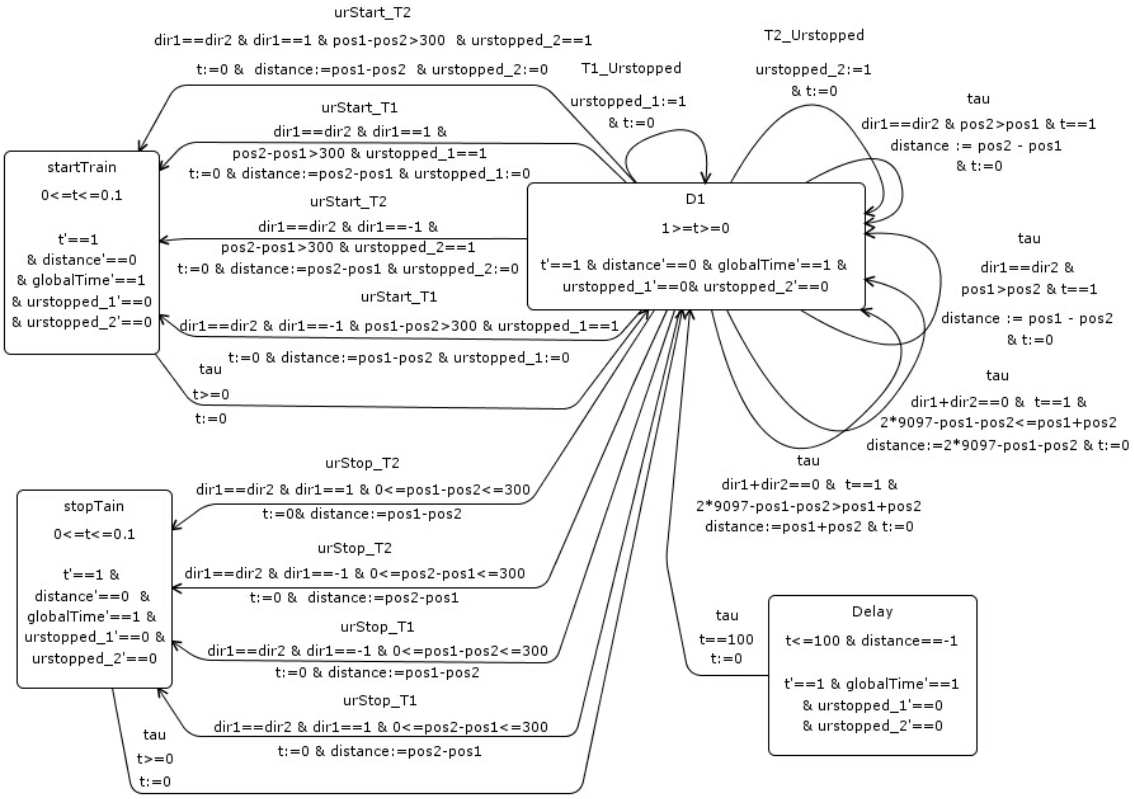


图 3.24: SpaceEx 中的紧急距离控制器自动机

表 3.2: SpaceEx 中列车和紧急距离控制器自动机涉及的部分变量

变量名	解释
$time$	在列车自动机中用作时钟变量，记录时间
$Delay$	列车的等待时间，等待时间到达后列车就停靠在第一个站点
v	列车的速度
pos	列车的位置
S	地铁站位置
dir	列车行驶方向，值为 1 表示方向 A，-1 表示方向 B（图3.17）
j	站点标识，列车前方到达站点或现已停靠站点的标识
t	时钟变量，用于紧急距离控制器自动机
$distance$	列车间的距离
$globalTime$	全局时间
$urstopped\_1$	紧急制动标识。若为 1 则第一辆列车需要紧急制动, 若取值为 0 则第一辆列车可以从紧急停车状态重启运行
$urstopped\_2$	紧急制动标识。用于第二辆列车

```
1  loc(trainsys_1.train)==stopAtStation & loc(trainsys_1.train)!=init &
2  loc(trainsys_1.traindoors)==closed &
3  loc(trainsys_1.controller)==closeScreenDoors &
4  loc(trainsys_2.train)==stopAtStation & loc(trainsys_2.train)!=init &
5  loc(trainsys_2.traindoors)==closed &
6  loc(trainsys_2.controller)==about_to_open2 &
7  ... ...
8  trainsys_1.traindoors.y == 0 & trainsys_1.train.S == 0 &
9  trainsys_1.train.time == 0 &  dir1 == 1 &
10 trainsys_1.train.v == 0 &  pos1 == 0 &
11 trainsys_1.stationid == 0 &
12 trainsys_2.traindoors.y == 0 & trainsys_2.train.S == 0 &
13 trainsys_2.train.time == 0 &  dir2 == 1 &
14 trainsys_2.train.v == 0 &
15 pos2 == 0 & trainsys_2.stationid == 0 &
16 trainsys_2.controller.z-trainsys_1.controller.z <= -1 &
17 trainsys_1.controller.z <= 2 & trainsys_2.controller.z >= 0
```

图 3.25: SpaceEx 验证结果

3.7　相关工作

屏蔽门系统是地铁控制系统的一个子系统。文献 [88] 报告了法国 ClearSy 公司 [35] 关于其屏蔽门控制器产品 COPPILOT [36] 的研究工作。该工作使用 B 方法 [1] 来开发屏蔽门系统。关于 B 方法的更多细节可以参考文献 [2] 以及 B 方法在工业界十五年的应用综述 [89]。B 方法和 Event-B 的相关工具和应用 [3, 132]，比如，在地铁中的应用可以在网站 [37] 中找到。

文献 [88] 中讨论的是一个实际工业界中的项目。COPPILOT 部署在法国巴黎地铁系统的若干地铁站台上：地铁 13 号线的荣军院（Invalides）站和圣拉扎尔（Saint-Lazare）站。与本章中模型的区别是，该工作在系统中采用红外线和雷达传感器来提升安全性。而我们研究的动机是，整合形式化验证和仿真技术并应用于实际系统。此外，在本章中，我们的模型是实际系统的抽象。而 [88] 的工作

还包括有从 B 方法模型到 LADDER 语言（[82] 中第八章有介绍）的代码转化。

关于屏蔽门系统的另外一个工作 [120] 是研究地铁站点的紧急疏散方案的，该工作以香港地铁有例，具体研究了在各种客流负载情况下的疏散时间。该工作的特色之处是使用经验方程和统计数据来对时间进行估计。另外，作者还考虑了列车停靠在站点时，列车门和屏蔽门的位置对疏散时间的影响。该问题在本章的案例中没有进行分析，我们将在未来工作中采用形式化方法来对该问题进行研究。

Metrô Rio 案例 [45] 采用 Matlab Simulink/Stateflow 开发列车自动防护系统（ATP：Automatic Train Protection）。对于由 Simulink 生成的代码，使用 Polyspace 检查代码的正确性。该工作与本章中讨论的研究的区别是，研究者在 Metrô Rio 案例的开发中没有考虑形式化验证与仿真的联系，而我们的工作可以让仿真和验证的结果相互反馈以促进系统的开发和正确性。

欧洲 ARTEMIS MBAT 项目 [99] 的一项研究 [96] 详细阐述了无人自动驾驶地铁系统的研究进展，提出了整合静态分析和动态测试技术的方法并被意大利 Ansaldo STS 公司[4]所采用。在对基于通信的列车控制案例分析中，他们研究了屏蔽门和列车门的开启问题，但关心的是哪个门应该先开。这与我们的案例所关注的不同，我们比较关心列车门和屏蔽门的关闭问题。与我们案例相同的是，该工作提出的分析过程同样整合了形式化验证和非形式化分析技术。在生命安全方面，文献 [123] 研究了地铁发生火灾时乘客逃生的情况。其中，火灾通过火灾动态模拟器（FDS：Fire Dynamics Simulator V406 [102]）进行仿真，以预测烟气扩展情况并分析安全逃生时间。研究指出，与未安装屏蔽门的地铁系统相比较，在地铁中安装屏蔽门可以更有利于保障乘客的生命安全。列车控制系统方面，文献 [141] 研究列车控制系统的需求分析。该工作首先将需求转化为 PSL[4] 模型，然后用需求分析工具 RATSY[21] 进行检验。文献 [80] 采用 Z 语言 [73] 和 Statechart[63, 64] 进行规范建模并验证。对比我们的研究，上述两项工作没有对列车的连续行为建模，而是将列车的连续动态行为抽象为离散的动作。

[4]http://www.ansaldo-sts.com/en/index

3.8　本章小结

本章，我们整合验证与仿真的优势，提出了协同验证分析过程，建立了屏蔽门系统和列车防碰撞控制系统的抽象模型，并运用形式化验证和仿真技术对模型进行分析和优化。

对于屏蔽门系统，我们在 Simulink 仿真过程中，基于交互式调试技术，发现模型中潜在的问题，即乘客可能会被夹于列车与屏蔽门之间，而处于危险境地。因此，我们对模型进行优化并重新仿真，然后在形式化模型中进行相应的模型修正，利用验证工具 PHAVer 的可达性分析技术对模型进行验证，保证了相关安全性质的满足以及模型的正确性。

对于列车防碰撞控制模型，我们通过仿真未能发现可能的列车碰撞场景，但仿真结果为模型的进一步抽象以适用于目前的形式化验证工具奠定了基础，从形式化验证工具 SpaceEx 的验证结果中可知，在模型中有两个可能的列车碰撞场景，如此，通过形式化验证我们可以检测到在仿真中无法分析出的模型缺陷。

因此，对于安全攸关系统，形式化验证和仿真是缺一不可的，在实际应用中，整合两者的优势，使用协同验证分析过程来设计和开发系统是完全有必要的。

第四章　基于混成关系的混成系统验证

在这一章中，我们提出一种基于混成关系理论的混成系统验证推理系统。混成系统的模型使用何积丰院士所提出的混成系统建模语言 HML [76] 来构建，在该语言中，控制器与环境的交互通过信号机制来实现。在本章中，我们使用的推理规则以霍尔三元组（Hoare Triples）的形式表示。最后，通过经典的水箱案例，以及列车防碰撞案例，我们展示了该验证方法的有效性。

4.1　引言

在文献 [75] 中，何积丰院士提出一个基于时钟的混成系统构建框架。其中，系统中事件的发生情况由时钟（clock）来记录。在 [75] 中，时钟是一个抽象的概念，与之相关的还有用于描述事件时序和时延关系的度量空间。系统中离散和连续变量的变化情况均可使用时钟来记录相应的演变时刻点。此外，何积丰院士还提出混成关系演算理论 [76]，混成关系中的字母表包含系统中的连续变量和输入/输出变量，因此，混成关系主要用于表示混成系统中这三种变量之间的关系。

在混成关系演算理论中，信号变量用于协调混成系统中各并发组件（或可称为子系统）间的行为。相应地，在混成系统建模语言 HML 中，信号机制用于同步各系统组件间的行为。在 HML 建立的混成系统模型中，信号是瞬时有效的，并且一旦产生即立刻向外广播。信号的存在与否（presence/absence）对于系统中所有组件来说都是一致的，即在同一时刻（或观察区间），任意两个组件对同一信号进行观察，只可能出现两种情况的其中一种。

在本章中，我们以 HML 为建模语言来对混成系统进行建模。在传统的建

模语言中（[65, 93, 100, 117]），混成系统的连续行为使用微分方程来表示，而在 HML 中，对于连续行为的约束，不仅有微分方程，还可以包含相应的连续变量的初值，以及连续行为终止条件等约束。信号作为系统组件之间的桥梁，主要用于表示组件之间的交互行为，如：通信、发送/接收控制命令等，而时钟用于存储信号的历史信息。如此，信号与时钟相辅相成地构建了混成系统中事件的产生，以及各组件之间进行通信的行为。

在形式化验证框架方面，何积丰院士和徐启文将时段演算（Duration Calculus[30, 32]）应用于串行混成程序，并提出相应的语义和证明框架 [78]。基于上述研究工作，我们扩展了霍尔逻辑推理规则以适用于 HML 以及混成系统的并发组合特性。对于 HML 模型，我们以霍尔三元组的形式给出相应的推理规则。这些推理规则在具体形式上受混成关系演算的启发。在混成关系演算中，信号的引入主要起着协调混成系统中各并发组件之间的交互行为的作用，连续变量用于表征物理环境的连续行为，谓词公式用于表示 HML 模型的行为。其中，未带撇标记的变量用于表示系统执行前的起始状态，相应的，带撇标记的变量用于表示系统执行连续或离散动作之后的新状态。同样地，时钟在混成关系演算中也是十分重要的。在对 HML 模型中并发程序进行验证时，各信号以及相关时钟之间的关系是构建相应的逻辑推理的基础。

本章中的案例分析包括经典的水箱案例以及列车防碰撞案例，其中列车防碰撞案例涉及两条地铁线路，并且这两条地铁线路共享部分轨道和站点。因而，在两条线路上行驶的列车需要通过控制器来协调各自的运行方式以防止两列车相撞的情况发生。本案例的模型是三个组件的并发组合混成程序，分别是两条地铁线路上行驶的两个列车以及相应的位置协调控制器。我们的需求是当两列车在共享区域行驶时必须保证不会发生列车相撞的情况。对于该安全性质的证明，首先，通过分析列车与控制器之间的信号，我们构造了信号之间的关系，然后，整个证明可分解为各组件对应的证明义务分别加以证明。信号之间关系的构建对于并发程序来说是极其重要的，这可以通过分析各组件中信号的时钟特性得以建立。

本章余下部分组织如下：在第4.2节中，我们主要描述建模语言 HML 的语法并解释相应的原子动作和程序语句。4.3节主要阐述混成关系演算的基本理论。在第4.4节，我们说明混成程序的断言设计，推理规则将在 4.5节中详细阐述，并在4.6中运用所提出的推理规则对水箱水位控制和防碰撞案例进行分析。

4.2　建模语言

在这一小节中，我们介绍混成系统建模语言 HML 的语法如下：

$$
\begin{aligned}
AP &::= \texttt{skip} \mid x := e \mid !s \mid \texttt{suspend}(e) \\
EQ &::= R(v,\dot{v}) \mid R(v,\dot{v})\ \texttt{init}\ v_0 \mid EQ \parallel EQ \\
P &::= AP \mid P \sqcap P \mid P;P \mid P \parallel P \mid P \triangleleft b \triangleright P \\
&\quad \mid \texttt{while}\ b\ \texttt{do}\ P\ \texttt{od} \mid EQ\ \texttt{until}\ g \mid \texttt{when}(G) \\
g &::= \epsilon \mid s \mid b \mid \texttt{out}(e) \mid g \odot g \mid g \oplus g \\
b &::= true \mid false \mid v \sim c \mid \neg b \mid b \wedge b \mid b \vee b \\
G &::= g \& P \mid G \,[\!]\, G
\end{aligned}
$$

其中，二元关系符 $\sim \in \{\geq, >, =, <, \leq\}$。原子程序 skip 表示一个空指令，该程序并不执行任何动作而且会立即终止。赋值语句 $x := e$ 将表达式 e 的取值赋给变量 x。原子程序 $!s$ 产生信号 s 并立即终止。信号产生后即被立即广播开来，而且信号是瞬时有效的。程序 $\texttt{suspend}(e)$ 等待 e 个单位时间然后终止，其中 e 的取值为大于 0 的数。关系谓词 $R(v,\dot{v})$ 为变量 v 及其时间导数 $\dot{v}$ 指定随时间变化的约束条件，用于表示连续的物理行为。例如，$\dot{v} = 1$ 和 $\dot{v} = v^2 + v - 5$ 是变量 v 及其导数 $\dot{v}$ 的两个以微分方程形式表示的关系谓词。此外，还有带初值条件的连续行为约束 $R(v,\dot{v})\ \texttt{init}\ v_0$，其中 v_0 是变量 v 的初值。对于多个方程联立的情况，可以用 $EQ_1 \parallel EQ_2$ 的形式来表示多个物理变量的连续演化。

符号 $\sqcap$ 用于表示非确定性选择，例如，$P_1 \sqcap P_2$ 表示从 P_1 和 P_2 中任意选择

一个程序执行。程序 $P;Q$ 表示程序 P 和 Q 的串行组合，即程序 P 首先执行，当 P 终止后再执行程序 Q。程序的并发组合用符号 $\|$ 表示，如并发组合 $P \| Q$ 表示程序 P 和 Q 的并发执行。条件选择语句 $P \triangleleft b \triangleright Q$ 表示当条件 b 为真时，程序 P 将会被执行，否则将执行程序 Q。循环程序 `while` b `do` P `od` 表示每当循环条件 b 成立时均执行程序 P，若在循环程序执行前或者在循环体程序 P 执行后条件 b 不成立则循环程序将终止。

带卫兵（guard）条件的微分方程 EQ `until` g 表示程序的执行遵循方程 EQ 的约束直到条件 g 满足为止。若在方程所对应的连续行为开始执行时条件 g 已经满足，则连续行为立即终止。卫兵条件可以是布尔条件表达式、信号或者二者的组合形式。例如，当卫兵条件 g_1 和 g_2 是布尔表达式时，卫兵条件 $g_1 \oplus g_2$ 等价于布尔表达式 $g_1 \vee g_2$，微分方程 EQ `until` $(g_1 \oplus g_2)$ 所对应的连续行为将会在表达式 $g_1 \vee g_2$ 为真时终止。当 g_1 和 g_2 为信号时，卫兵条件 $g_1 \odot g_2$ 将会在信号 g_1 和 g_2 同时出现时满足。空卫兵条件 ϵ 是立即满足的。信号卫兵条件 s 是在信号 s 出现时满足，卫兵条件 b 是布尔表达式。超时卫兵条件 `out`(e) 在相应的程序执行 e 个单位时间后满足。

等待选择程序 `when`$(g_1 \& P_1 [] \ldots [] g_n \& P_n)$ 的执行过程是：程序首先等待，若在等待过程中，有卫兵条件 g_i 成立则执行相应的子程序 P_i。若在等待过程中，没有卫兵条件满足，则继续等待；若同时存在多个卫兵条件成立则会在相应的子程序中任意选择其中一个执行。需要注意的是，在等待过程中，只要有卫兵条件成立则程序就无需继续等待，若在开始等待时就有卫兵条件满足，则立即停止等待并执行相应的子程序。

例 4.2.1. *弹跳球（Bouncing Ball）：假设有一人用手拿着一个具有弹性的球体，初始时处于地面上方一定高度处，变量 h 表示球的位置，球的初始速度 v 为 0 米/秒，当松开手后，球受重力作用垂直下落，当球到达地面时与地面碰撞然后反弹起来，碰撞会导致球的能量损耗，反弹后球的速度和方向会发生变化，具体表现为速度的绝对值会变小，方向与下落时相反，当球反弹到一定高度时球的速度为 0，如此该球将重新开始下落继续之前的下落和反弹过程。如下所示为弹跳*

球的一个 *HML* 模型。

```
1  while true do
2     ḣ = v ∥ v̇ = −g until h = 0;
3     (ḣ = v init 0) ∥ (v̇ = −g init  − c ∗ v) until v = 0
4  od
```

其中，c 表示 *0* 到 *1* 之间的一个常数，g 为重力加速度，方向向下，速度的方向取向上为正，向下为负。微分方程中的卫兵条件 $h = 0$ 表示当球体到达地面（位置为 *0*）时，球体下落的连续行为即可终止。

4.3　混成关系

首先，我们介绍时钟和信号的概念及相关的定义。简单来说，时钟是由非负实数组成的递增序列。在本章中，时钟记录了信号的发生时间。时钟 c 的所有元素组成的集合可用符号 $\mathbf{Set}(c)$ 来表示。时钟 c 的第 n 个元素表示为 $c[n]$，其中 n 为自然数。基于时钟的递增性，对于时钟 c，若有 $c[n]$，$c[n+1] \in \mathbf{Set}(c)$，我们有 $c[n] \leq c[n+1]$。

例 4.3.1. 时钟：假设信号 s 表示列车到站的事件，则相应的时钟 c_s 可能为 $c_s = \langle 0, 10, 21 \rangle$，表示在 *0* 时刻，以及 *10* 和 *21* 时刻有列车到站。时间的单位可以取秒或者分钟等等。

时钟 c 的基数（Cardinal Number）定义为该时钟的集合 $\mathbf{Set}(c)$ 的基数，表示时钟内元素的数量，即 $|c| =_{df} |\mathbf{Set}(c)|$。对于例4.3.1中的时钟 c_s，我们有 $|c_s| = 3$。

对于两个时钟 c 和 d，定义二元关系 $\preceq$ 表示其中一个时钟快于（faster）另一个时钟，即 $c \preceq d$ 当且仅当对于所有的 $i \in \mathbb{N}$，$c[i] \leq d[i]$。举例来说，时钟 $c_1 = \langle 0, 1, 1.4 \rangle$ 快于时钟 $c_2 = \langle 2, 4.3, 18 \rangle$。相应地，我们可以定义有界二元关系 $\preceq_t$，$c \preceq_t d$ 当且仅当 $c \preceq d$ 而且对于所有的 $i \in \mathbb{N}$，有 $c[i] \leq t$ 和 $d[i] \leq t$。此外，我们可以为时钟元素附加一个序号 n，则新的时钟元素是一个二元组 (t, n)，其

中时间 $t \in \mathbb{R}_{\geq 0}$，序号 $n \in \mathbb{N}$。在程序收到某个信号后，序号的值会增大，当发送信号时，新的序号值会记录在相应信号的时钟序列中，因此，序号 n 的值可以用于鉴别程序的输入信号和输出信号之间的依赖关系。为了下文中使用方便，我们在此定义作用在时钟元素上的两个映射，令 $\rho = (t, n)$ 为一个时钟元素，则映射 $\pi_1(\rho) = t$，映射 $\pi_2(\rho) = n$，即 π_1 是将时钟元素映射到其第一个分量上，π_2 则映射为第二个分量。

在通信系统中，信号可看作是一个函数，用于传递关于系统行为的信息，或者表征与系统所处环境的变化以及与某些现象的性质相关的信息 [119]。在 HML 语言中，每个信号都有两种状态，即出现（presence）或者未出现（absence），我们使用 $\xi(s)$ 和 $\theta(s)$ 分别表示信号 s 处于出现和未出现状态。

同时，每个信号 s 伴随有对应的时钟，使用变量 $s.clock$ 表示，该时钟用于记录信号 s 出现的时间点。相应的，我们使用带撇的变量 $s.clock'$ 表示程序开始执行后信号 s 的时钟的新值。

令时间区间 $[t, t']$ 为 HML 程序行为的观察区间。起始点 t 表示程序开始执行的时刻，终止点 t' 表示程序执行结束的时刻或者程序开始执行后而未终止执行的某一时刻点。在观察区间 $[t, t']$ 之上，信号 s 的出现和未出现可以定义为如下谓词表示：

- 信号 s 出现（presence），$\xi(s) =_{df} ((\pi_1^*(s.clock') \cap [t, t']) = \{t'\})$；
- 信号 s 未出现（absence），$\theta(s) =_{df} ((\pi_1^*(s.clock') \cap [t, t']) = \emptyset)$，

其中 π_1^* 是一个映射函数，用于将时钟映射为其所有时刻点组成的集合。相应地，对于连接符 $\oplus$ 和 $\odot$，我们有：

- $\xi(s_1 \oplus \ldots \oplus s_n) =_{df} \big((\bigcup_{i=1}^{n} \pi_1^*(s_i.clock')) \cap [t, t'] = \{t'\}\big)$；
- $\theta(s_1 \oplus \ldots \oplus s_n) =_{df} \big((\bigcup_{i=1}^{n} \pi_1^*(s_i.clock')) \cap [t, t'] = \emptyset\big)$；
- $\xi(s_1 \odot \ldots \odot s_n) =_{df} \big((\bigcap_{i=1}^{n} \pi_1^*(s_i.clock')) \cap [t, t'] = \{t'\}\big)$；

- $\theta(s_1 \odot \ldots \odot s_n) =_{df} \left((\bigcap_{i=1}^{n} \pi_1^*(s_i.clock')) \cap [t,t'] = \emptyset\right)$，

其中，$n \in \mathbb{N}$ 且 $n \geq 2$。从上述定义可以看出，判断一个信号处于出现状态需要该信号在观察区间的终止时刻点是出现的而在其他时刻点该信号是不出现的，而判断一个信号处于未出现状态需要该信号在观察区间的所有时刻点均未出现过。对于连接符 $\oplus$，只要存在有信号在终止点出现而且在其他时刻点没有出现则判断为相应的组合信号出现，对于连接符 $\odot$ 则需要所有信号同时只在终止点出现才能判断为组合信号出现，其他谓词定义可做类似解释。

对于布尔条件，我们也可定义类似的谓词用于判断布尔条件是否成立：

- $\xi(b) =_{df} b(t') \wedge \forall \tau \in [t,t') \bullet \neg b(\tau)$；
- $\theta(b) =_{df} \forall \tau \in [t,t'] \bullet \neg b(\tau)$；
- $\xi(b_1 \odot \ldots \odot b_n) =_{df} \xi(\bigwedge_{i=1}^{n} b_i)$；
- $\xi(b_1 \oplus \ldots \oplus b_n) =_{df} \xi(\bigvee_{i=1}^{n} b_i)$；
- $\theta(b_1 \odot \ldots \odot b_n) =_{df} \theta(\bigwedge_{i=1}^{n} b_i)$；
- $\theta(b_1 \oplus \ldots \oplus b_n) =_{df} \theta(\bigvee_{i=1}^{n} b_i)$，

其中，$b(t')$ 表示在 t' 时刻布尔条件 b 的取值，$n \in \mathbb{N}$ 且 $n \geq 2$。

此外，HML 程序可以使用动作 $!s$ 输出信号也可以在语句中表示接收或等待信号，如：EQ `until` s。在本章中，我们使用 $InSignal$ 代表 HML 程序的输入信号的集合，用 $OutSignal$ 代表输出信号的集合。

对 HML 程序行为的观察往往涉及模型中所有变量的取值变化情况，变量在 HML 程序执行之前、在程序终止时以及执行过程中的取值均是我们所要观察的。上述观察不仅关心程序的输入输出特征，还涉及程序在执行中的状态，因而可用于构建 HML 程序的一种逻辑关系，即混成关系（Hybrid Relation）。首先，让我们先来了解下，对于 HML 程序如何去表征程序的执行状态（status）。我们使用

变量 st 表示程序执行前的状态，相应地用 st' 表示程序执行后的状态。状态变量 st 和 st' 的取值范围为集合 $\{term, stable, div\}$。

- 若 $st = term$，则当前程序的前驱程序已终止，因而，当前程序可以开始执行。相应地，若 $st' = term$，则表示当前程序已终止。
- 若 $st = stable$，则表示前驱程序正处于等待信号或者等待条件退出，因而，当前程序无法开始执行。若 $st' = stable$，则表示当前程序处于等待状态。
- 若 $st = div$，则前驱程序处于一直发散状态，即前驱程序既非终止亦非等待，因而当前程序也无法开始执行。相应地，如果 $st' = div$ 则表示当前程序是处于发散状态的。

令 $in\alpha P$ 为 HML 程序 P 的输入变量集合，$out\alpha P$ 表示输出变量的集合，则变量 $st \in in\alpha P$，$st' \in out\alpha P$。

现在，我们给出混成关系的形式化定义如下：

定义 4.3.1. 一个混成关系是一个二元关系 P，其字母表 αP 定义如下：

$$\alpha P =_{df} in\alpha P \cup con\alpha P \cup out\alpha P$$

其中，$in\alpha P$ 为输入变量的集合用于指示程序执行之前的初值，$out\alpha P$ 表示输出变量的集合，可用于指示程序执行开始执行后的新值，$con\alpha P$ 用于表示连续变量的取值，是连续变量的集合。

例 4.3.2 (微分方程的混成关系表达). 考虑如下简单的微分方程:

$$EQ =_{df} (\dot{v} = 1)$$

其对应的混成关系可以使用谓词形式 P 表示如下：

$$P =_{df} (st = term) \wedge (t \leq t') \wedge (st' = stable) \wedge \forall \tau \in [t, t') \bullet \dot{v}(\tau) = 1$$

相应的字母表为：

$$\text{输入变量：} \quad in\alpha = \{st, t\},$$

$$\text{输出变量：} \quad out\alpha = \{st', t'\},$$

$$\text{连续变量：} \quad con\alpha = \{v\},$$

其中，输入变量 t 代表程序开始执行时的时刻点，变量 t' 表示程序执行后的时刻点。变量 v 是连续变量，可以连续变化。

谓词 P 表明变量 v 的一阶导数在区间 $[t, t')$ 上始终为 *1*。需要注意的是，我们的区间是右开区间，因为在 t' 时刻变量 v 的左右导数未必相等（比如变量 v 的值被重置则会导致间断点的存在）。

在 HML 语言中，输入和输出变量之间存在如下关系：

$$out\alpha P = in\alpha' P$$

其中，$in\alpha' P =_{df} \{v' \mid v \in in\alpha P\}$，即在形式上，输出变量是带撇标记的，而输入变量没有带撇标记。一般而言，一个混成关系 P 的输入变量定义如下：

$$in\alpha P =_{df} \{st, t, count\} \cup PVar \cup ClockVar$$

其中，$PVar$ 是离散变量的集合，$ClockVar =_{df} \{s.clock \mid s \in InSignal \uplus OutSignal\}$ 是用于表征与信号相关的所有时钟变量集合，$count$ 表示信号的序号。为了下文表述方便，我们在此定义混成关系的串行组合如下：

定义 4.3.2. 令 P 和 Q 为两个混成关系，若 $\alpha P = \alpha Q$，则混成关系 P 与 Q 的串行组合定义如下：

$$(P; Q) =_{df} \exists o \bullet P[o/v'] \wedge Q[o/v]$$

其中，v 代表混成关系 Q 的输入变量。$P[o/v']$ 为将 P 中出现的所有 v' 变量替换为 o 所得的混成关系。

4.4　HML 程序断言设计

基于 HML 程序的混成关系，对于 HML 程序的行为，我们引入设计（Design[71]）的概念。对于 HML 程序 P，其设计将涉及以谓词形式表示的，在 P 开始执行时所处的运行环境或者系统状态的假定，以及程序 P 执行后对环境的所产生的影响。对于 HML 程序，因其往往代表的是现实中的物理对象模型，我们将设定某些隐含的合理假设，比如，在程序执行过程中时间是一直朝前走

的，这可以用谓词 $t \leq t'$ 来表示。还有就是，每个信号所对应的时钟所包含的元素也会随时间增多的而不会减少的。因此，使用混成关系来表征 HML 程序的行为时，HML 程序需要满足如下两个健康条件 $\mathcal{HC}_1$ 和 $\mathcal{HC}_2$：

- $\mathcal{HC}_1$ ： $P = P \wedge t \leq t'$；
- $\mathcal{HC}_2$ ： $P = P \wedge \bigwedge_{s \in S} inv(s)$，

其中，$S =_{df} Insignal \uplus OutSignal$ 是信号的集合，$inv(s) =_{df} (s.clock \subseteq s.clock') \wedge (\pi_1^*(s.clock') \subseteq (\pi_1^*(s.clock) \cup [t, t']))$。相应地，我们有两个映射函数用于将混成关系映射为健康的混成关系：

- $\mathcal{H}_1$ ： $\mathcal{H}_1(P) =_{df} P \wedge t \leq t'$；
- $\mathcal{H}_2$ ： $\mathcal{H}_2(P) =_{df} P \wedge \bigwedge_{s \in S} inv(s)$，

现令 $\mathcal{H}_{12}$ 为 $\mathcal{H}_1$ 和 $\mathcal{H}_2$ 的组合函数，即：$\mathcal{H}_{12} =_{df} \mathcal{H}_1 \circ \mathcal{H}_2$，则有，

$$\mathcal{H}_{12}(\bot) =_{df} (t \leq t') \wedge \bigwedge_{s \in S} inv(s)$$

其中，$\bot$ 表示最坏的混成关系，其行为不可预测，可用于发散程序行为的表示。如此，我们现在可以对 HML 程序的行为使用上述关系进行定义。例如，对于空程序 `skip` 我们可以定义其混成关系如下（为了表述方便，我们使用相同的名称 `skip` 来表示）：

$$\texttt{skip} =_{df} (II_A \lhd st \neq div \rhd \mathcal{H}_{12}(\bot))$$

其中，$II_A =_{df} \forall x \in A \bullet (x' = x)$ 表示输入变量集合 A 中的变量没有发生变化。若前驱程序为发散的，则我们只知道时间和时钟的基本性质，而对程序的其他方面是不了解的；当前驱程序不是处于发散状态时，空程序不会改变输入变量的取值（当前驱程序处于终止状态时，空程序不做任何动作所以不会改变变量的值；若前驱程序为等待状态，空程序不会被执行，所以也不会影响变量的取值的）。若无特殊说明，在本章中，我们将默认符号 A 用于表示程序的输入变量的集合。

基于以上说明和约定，我们定义混成关系的健康映射 $\mathcal{H}$ 如下：

定义 4.4.1. 令 P 为一个混成关系，则其健康映射为：

$$\mathcal{H}(P) =_{df} \mathcal{H}_{12}(P; \texttt{skip}) \triangleleft st = term \triangleright \texttt{skip}$$

其中，skip 为前文中所定义的关系。

从混成关系的健康映射定义可以看出，若混成关系 P 所对应的程序的前驱程序处于终止状态，则混成关系 P 将映射为其自身与 skip 的串行组合并满足健康条件 $\mathcal{HC}_1$ 和 $\mathcal{HC}_2$。当前驱不是处于终止状态时，P 将映射为 skip。下面，我们给出设计的定义。

定义 4.4.2. 令混成关系谓词 β、ς 和 τ 分别表示 *HML* 程序执行前、执行中以及执行结束时所满足的约束条件。则 *HML* 程序的一个设计 $[\beta \vdash_{\mathcal{H}} \langle \varsigma \bowtie \tau \rangle]$ 可定义为：

$$[\beta \vdash_{\mathcal{H}} \langle \varsigma \bowtie \tau \rangle] =_{df} \mathcal{H}[(st = term \wedge \beta) \Rightarrow \langle \varsigma \bowtie \tau \rangle]$$

其中 β、ς 和 τ 均不含有变量 st 和 st'，$\langle \varsigma \bowtie \tau \rangle =_{df} (\varsigma \wedge st' = stable) \vee (\tau \wedge st' = term)$。

对于一个 HML 程序的设计，其直观含义是，若前驱程序终止则在满足条件 β 的前提之下，程序在执行过程中将满足条件 ς，程序终止时则满足条件约束 τ。

令 α 为一混成关系，且不含变量 st 和 st'。对于设计 $[\beta \vdash_{\mathcal{H}} \langle \varsigma \bowtie \tau \rangle]$ 我们在其中引入混成关系 α 作为不变式关系，得到设计的一种新形式：

$$[\alpha : (\beta \vdash_{\mathcal{H}} \langle \varsigma \bowtie \tau \rangle)] =_{df} [\beta \vdash_{\mathcal{H}} \langle (\alpha \wedge \varsigma) \bowtie (\alpha \wedge \tau) \rangle]$$

由上述定义可知，不变式关系 α 在程序执行中和结束时均是满足的。在本章中，对于 HML 程序，我们使用断言公式 $[\alpha : \{\beta\}\ [P]\ \{\langle \varsigma \bowtie \tau \rangle\}]$ 来表示程序 P 实现了设计 $[\alpha : (\beta \vdash_{\mathcal{H}} \langle \varsigma \bowtie \tau \rangle)]$，即程序 P 的行为满足该设计所对应的混成关系约束。为了方便阐述，在本章中，我们将使用符号 $\mathcal{D}(P)$ 表示 HML 程序 P 的一个设计。

例 4.4.1 (HML 程序的设计). 考虑一个只包含赋值语句 $x := 1$ 的 *HML* 程序，

我们有如下断言公式：

$$[x' \geq 0 : \{x = 0\}\ [x := 1]\ \{\langle false \bowtie x' = 1\rangle\}]$$

表示程序 $x := 1$ 实现如下设计：

$$[x' \geq 0 : (x = 0 \vdash_{\mathcal{H}} \langle false \bowtie x' = 1\rangle)]$$

但是该程序却不能实现设计：

$$[x' = 0 : (x = 0 \vdash_{\mathcal{H}} \langle false \bowtie x' \geq 0\rangle)]$$

因为，当程序终止时，变量 x 的新值是不等于 *0* 的，这与不变式关系谓词 $x' = 0$ 相矛盾。

定理 4.4.1 (设计封闭性). 对于设计，我们有如下规则：

1. $[\beta_1 \vdash_{\mathcal{H}} \langle \varsigma_1 \bowtie \tau_1\rangle]; [\beta_2 \vdash_{\mathcal{H}} \langle \varsigma_2 \bowtie \tau_2\rangle]$
   $= [\neg(\neg\beta_1; \mathcal{H}_{12}(\bot)) \wedge \neg(\tau_1; \neg\beta_2) \vdash_{\mathcal{H}} \langle(\varsigma_1 \vee (\tau_1; \varsigma_2)) \bowtie (\tau_1; \tau_2)\rangle]$

2. $[\beta_1 \vdash_{\mathcal{H}} \langle \varsigma_1 \bowtie \tau_1\rangle] \lhd b \rhd [\beta_2 \vdash_{\mathcal{H}} \langle \varsigma_2 \bowtie \tau_2\rangle]$
   $= [(\beta_1 \lhd b \rhd \beta_2) \vdash_{\mathcal{H}} \langle(\varsigma_1 \lhd b \rhd \varsigma_2) \bowtie (\tau_1 \lhd b \rhd \tau_2)\rangle]$

3. $[\beta_1 \vdash_{\mathcal{H}} \langle \varsigma_1 \bowtie \tau_1\rangle] \vee [\beta_2 \vdash_{\mathcal{H}} \langle \varsigma_2 \bowtie \tau_2\rangle]$
   $= [(\beta_1 \wedge \beta_2) \vdash_{\mathcal{H}} \langle(\varsigma_1 \vee \varsigma_2) \bowtie (\tau_1 \vee \tau_2)\rangle]$

4. $[\beta_1 \vdash_{\mathcal{H}} \langle \varsigma_1 \bowtie \tau_1\rangle] \wedge [\beta_2 \vdash_{\mathcal{H}} \langle \varsigma_2 \bowtie \tau_2\rangle]$
   $= [(\beta_1 \vee \beta_2) \vdash_{\mathcal{H}} \langle(\beta_1 \Rightarrow \varsigma_1 \wedge \beta_2 \Rightarrow \varsigma_2) \bowtie (\beta_1 \Rightarrow \tau_1 \wedge \beta_2 \Rightarrow \tau_2)\rangle]$

上述定理表明设计对串行组合、条件选择、析取和合取操作是封闭的。对于 HML 程序 $P_1; P_2$，如果子程序 P_1 实现了设计 $[\beta_1 \vdash_{\mathcal{H}} \langle \varsigma_1 \bowtie \tau_1\rangle]$，$P_2$ 实现了设计 $[\beta_2 \vdash_{\mathcal{H}} \langle \varsigma_2 \bowtie \tau_2\rangle]$，则串行程序 $P_1; P_2$ 可以实现设计 $[\neg(\neg\beta_1; \mathcal{H}_{12}(\bot)) \wedge \neg(\tau_1; \neg\beta_2) \vdash_{\mathcal{H}} \langle(\varsigma_1 \vee (\tau_1; \varsigma_2)) \bowtie (\tau_1; \tau_2)\rangle]$。

串行程序 $P_1; P_2$ 的发散行为可以分两种情况来说明。第一种情况，对于子程序 P_1，公式 $(\neg\beta_1; \mathcal{H}_{12}(\bot))$ 表示 P_1 的发散行为满足 $\neg\beta_1$ 并有健康条件 $\mathcal{H}_{12}(\bot)$ 跟随其后。该健康条件只表示时间的向前推进以及时钟的递增性。另一种情况是

关于子程序 P_2 的，公式 $(\tau_1;\neg\beta_2)$ 表示 P_1 是可终止的，而 P_2 的发散行为满足 $\neg\beta_2$。

串行程序 $P_1;P_2$ 在执行过程中的行为满足的条件可以表示为子程序 P_1 在执行过程中满足的条件 ς_1，或者当 P_1 终止后，可以表示为 P_1 的终止条件串上 P_2 的等待条件，即公式：$(\tau_1;\varsigma_2)$。串行程序的终止条件可以表示为子程序终止条件的串行组合 $(\tau_1;\tau_2)$。

对于条件选择程序 $(P_1 \lhd b \rhd P_2)$，如果条件 b 为真，子程序 P_1 可以实现设计 $[\beta_1 \vdash_{\mathcal{H}} \langle\varsigma_1 \bowtie \tau_1\rangle]$；条件 b 为假时，子程序 P_2 可以实现设计 $[\beta_2 \vdash_{\mathcal{H}} \langle\varsigma_2 \bowtie \tau_2\rangle]$，则程序 $(P_1 \lhd b \rhd P_2)$ 可以实现设计 $[(\beta_1 \lhd b \rhd \beta_2) \vdash_{\mathcal{H}} \langle(\varsigma_1 \lhd b \rhd \varsigma_2) \bowtie (\tau_1 \lhd b \rhd \tau_2)\rangle]$。

对于 HML 程序 $(P_1 \sqcap P_2)$，子程序的选择是非确定的。在程序执行时，选择哪个子程序是无法预知的，因此程序执行的前条件 β_1 和 β_2 均需满足。一旦选择后，程序表现出的行为只可能是其中的某个子程序的行为，因此我们使用析取操作符来连接相关的断言公式。

第 4 条规则表示 HML 程序 P 有两个可以实现的设计 $[\beta_1 \vdash_{\mathcal{H}} \langle\varsigma_1 \bowtie \tau_1\rangle]$ 和 $[\beta_2 \vdash_{\mathcal{H}} \langle\varsigma_2 \bowtie \tau_2\rangle]$ 的情况。需要注意的是，程序 P 在开始执行时，两个前条件 β_1 和 β_2 可能会存有冲突（不能同时满足），那么若假设 β_1 满足，则 P 的行为将可分别表示为 ς_1 和 τ_1；若假设 β_2 满足，则 P 的行为分别表示为 ς_2 和 τ_2。若前条件 β_1 和 β_2 是一致的，则 P 的行为可以表示为 $(\varsigma_1 \wedge \varsigma_2)$ 以及 $(\tau_1 \wedge \tau_2)$。

4.5　推理规则

在本小节中，我们将提出 HML 程序的霍尔逻辑（Hoare Logic）推理规则，涉及离散、连续以及组合等 HML 程序语句的推理。

离散赋值（SA）

将表达式 e 的取值赋给变量 x 的推理规则如下所示：

$$\frac{}{\mathtt{T} : \{\beta\}\ [x := e]\ \{\langle \mathtt{F} \bowtie II_A[e/x]\rangle\}}$$

其中，β 代表前条件，谓词 $true$ 和 $false$ 分别用符号 T 和 F 表示。集合 $A =_{df} in\alpha(x := e)$，表示程序所有的输入变量。在此，谓词 F 可以理解为离散动作的中间状态是不存在的（因为是瞬时动作）。断言 $II_A[e/x]$ 表示除了变量 x 外的所有变量没有被修改，而变量 x 的新值等于表达式 e 的取值，也即：$II_A[e/x] =_{df} II_{A\setminus\{x\}} \wedge (x' = e)$。

例 4.5.1. 假设赋值程序 $x := x + 1$ 的输入变量集合 $A = \{x\}$，则我们有如下推理规则实例：

$$\frac{}{\mathtt{T} : \{x = 5\}\ [x := x + 1]\ \{\langle \mathtt{F} \bowtie x' = x + 1\rangle\}}$$

由前条件 $\beta = (x = 5)$ 可知，该赋值程序执行结束时，其满足的断言 $x' = x + 1$ 等价于 $x' = 5 + 1$。

发送信号（SE）

发送信号也是一个离散动作，其推理规则如下所示：

$$\frac{}{\mathtt{T} : \{\beta\}\ [!s]\ \{\langle \mathtt{F} \bowtie II_A[(s.clock \cup \{(t, count)\})/s.clock]\rangle\}}$$

同样地，A 也表示该程序的输入变量集合，$A =_{df} in\alpha(!s)$。后条件 $II_A[(s.clock \cup \{(t, count)\})/s.clock]$ 表示信号 s 的时钟的新值将包含当前时间和序号组成的二元组，变量 $count$ 用于表示当前的序号。

执行等待（SD）

程序 suspend(e) 表示等待一定单位的时间 e，在等待过程中，程序不做其他任何动作。等待程序的推理规则如下：

$$\frac{}{\mathtt{T} : \{\beta\}\ [\mathtt{suspend}(e)]\ \{II_B \wedge \langle t' \in [t, t + e) \bowtie t' = t + e\rangle\}}$$

假设 A 为程序所有的输入变量的集合，则集合 $B =_{df} A\backslash(\{t\}\cup\{st\}\cup\{s.clock \mid s \in InSignal\})$ 表示除了时间变量 t、状态变量 st 以及输入信号的时钟之外的所有输入变量的集合。形如 $\phi \wedge \langle \phi_1 \bowtie \phi_2 \rangle$ 的公式是谓词 $\langle (\phi \wedge \phi_1) \bowtie (\phi \wedge \phi_2) \rangle$ 的简写形式。断言谓词 $t' \in [t, t+e)$ 表示在等待程序未结束时，时间变量的新值满足约束 $t \leq t' < t+e$。当等待结束时，时间变量的新值等于初始时刻的值加上程序所等待时间，即 $t+e$。

微分方程（EQ）

对于微分方程 EQ 及其终止条件 g，我们有如下的推理规则：

$$\frac{}{\begin{array}{l}\mathtt{T} : \{\beta\}\ [EQ\ \mathtt{until}\ g] \\ \{\psi \wedge \langle (\theta(g) \wedge count' = count) \bowtie (\xi(g) \wedge count' > \lambda) \rangle\}\end{array}}$$

其中，λ 表示程序终止时的序号值，取所有信号对应的时钟元素中的最大值，$\lambda =_{df} \mathtt{max}(\{0, count\} \cup \{\pi_2(last(s.clock')) \mid s \text{ 是 } g \text{ 中的信号}\})$，函数 $last$ 取时钟的最后一个元素，π_2 表示将时钟元素映射为其第二分量，即序号。

回顾前文中的定义，在此推理规则中，$\theta(g)$ 表示在程序执行过程中，终止条件 g（可能为信号或者布尔条件）不可满足，而当程序终止时，$\xi(g)$ 成立，表示终止条件 g 满足。断言公式 $\psi =_{df} (II_B \wedge \forall \tau \in [t, t') \bullet R(\tau))$，其中 $B =_{df} PVar \cup \{s.clock \mid s \in OutSignal\}$，$R$ 代表微分方程 EQ 的导数约束。例如，若 $EQ =_{df} (\dot{h} = 1)$，则其导数约束 $R(\tau) =_{df} (\dot{h}(\tau) = 1)$，表示在观察区间上，变量 h 的导数始终遵循微分方程中设定的值，即等于常数 1。

推论规则（CQ）

对于 HML 程序 S，有如下推论规则（Consequence Rule）：

$$\frac{\begin{array}{l}\beta \Rightarrow \eta, \quad \omega : \{\eta\}\ [S]\ \{\langle \mu \bowtie \nu \rangle\}, \\ \eta \wedge \omega \wedge \mu \Rightarrow \alpha \wedge \varsigma, \quad \eta \wedge \omega \wedge \nu \Rightarrow \alpha \wedge \tau\end{array}}{\alpha : \{\beta\}\ [S]\ \{\langle \varsigma \bowtie \tau \rangle\}}$$

其中，前条件 β 被强化，而后条件 $(\alpha \wedge \varsigma)$ 和 $(\alpha \wedge \tau)$ 被弱化。需要注意的是，我们有等价关系：$(\alpha : \{\beta\}\ [S]\ \{\langle \varsigma \bowtie \tau \rangle\}) \equiv (\{\beta\}\ [S]\ \{\langle (\alpha \wedge \varsigma) \bowtie (\alpha \wedge \tau) \rangle\})$。

串行组合（SC）

令 S_1 和 S_2 为两个 HML 程序，且 $out\alpha(S_1) = \{x' \mid x \in in\alpha(S_2)\}$，则对于二者的串行组合程序（$S_1; S_2$），我们有如下推理规则：

$$\frac{\begin{array}{l}\alpha_1 \vee (\alpha_1; \alpha_2) \Rightarrow \alpha,\\ \varsigma_1 \vee (\gamma_1; \varsigma_2) \Rightarrow \varsigma, \quad (\gamma_1; \tau_2) \Rightarrow \tau,\\ \alpha_1 : \{\beta\}\ [S_1]\ \{\langle \varsigma_1 \bowtie \gamma_1 \wedge \varepsilon \rangle\},\\ \alpha_2 : \{\varepsilon[*/*']\}\ [S_2]\ \{\langle \varsigma_2 \bowtie \tau_2 \rangle\}\end{array}}{\alpha : \{\beta\}\ [S_1; S_2]\ \{\langle \varsigma \bowtie \tau \rangle\}}$$

其中，子程序 S_1 终止时，谓词条件 $(\gamma_1 \wedge \varepsilon)$ 成立，ε 是一个不含输入变量的混成关系，其字母表为 $out\alpha(S_1) \cup con\alpha(S_1)$。相应地，$\varepsilon[*/*']$ 表示将混成关系 ε 中所有的输出变量替换为输入变量，因而 $\varepsilon[*/*']$ 是一个不含输出变量的混成关系。举例来说，如果 $\varepsilon =_{df} (x' \geq 1 \wedge y' = 0)$，则 $\varepsilon[*/*'] =_{df} (x \geq 1 \wedge y = 0)$。

循环执行（LP）

对于一个具有循环条件 b 以及循环体 S 的 HML 循环程序，我们有如下推理规则：

$$\frac{\begin{array}{c}(t' = t) \Rightarrow \alpha, \quad (\alpha; \alpha) \Rightarrow \alpha,\\ \alpha : \{b \wedge \beta\}\ [S]\ \{\langle \varsigma \bowtie \beta[*'/*] \rangle\}\end{array}}{\alpha : \{\beta\}\ [\texttt{while}\ \ b\ \ \texttt{do}\ \ S\ \texttt{od}]\ \{\langle \varsigma \bowtie (\neg b[*'/*] \wedge \beta[*'/*]) \rangle\}}$$

其中，$b[*'/*]$ 表示将 b 中所有的输入变量（不带撇标记的变量）替换为相应的输出变量（带撇标记的变量）。例如，如果 $b =_{df} (x > y)$，则 $b[*'/*] =_{df} (x' > y')$。前提条件 $(t' = t) \Rightarrow \alpha$ 强调在初始时刻，不变式约束 α 也是需要满足的。

等待选择（WP）

程序 $\texttt{when}(g_1\&P_1 \,[]\, \cdots \,[]\, g_n\&P_n)$ 首先等待卫兵条件 g_i 的成立，然后调度相应的子程序 P_i 继续执行。因此在断言的设计上需要考虑这两个阶段的性质，在下面的推理规则中，对于相应的混成关系谓词我们使用了串行组合操作：

$$\frac{\begin{array}{c}(\gamma;\omega)\vee\varsigma\Rightarrow\alpha,\\ \forall i\in I\bullet\omega:\{\nu[*/*']\}\ [P_i]\ \{\langle\zeta\bowtie\eta\rangle\}\end{array}}{\alpha:\{\beta\}\ [\texttt{when}(g_1\&P_1\,[]\,...\,[]\,g_n\&P_n)]\ \{\langle(\varsigma\vee(\gamma;\zeta))\bowtie((\gamma\wedge\nu);\eta)\rangle\}}$$

其中，$I=\{1,...,n\}$ 是所有子程序 P_i 的下标组成的有限集合。令 $A=PVar\cup\{s.clock\mid s\in OutSignal\}$，$\lambda=\texttt{max}(\{0,count\}\cup\{\pi_2(last(s.clock'))\mid s\in g_i,i\in I\})$，则 $\varsigma=_{df}\ I\!I_A\wedge count'=count\wedge\forall i\in I\bullet\theta(g_i)$，$\gamma=_{df}\ I\!I_A\wedge\exists i\in I\bullet\xi(g_i)\wedge count'>\lambda$。谓词 ς 用于表示等待行为，γ 表示有卫兵触发时所应满足的条件。

此外，需要注意的是，谓词 ν 是不含输入变量的混成关系。从前提条件可以看出，对于所有的子程序 P_i，都应满足相应的断言设计。

在引入并发推理规则之前，我们先介绍用于并发推理时对混成关系进行组合的合并机制 $\mathcal{M}$。合并机制是作用在混成关系中的变量上的一种运算：

$$\mathcal{M}=_{df}\ (x:\mathcal{T},\star)$$

其中，$\mathcal{T}$ 代表变量的类型（如实数 $\mathbb{R}$，自然数 $\mathbb{N}$ 等等），x 表示类型 $\mathcal{T}$ 的变量，$\star$ 是定义于类型 $\mathcal{T}$ 上的二元运算符。对于混成关系 P 和 Q，我们定义 $\mathcal{M}_{P,Q}=_{df}\ \exists m,n\bullet(P[m/x]\wedge Q[n/x]\wedge x=\star(m,n))$。

例 4.5.2. 令合并机制中的运算 $\star$ 为取最大值操作 max，类型 $\mathcal{T}$ 为实数集 $\mathbb{R}$。对于两个混成关系 P 和 Q，我们定义合并机制 $\mathcal{M}=_{df}\ (x':\mathbb{R},\ \texttt{max})$，其中变量 $x'\in out\alpha P\cap out\alpha Q$，则 P 和 Q 的合并可以表示为 $\mathcal{M}_{P,Q}=_{df}\ \exists m,n\bullet(P[m/x']\wedge Q[n/x']\wedge x'=\texttt{max}(m,n))$。

下面，我们开始引入并发组合程序的推理规则。

并发组合（PC）

HML 程序 P_1 和 P_2 的并发组合程序 $P_1\parallel P_2$ 的推理规则如下所示：

$$\frac{\begin{array}{c}\alpha_1 \wedge \alpha_2 \Rightarrow \alpha,\\ \alpha_1 : \{\beta_1\}\ [P_1]\ \{\langle \varsigma_1 \bowtie \tau_1\rangle\},\\ \alpha_2 : \{\beta_2\}\ [P_2]\ \{\langle \varsigma_2 \bowtie \tau_2\rangle\}\end{array}}{\alpha : \{\beta_1 \wedge \beta_2\}\ [P_1 \parallel P_2]\ \{\langle \mathcal{M}^s \bowtie \mathcal{M}^t\rangle\}}$$

其中，基本的合并机制 $\mathcal{M} =_{df} \{count' : \mathbb{N},\ \texttt{max}\}$。程序执行过程中满足混成关系 $\mathcal{M}^s =_{df} \mathcal{M}_{\varsigma_1,\varsigma_2} \vee \mathcal{M}_{\varsigma_1,(\tau_2;\delta_2)} \vee \mathcal{M}_{(\tau_1;\delta_1),\varsigma_2}$，在终止时满足混成关系 $\mathcal{M}^t =_{df} \mathcal{M}_{(\tau_1;\delta_1),\tau_2} \vee \mathcal{M}_{\tau_1,(\tau_2;\delta_2)}$。对于任意 $i = 1, 2$，$\delta_i =_{df} \mathit{II}_{A_i}$，其中 $A_i =_{df} PVar(P_i) \cup \{s.clock \mid s \in OutSignal(P_i)\}$。

需要注意的是，子程序 P_1 和 P_2 需满足如下的约束：

- $PVar(P_1) \cap PVar(P_2) = \emptyset$，
- $con\alpha(P_1) \cap con\alpha(P_2) = \emptyset$，
- $OutSignal(P_1) \cap OutSignal(P_2) = \emptyset$，

其中，第一条表示两个子程序不共享离散变量，第二条表示二者也不共享连续变量，第三条表示输出信号也是不共享的，即若 P_1 有输出信号 s，则其他与之并发执行的程序不应含有同样的信号输出（但等待或接收该信号是允许的）。

例 4.5.3. *考虑如下 HML 循环程序 P：*

$$P =_{df} (\texttt{while}\ true\ \texttt{do}\ S\ \texttt{od})$$

其中，$S =_{df} (\dot{h} = 1\ \texttt{until}\ h \geq 3;\ \dot{h} = -1\ \texttt{until}\ h \leq 1)$，$h$ 为连续变量。下面，我们将使用规则 **LP** *进行推理验证。令 $b_1 =_{df} (h \geq 3)$，$b_2 =_{df} (h \leq 1)$ 为两个特殊的信号（以布尔条件成立表示信号的出现），假设不变式 $\alpha = \mathtt{T}$，前条件 $\beta =_{df} t \geq 0 \wedge h(t) \in [1, 3] \wedge b_1.clock \preceq_t b_2.clock$，执行过程中满足条件 $\varsigma =_{df} (b_1.clock' \preceq_{t'} b_2.clock')$，因此，规则* **LP** *可以实例化如下：*

$$\frac{(t' = t) \Rightarrow \mathtt{T},\quad (\mathtt{T};\mathtt{T}) \Rightarrow \mathtt{T},\quad \mathtt{T} : \{\beta\}\ [S]\ \{\langle \varsigma \bowtie \beta[*'/*]\rangle\}}{\mathtt{T} : \{\beta\}\ [P]\ \{\langle \varsigma \bowtie \mathtt{F} \wedge \beta[*'/*]\rangle\}}$$

其中 $\beta[*'/*] =_{df} t' \geq 0 \wedge h(t') \in [1,3] \wedge b_1.clock' \preceq_{t'} b_2.clock'$。接下来，我们证明前提 $\mathtt{T} : \{\beta\}\ [S]\ \{\langle \varsigma \bowtie \beta[x'/x]\rangle\}$。相应地，应用规则 **SC** 可得：

$$\frac{\begin{array}{l}\varsigma \vee (\gamma_1; \varsigma) \Rightarrow \varsigma,\\ (\gamma_1; \tau_2) \Rightarrow \beta[*'/*],\quad \mathtt{T} \vee (\mathtt{T}; \mathtt{T}) \Rightarrow \mathtt{T},\\ \mathtt{T} : \{\beta\}\ [S_1]\ \{\langle \varsigma \bowtie \gamma_1 \wedge \varepsilon\rangle\},\\ \mathtt{T} : \{\varepsilon[*/*']\}\ [S_2]\ \{\langle \varsigma \bowtie \tau_2\rangle\}\end{array}}{\mathtt{T} : \{\beta\}\ [S_1; S_2]\ \{\langle \varsigma \bowtie \beta[*'/*]\rangle\}}$$

其中，S_1 和 S_2 为循环体 S 的两个子程序，即 $S_1 =_{df} (\dot{h} = 1\ \mathtt{until}\ h \geq 3)$，$S_2 =_{df} (\dot{h} = -1\ \mathtt{until}\ h \leq 1)$。$S_1$ 执行结束时满足 γ_1 表示卫兵条件 b_1 成立。其他断言公式定义为：$\tau_2 =_{df} \beta[*'/*]$，$\varepsilon =_{df} |b_1.clock'| > |b_2.clock'| \wedge b_1.clock' \preceq_{t'} b_2.clock'$。因此，子程序 S_2 的前条件 $\varepsilon[*/*'] =_{df} |b_1.clock| > |b_2.clock| \wedge b_1.clock \preceq_t b_2.clock$。断言 ε 的直观含义是，条件 b_1 的成立要先于 b_2 的成立。现在，使用规则 **CQ**，可以处理第一个子程序 S_1：

$$\frac{\begin{array}{l}\beta \Rightarrow \beta,\quad \beta \wedge \psi \wedge \theta(b_1) \wedge \mathcal{C}_s \Rightarrow \varsigma,\\ \beta \wedge \psi \wedge \xi(b_1) \wedge \mathcal{C}_t \Rightarrow \gamma_1 \wedge \varepsilon,\\ \mathtt{T} : \{\beta\}\ [S_1]\ \{\psi \wedge \langle(\theta(b_1) \wedge \mathcal{C}_s) \bowtie (\xi(b_1) \wedge \mathcal{C}_t)\rangle\}\end{array}}{\mathtt{T} : \{\beta\}\ [S_1]\ \{\langle \varsigma \bowtie \gamma_1 \wedge \varepsilon\rangle\}}$$

其中，$\lambda =_{df} \mathtt{max}(\{0, count\} \cup \{\pi_2(last(b_1.clock'))\})$，$\mathcal{C}_s =_{df} (count' = count)$，$\mathcal{C}_t =_{df} (count' > \lambda)$，$\psi =_{df} (II_B \wedge \forall \tau \in [t, t') \bullet \dot{h}(\tau) = 1)$。字母表 $B =_{df} PVar \cup \{s.clock \mid s \in OutSignal\}$。对于子程序 S_2，我们也可以做类似推理，在此不再列出其详细的推理过程。

4.6　案例分析

在本节中，我们使用前文中提出的推理规则来分析经典的水箱案例以及列车防碰撞案例。

表 4.1: 水箱模型中涉及的变量（常量）及其取值情况，水位高度的度量采用厘米（cm）为单位，时间以秒（s）为单位

名称	类别	初值	解释
H	离散	10	常量，最高水位 10 cm
L	离散	5	常量，最低水位 5 cm
h_i	离散	8	常量，初始水位 8 cm
h	连续	8	表征水位的变量，初值为 h_i
ε	离散	0.5	常量，用于控制水位的参数，其取值满足 $\varepsilon < \mathtt{min}(H - h_i, h_i - L)$
a	离散	1	常量，当控制阀打开时，水位上升，用于表示上升的速度（单位为 cm/s）
b	离散	1	常量，当控制阀关闭时，水位下降，用于表示下降的速度（单位为 cm/s）

4.6.1　水位控制

使用我们提出的推理规则，在这一小节中，我们对经典的水箱（water tank）水位控制案例进行分析。

水箱水位控制模型主要含有两个模态，一个是通过将水注入水箱而导致的水位上升，另一个是放水的过程，此时水位呈下降变化。表4.1中列车了水箱模型中使用到的变量（常量）及其初始取值。

水箱模型

首先，我们定义系统模型中两个与水位变化相关的子程序：

$$open =_{df} (\dot{h} = a)\ \mathtt{until}\ (h \geq H - \varepsilon),$$

$$closed =_{df} (\dot{h} = -b)\ \mathtt{until}\ (h \leq L + \varepsilon),$$

HML 程序 *open* 表示控制阀打开后，水位上升过程遵循微分方程 $\dot{h} = a$，并且其终止条件为 $h \geq H - \varepsilon$。当终止条件满足时，水位上升过程结束。程序 *closed* 表示控制阀关闭后，水位下降过程遵循微分方程 $\dot{h} = -b$，其终止条件为 $h \leq L + \varepsilon$。因而，水箱模型可以表示为如下 HML 程序，记为 $\mathcal{W}$：

```
1   while true do
2       !on; open;
3       !off; closed
4   od
```

其中，on 和 off 为两个信号，分别表示控制阀的开启和关闭。现在，考虑上述程序 $\mathcal{W}$ 的一个设计：

$$\mathcal{D}(\mathcal{W}) =_{df} [\alpha : (\beta \vdash_{\mathcal{H}} \langle \varsigma \bowtie \tau \rangle)]$$

其中，不变式 $\alpha = \mathtt{T}$。令 $c = (h \geq H - \varepsilon)$，则其他谓词可以表示为 $\beta = (h(t) \in [L, H] \wedge c.clock \preceq_t off.clock)$，$\varsigma = c.clock' \preceq_{t'} off.clock'$，$\tau = \beta[*'/*]$。谓词 β 的直观含义是初始时水位在 L 和 H 之间，而且条件 c 的成立要早于（$\preceq_t$）信号 off 的出现。程序 $\mathcal{W}$ 实现设计 $\mathcal{D}(\mathcal{W})$ 当且仅当我们可以证明如下断言成立：

$$\alpha : \{\beta\}\ [\mathcal{W}]\ \{\langle \varsigma \bowtie \tau \rangle\}$$

上述断言表示，程序 $\mathcal{W}$ 在开始执行以及执行结束时均满足水位在 L 和 H 之间的约束，并且条件 c 的成立要早于信号 off 的出现。在等待状态时，则满足相应的时钟关系，表明水位超过或等于 $H - \varepsilon$ 的时间点要早于（$\preceq_{t'}$）控制阀关闭的时间点。

水箱模型验证

图4.1展示的是水箱模型基本的推理过程。表中最左的 T 表示的是不变式 α 的取值，为了简单起见，对于 $\mathcal{W}$ 的子程序我们没有在图中加入不变式 T。图中，第二列的第一行表示的是谓词 β，最后一行表示的是程序在执行过程中满足 ς，终止时满足谓词 τ，但是需要注意的是，我们在 τ 中加入了 F，因为根据推理规则 **LP**，在循环终止时需满足 $\neg b$，即 $\neg true$，而 $\mathtt{F} \equiv false \equiv \neg true$，因此，我们需要在 τ 中加入 F。

图4.1中的推理包含有四个子程序的推理，分别对应程序 $!on$ 、$open$、$!off$ 以及程序 $closed$。在四个子程序的推理过程中，前一程序终止时的后条件经过变量替换后转化为后一程序的前条件。比如，程序 $!on$ 的后条件 $(h(t') \in [5, 10]) \wedge$

T	$\{h(t) \in [5,10] \wedge c.clock \preceq_t off.clock\}$
	[while *true* do
	$\{\mathtt{T} \wedge h(t) \in [5,10] \wedge c.clock \preceq_t off.clock\}$
	$[!on]$
	$\{\langle (c.clock' \preceq_{t'} off.clock') \bowtie (\mathbf{h(t') \in [5,10]}) \wedge (\mathbf{c.clock' \preceq_{t'} off.clock'})\rangle\}$
	$\{(h(t) \in [5,10]) \wedge (c.clock \preceq_t off.clock)\}$
	$[(\dot{h} = 1)$ until $(h \geq 9.5)]$
	$\{\langle (c.clock' \preceq_{t'} off.clock') \bowtie (\mathbf{h(t') \in [5,10]}) \wedge (\mathbf{c.clock' \preceq_{t'} off.clock'}) \wedge (\mathbf{\lvert c.clock'\rvert > \lvert off.clock'\rvert})\rangle\}$
	$\{h(t) \in [5,10] \wedge c.clock \preceq_t off.clock \wedge \lvert c.clock\rvert > \lvert off.clock\rvert\rangle\}$
	$[!off]$
	$\{\langle (c.clock' \preceq_{t'} off.clock') \bowtie (\mathbf{h(t') \in [5,10]}) \wedge (\mathbf{c.clock' \preceq_{t'} off.clock'})\rangle\}$
	$\{h(t) \in [5,10] \wedge c.clock \preceq_t off.clock\}$
	$[(\dot{h} = -1)$ until $(h \leq 5.5)]$
	$\{\langle c.clock' \preceq_{t'} off.clock' \bowtie h(t') \in [5,10] \wedge c.clock' \preceq_{t'} off.clock'\rangle\}$
	od]
	$\{\langle c.clock' \preceq_{t'} off.clock' \bowtie \mathtt{F} \wedge h(t') \in [5,10] \wedge c.clock' \preceq_{t'} off.clock'\rangle\}$

图 4.1: 程序 $\mathcal{W}$ 的基本推理过程。L 和 H 等常量符号已使用表4.1中的数值取代

$(c.clock' \preceq_{t'} off.clock')$ 经变量替换后变为 $(h(t) \in [5,10]) \wedge (c.clock \preceq_t off.clock)$ 成为程序 *open* 的前条件。下面，我们将对程序 *open* 的推理过程进行细化，其断言如下所示：

$$
\begin{aligned}
&\{(h(t) \in [5,10]) \wedge (c.clock \preceq_t off.clock)\}\\
&\quad [(\dot{h} = 1) \texttt{ until } (h \geq 9.5)]\\
&\{\langle (c.clock' \preceq_{t'} off.clock') \bowtie (h(t') \in [5,10]) \wedge\\
&(c.clock' \preceq_{t'} off.clock') \wedge (|c.clock'| > |off.clock'|)\rangle\}
\end{aligned}
$$

其中，$|c.clock'| > |off.clock'|$ 表示 c 的时钟含有的元素在数量上比 off 的时钟更多。基于规则 **CQ**，对于上述断言我们有如下的证明义务：

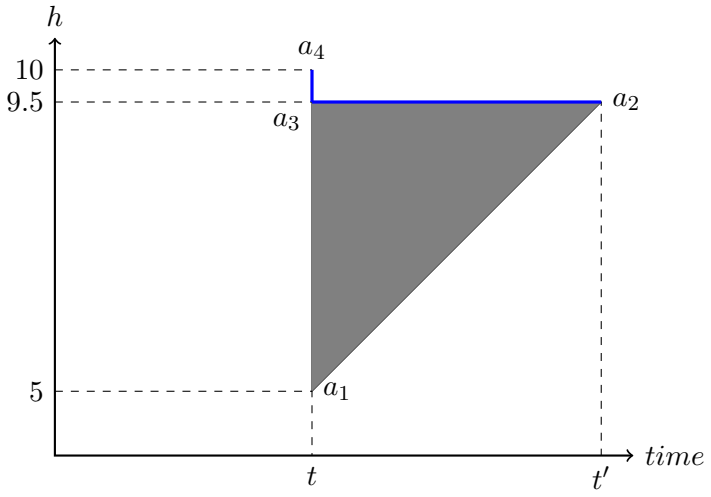


图 4.2: 程序 *open* 的状态空间示意图（初始水位在区间 $[5,10]$ 内）

1	$(h(t) \in [5,10]) \wedge (c.clock \preceq_t off.clock) \Rightarrow (h(t) \in [5,10]) \wedge (c.clock \preceq_t off.clock)$
2	$\{(h(t) \in [5,10]) \wedge (c.clock \preceq_t off.clock)\}$ $[(\dot{h} = 1)$ `until` $(h \geq 9.5)]$ $\{\psi \wedge \langle \theta(c) \wedge (count' = count) \bowtie \xi(c) \wedge (count' > \lambda) \rangle\}$
3	$(h(t) \in [5,10]) \wedge (c.clock \preceq_t off.clock) \wedge \psi \wedge \theta(c) \Rightarrow (c.clock' \preceq_{t'} off.clock')$
4	$(h(t) \in [5,10]) \wedge (c.clock \preceq_t off.clock) \wedge \psi \wedge \xi(c) \wedge (count' > \lambda)$ $\Rightarrow (h(t') \in [5,10]) \wedge (c.clock' \preceq_{t'} off.clock') \wedge (\lvert c.clock' \rvert > \lvert off.clock' \rvert)$

其中，$\psi =_{df} (II_B \wedge \forall v \in [t,t') \bullet \dot{h}(v) = 1)$，字母表 $B =_{df} PVar \cup \{s.clock \mid s \in OutSignal\}$。

第一个证明义务可以很容易直接得证，第二个证明义务可以使用规则 **EQ** 来证明。对于第三个证明义务，首先，我们知道的是，时钟 $c.clock'$ 与其初值 $c.clock$ 是相等的，这可以由 $\theta(c)$ 推出。此外，因为 $off.clock \in B$，所以可以从 II_B 推出 $off.clock' = off.clock$ 。从而，$(c.clock' \preceq_{t'} off.clock')$ 成立。在第四个证明义务中，$\xi(c)$ 表示 c 在 t' 时刻成立，则我们有 $|c.clock'| = |c.clock| + 1$ 成立，同时我们可以从 II_B 推出 $off.clock' = off.clock$，所以，$|c.clock'|$ 要比 $|off.clock'|$ 大。

图4.2表示的是程序 *open* 的状态空间，其中，横坐标代表时间，纵坐标代表

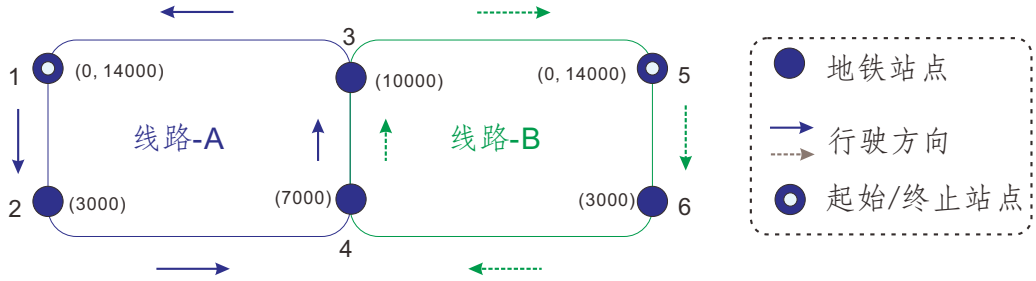


图 4.3: 地铁轨道线。左子图（包含站点 1、2、3 和 4）代表地铁轨道线路 A，列车以逆时针方向行驶；右子图（包含站点 3、4、5 和 6）为轨道线路 B，列车以顺时针方向行驶

水位。图中的灰色区域，加上线段 $[a_2, a_3]$ 和 $[a_3, a_4]$ 表示程序在时间区间 $[t, t']$ 上执行时水位可能的取值情况。线段 $[a_2, a_3]$ 和 $[a_3, a_4]$ 表示程序终止时所有可能的状态。

线段 $[a_2, a_3]$ 上的点表示水位为 9.5 时的状态，如果初始时水位 $h(t) \in [5, 9.5]$，则程序最终将会终止于线段 $[a_2, a_3]$ 中的状态。若初始时水位 $h(t) \in [9.5, 10]$，则程序将立即终止，此时所有可能的终止状态均落于线段 $[a_3, a_4]$ 内，而且 $t' = t$。对于上述两种情况，我们始终有 $h(t') \in [9.5, 10]$，而且 $h(t') \in [9.5, 10] \Rightarrow h(t') \in [5, 10]$，所以水位始终控制在 L 和 H 之间。

4.6.2　列车防碰撞控制

这一小节，我们对两条具有共享轨道（和站点）的地铁线路以及运行在线路上的列车进行建模并分析。

共享轨道地铁系统模型

如图4.3所示，我们的模型中共有 6 个地铁站点，垂直方向上的两个站点之间的距离为 3 km，水平方向上相邻站点间距离为 4 km。例如，站点 1 与站点 3 之间的距离为 4 km。图中，左半部分为地铁线路 A，右半部分为地铁线路 B，它们共享站点 3 和 4，以及站点 3 和 4 之间的地铁轨道线。这两条地铁线路的周长均为 14 km。在每个站点附近，我们都有位置标记，如：站点 1 的位置标记

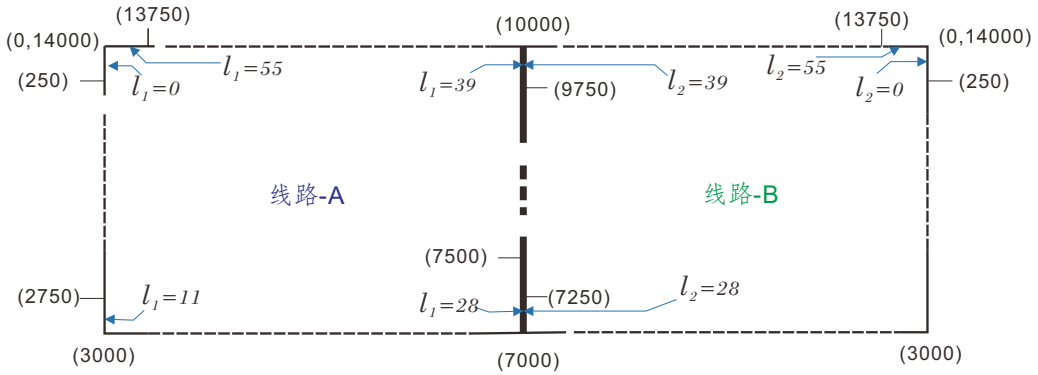


图 4.4: 地铁轨道线路区段划分。每 250 米为一轨道区段，每条线路有 56 个区段

为 $(0, 14000)$，因其既是起始又是终止站点，站点 2 的位置标记为 (3000)，因其距离站点 1 为 3000 m。在我们的案例中，每条地铁线路上有一辆列车 T_i（i =1 或 2，分别表示线路 A 和 B 的列车）运行。变量 v_i 表示列车 T_i 的行驶速度，p_i 表示列车的位置。模型中表示列车行为的 HML 程序如下所示：

```
1   Id_i := 0;
2   while true do
3         Stop; Id_i := Id_i + 1;
4         Cho;  frun_i := true;
5         (ṗ_i = 0 init 0 until true) ◁ Id_i = 2 ▷ skip;
6         while frun_i do Run od
7   od
```

其中，变量 Id_i 是一个位置标志，用于在子程序 Cho 中更新列车前方将要到达的站点的位置。变量 S_i 表示列车运行前方站点的位置。例如，当列车 T_1 停靠在站点 1 时，前方将要到达的站点是站点 2，那么通过判断 Id_1 的值，在程序 Cho 中 S_1 将会被设置为 3000，即站点 2 的位置。上述程序代码中涉及到的子程序 $Stop$ 和 Cho 定义如下：

$$Stop =_{df} l_i := \lfloor p_i/250 \rfloor; l_i := l_i \ \mathbf{mod}\ 56; !g_{i,l_i}; \mathtt{suspend}(50);$$

$$Cho =_{df} \mathtt{when}(Id_i = 2\&(S_i := 3000) \,[]\, Id_i = 3\&(S_i := 7000)$$

$$[]\, Id_i = 4\&(S_i := 10000) \,[]\, Id_i = 5\&S_i := 14000; Id_i := 1)$$

其中，$\lfloor p_i/250 \rfloor$ 表示对除法运算 $p_i/250$ 的结果向下取整，**mod** 表示取余运算，变量 l_i 表示轨道区段的序号。图4.4表示轨道的区段划分，每条轨道线划分为 56

段，每段长为 250 m。此外，信号 g_{i,l_i} 将会在列车 T_i 到达区段 l_i 时发出。

子程序 Run 的详细代码如下所示：

```
1  fcrun_i := true; l_i := ⌊p_i/250⌋;
2  !g_{i,l_i}; l_i := l_i + 1;
3  while fcrun_i do
4     (v̇_i = 0.5 || ṗ_i = v_i)
5     until (p_i = 250 * l_i) ⊕ (S_i = p_i + 500) ⊕ (v_i = 20) ⊕ urstop_i;
6     when((p_i = 250 * l_i)&(!g_{i,l_i}; l_i := (l_i + 1) mod 56) []
7           ((v_i = 20) ⊕ (S_i = p_i + 500) ⊕ urstop_i)&(fcrun_i := false))
8  od;
9  when((S_i = p_i + 500)&Near [] (v_i = 20)&Srun [] urstop_i&Urde)
```

其中，变量 $fcrun$ 是一个布尔类型的变量，用于循环条件的控制。程序 Run 主要用于表示列车离开站点后的动态行为。从 $\dot{v} = 0.5$ 可以知道，列车离开站点后的加速度为 0.5 m/s^2。当速度到达 20 m/s 的时候，系统的执行将切换为子程序 $Srun$，列车将以 20 m/s 的速度行驶，不再加速。当列车距离前方站点 500 m 时，将切换为子程序 $Near$，列车将减速并最终停靠在站点。如果在列车 T_i 行驶过程中收到信号 $urstop_i$，则该列车将进入紧急减速子程序 $Urde$。此外，我们每进入一个轨道区段 l_i 都将发出相应的信号 g_{i,l_i}。相关子程序的详细代码在附录A中列出，在本小节中不再做详细介绍。

为了防止列车在共享轨道上相撞，我们还需要一个控制器来协调两辆列车之间的行驶方式。该控制器主要根据接收到的信号 g_{i,l_i} 来判断两车的位置，进而在适当的时机，通过发送信号 $urstop_i$ 控制列车的紧急停车，或者通过信号 $urstart_i$ 让列车重新运行。控制器对应的 HML 程序如下所示：

```
1  f_1 := 0; f_2 := 0;
2  cf_1 := 0; cf_2 := 0;
3  while true do
4     when(g_{1,f_1}&(b := true; cf_1 := f_1) [] g_{1,f_2}&(b := false; cf_2 := f_2));
5     dcond := (26 ≤ cf_1 ∧ cf_1 ≤ 40 ∧ 26 ≤ cf_2 ∧ cf_2 ≤ 40);
6     Decide ◁ dcond ▷ (!urstart_1; !urstart_2);
7     (f_1 := (f_1 + 1 mod 56)) ◁ b ▷ (f_2 := (f_2 + 1 mod 56))
8  od
```

其中，紧急控制的判断主要发生在当列车处于轨道区段 26 至 40 的时候，因为在该区域上两列车将会共享轨道线路行驶。

变量 f_i 和 cf_i 分别表示列车 T_i 行驶前方将要到达的区段以及当前所在的区段的编号。子程序 $Decide$ 主要判断是否发送紧急停车或紧急重启的信号，其程序代码如下：

```
1   ds1 := cf1 ≤ cf2 ∧ cf1 ≥ cf2 − 2;
2   ds2 := cf2 ≤ cf1 ∧ cf2 ≥ cf1 − 2;
3   !urstop1 ◁ ds1 ▷ (!urstop2 ◁ ds2 ▷ (!urstart1; !urstart2))
```

可以看出，若 ds_1 为真，则列车 T_1 可能在列车 T_2 后方，并且两列车所处轨道区段编号最多相差为 2，此时需要将列车 T_1 紧急停车。类似地，变量 ds_2 的作用主要是用于 T_2 在 T_1 后方的情况。

若上述两种情况均未发生，则两列车均可正常行驶（需要注意的是在列车行驶过程中信号 $urstart_i$ 是不会对列车造成副作用的）。

现在，令 $\mathcal{C}$ 代表控制器程序，$\mathcal{T}_i$ 代表列车 T_i 所对应的 HML 程序，则本案例中的共享轨道地铁系统模型程序 $\mathcal{O}$ 可以表示如下并发程序：

$$\mathcal{O} =_{df} \mathcal{C} \,||\, \mathcal{T}_1 \,||\, \mathcal{T}_2$$

其中，控制器 $\mathcal{C}$ 主要在列车 T_1 和 T_2 在共享区段行驶时起到协调控制的作用。下面，我们将对该系统模型进行分析和验证。

共享轨道地铁模型的验证

基于列车防碰撞的基本需求，对于我们的模型，在系统运行过程中，下面的谓词公式是需要满足的：

$$(p_1 \in [7000, 10000] \wedge p_2 \in [7000, 10000]) \Rightarrow (p_1 \neq p_2)$$

该谓词公式的含义是指当两列车在共享区段运行时，将不会出现两列车位置相等的情况，即不会相撞。考虑如下设计：

$$\mathcal{D}(\mathcal{O}) =_{df} [\alpha : (\beta \vdash_{\mathcal{H}} \langle \varsigma \bowtie \tau \rangle)]$$

其中，$\alpha = \mathtt{T}$，$\tau = \mathtt{F}$，前条件$\beta = (\forall i \in \{1,2\} \bullet (p_i(t) = 0 \wedge f_i = 0 \wedge cf_i = 0 \wedge Id_i = 0 \wedge l_i = 0) \wedge t = 0 \wedge \forall i \in \{1,2\}, j \in \{0,...,55\} \bullet g_{i,j}.clock = \emptyset)$，表示系统开始运行的时间点为 0，所有的时钟均不含有元素，列车的初始位置均为 0。系统运行过程中满足 $\varsigma = (p_1(t') \notin [7000, 10000] \vee p_2(t') \notin [7000, 10000] \vee p_1(t') \neq p_2(t'))$，表示列车不在共享区段上或者两列车的位置不相等。

对于控制器程序 $\mathcal{C}$，我们指定其前条件为：

$$\begin{aligned}\beta_{\mathcal{C}} =_{df}\ & \psi \wedge (cf_1 < 27 \vee cf_1 > 40 \vee \\ & cf_1 < cf_2 - 1\ \vee cf_2 < 27 \vee \\ & cf_2 > 40\ \vee cf_2 < cf_1 - 1)\end{aligned}$$

其中，$\psi = \forall i \in \{1,2\}, j \in \{0,...,55\} \bullet (j \neq cf_i) \Rightarrow \pi_1(last(g_{i,cf_i}.clock)) > \pi_1(last(g_{i,j}.clock))$。公式 ψ 中的关系式 $\pi_1(last(g_{i,cf_i}.clock)) > \pi_1(last(g_{i,j}.clock))$ 表明信号 g_{i,cf_i} 是控制器从列车 T_i 处接收到的最新的信号。

对于列车 T_i，我们指定其前条件为：

$$\begin{aligned}\beta_{\mathcal{T}_i} =_{df}\ & p_i(t) \in [250 * (l_i - 1), 250 * (l_i + 1)) \\ & \wedge p_i(t) \in [0, 14000] \wedge I_{clock}\ \wedge \psi_{clock}\end{aligned}$$

其中，$i \in \{1,2\}$，各信号之间的关系为 $I_{clock} = (g_{i,0}.clock \preceq_t ... \preceq_t g_{i,55}.clock)$，$\psi_{clock} =_{df} \forall j \in \{0,...,55\} \bullet (l_i - 1 \geq 0 \wedge j \neq l_i - 1) \Rightarrow \pi_1(last(g_{i,l_i-1}.clock)) > \pi_1(last(g_{i,j}.clock))$。与公式 ψ 类似的是，ψ_{clock} 中的关系式 $\pi_1(last(g_{i,l_i-1}.clock)) > \pi_1(last(g_{i,j}.clock))$ 表明信号 g_{i,l_i-1} 是最新发出的信号。

最新信号在断言中的作用在于其可用于表示列车的最新位置，因为在模型中，我们每进入一个新的轨道区段都将发送该区段对应的信号，从而，最新信号可以在列车和控制器之间形成共享信息，以信号为基础的共享信息是验证并发程序所必需的。

此外，因我们的程序在最外层均为无限循环的程序形式（循环条件为 $true$），因此，谓词 F 用于表示循环的不可终止性。HML 程序 $\mathcal{O}$ 的基本推理过程可以表示如下：

$$\frac{\begin{array}{l}\text{程序 } \mathcal{C}, \mathcal{T}_1 \text{ 和 } \mathcal{T}_2 \text{ 的独立推理过程...}\end{array}}{\begin{array}{l}\mathtt{T}:\{\beta_{\mathcal{C}}\}\ \ [\mathcal{C}]\ \ \{\langle\beta_{\mathcal{C}}[*'/*] \bowtie \mathtt{F}\rangle\},\\ \mathtt{T}:\{\beta_{\mathcal{T}_1}\}\ [\mathcal{T}_1]\ \{\langle\beta_{\mathcal{T}_1}[*'/*] \bowtie \mathtt{F}\rangle\},\\ \mathtt{T}:\{\beta_{\mathcal{T}_2}\}\ [\mathcal{T}_2]\ \{\langle\beta_{\mathcal{T}_2}[*'/*] \bowtie \mathtt{F}\rangle\}\end{array}}$$

$$\frac{\begin{array}{l}\beta \Rightarrow (\beta_{\mathcal{C}} \wedge \beta_{\mathcal{T}_1} \wedge \beta_{\mathcal{T}_2}),\\ \mathtt{T}:\{\beta_{\mathcal{C}} \wedge \beta_{\mathcal{T}_1} \wedge \beta_{\mathcal{T}_2}\}\\ \qquad [\mathcal{C} \parallel \mathcal{T}_1 \parallel \mathcal{T}_2]\\ \quad \{\langle(\beta_{\mathcal{C}} \wedge \beta_{\mathcal{T}_1} \wedge \beta_{\mathcal{T}_2})[*'/*] \bowtie \mathtt{F}\rangle\},\\ (\beta_{\mathcal{C}} \wedge \beta_{\mathcal{T}_1} \wedge \beta_{\mathcal{T}_2})[*'/*] \Rightarrow \varsigma,\ \mathtt{F} \Rightarrow \tau\end{array}}{\mathtt{T}:\{\beta\}\ [\mathcal{O}]\ \{\langle\varsigma \bowtie \tau\rangle\}}$$

其中，程序 $\mathcal{C}$，$\mathcal{T}_1$ 和 $\mathcal{T}_2$ 之间不共享离散和连续变量，在程序中输出信号也不共享，因此各自的推理可以独立地完成。另外，我们的断言设计中并没有关注信号序号的相关性质，因此，在此省略了序号的合并操作。

4.7　本章小结

本章主要基于混成关系理论和霍尔逻辑，提出了混成系统建模语言 HML 的一套验证系统。使用 HML 建模语言，我们对经典的水箱水位控制模型进行了建模，并在我们提出的推理规则基础上对模型进行了分析。此外，对于 HML 并发模型，我们建立了共享轨道的地铁系统的 HML 模型，并通过信号之间的关系建立了模型中组件之间的联系，并展示了基本的推理过程。在下一章中，我们将对 HML 语言进行扩展并将其模型自动转换为 SMT（Satisfiability Modulo Theories）公式，以实现 HML 模型的自动验证。

第五章　基于 SMT 的混成系统仿真与验证

本章，我们对前一章中介绍的混成系统建模语言 HML 进行了扩展，在语言层面主要的变化是加入了程序模板结构，以便于在建模过程中实现 HML 程序代码的复用。此外，为了对 HML 模型进行自动分析和验证，我们将 HML 模型进行了自动转换，转换之后的模型以 SMT 公式的形式表现。基于 SMT 求解器 dReal[53]，我们开发了一个原型工具可以实现模型的仿真和可达性分析。为了检验我们所提出的方法的有效性，我们对一个倒立摆（Inverted Pendulum）控制系统进行了建模并在原型工具中进行了分析，结果表明，我们的方法可以有效地支撑使用 HML 语言构建的混成系统模型的仿真以及有界可达性分析。

5.1　引言

目前，混成系统建模语言有：混成自动机（Hybrid Automata [7]）、混成 CSP（Hybrid CSP [74, 31]）、Simulink/Stateflow、混成进程代数（HyPA [39]）以及混成程序（Hybrid Programs [117]）等。本章，我们主要采用的建模语言是由何积丰院士提出的混成系统建模语言 [76]，其基本特点是，在对混成系统进行建模时，采用抽象的信号机制来表达系统内部各组件之间的复杂交互行为，以简化混成系统的建模。为了适应实际的建模需求，我们对该语言的语法作了微小的改动，并增加了变量的基本数据类型声明、变量的取值范围有界约束等。我们知道，在软件工程的复用原则中，代码重用是一个非常基础和重要的软件复用方法，其主要通过直接利用或者封装系统开发过程中已有的程序代码来降低软件冗余程度，以实现代码结构的优化，提升程序的可读性和可维护性，节约软件开发和维护的成

本，降低软件的开发周期。基于软件复用的基本思想，我们在建模语言中引入模板（Template）以实现模型代码的复用。模板是一个有限的程序语句组成的集合，并且在使用时可通过指定参数来实例化模板，不同的参数可对应不同的实例模型，从而实现了模型代码的复用。在我们的建模语言中，代码重用可以适用于粗粒度的应用方式，如：对整个程序语句块进行重用；也可以用于细粒度的语句共享，比如，可以只重用一个微分方程或者控制逻辑中的一个简单的赋值操作等等。

为了实现对模型的自动分析，我们将混成系统模型自动转换到 SMT 公式，然后利用 SMT 求解器 dReal[1] [53] 的可满足性求解技术，实现模型的自动化分析和验证。SMT 求解器 dReal 支持实数域上的包含非线性函数的 SMT 问题的分析，所支持的非线性函数有：三角函数、指数函数以及对数函数等。SMT 求解器 dReal 基于 OpenSMT[26] 来实现 DPLL(T) 框架 [106]，并利用 RealPaver[58] 来判定区间约束的可满足性（Interval Constraint Satisfaction）。对于 SMT 公式，dReal 的分析结果主要有 UNSAT 和 δ-SAT 两种，其中 δ 是一个正的有理数表示误差范围。此外，dReal 还提供一些验证结果文件，当分析结果为不可满足时，工具会产生相关的证明；若满足，则工具会提供输入公式的可满足解。基于 dReal 提供的分析数据，我们可以实现混成系统的仿真和可达性分析。我们整合了模型转换和形式化分析的过程，并开发了一个开源的原型工具，提供基本的自动化验证和仿真结果的图形化展示功能。

在本章的案例分析部分，我们对一个具有非线性行为的倒立摆进行建模，并进行了仿真和可达性分析。在我们的模型中，倒立摆由 PID（Proportional-Integral-Derivative [121]）控制器进行控制。PID 控制器是一种控制回路反馈机制，在各种工业控制系统中应用极为广泛。一般地，PID 控制器含有三个可调控制参数，分别为比例增益（proportional gain）K_p、积分增益（integral gain）K_i，以及微分增益（derivative gain）K_d。在案例分析过程中，我们首先通过仿真发现适用于倒立摆系统的控制参数，然后，这些参数用于实现倒立摆的离散 PID

[1] http://dreal.github.io

控制器（采样时间为 0.08 s）。可达性分析主要用于检验在 PID 控制器的控制之下倒立摆位置的稳定性。

本章研究工作的主要贡献总结如下：

- 我们扩展了混成系统建模语言 HML，在语言层面提供模板机制用于实现代码重用。扩展后的语言可以实现混成系统在粗粒度和细粒度级别上的模型复用，从而可适用于规模较大的混成系统形式化模型的构建。
- 我们提供了一种将 HML 模型自动转换为 SMT 求解器可以处理的逻辑公式的方法，并整合了相关的转换和形式化分析过程，开发了对应的原型工具。HML 模型的仿真和可达性分析可以在我们提出的原型工具中有效地进行。
- 在本章中，我们对倒立摆系统的连续和离散 PID 控制模型进行了分析，同时检验了原型工具对非线性混成系统的建模和形式化分析能力。

本章余下部分组织如下：第5.2节介绍建模语言的语法及其扩展。由 HML 模型到 dReal 的转换在第5.3节中进行了详细阐述。在第5.4节中，我们对倒立摆系统进行了建模和分析。

5.2　建模语言

我们在本章中所采用的混成系统建模语言 HML 的语法如下所示：

$$
\begin{aligned}
AP &::= \texttt{skip} \mid v = e \mid !s \mid \texttt{wait}(e) \\
EQ &::= R(v, \dot{v}) \mid R(v, \dot{v})\ \texttt{init}\ v_0 \mid EQ \,||\, EQ \\
P &::= AP \mid P; P \mid \{P \,||\, P\} \mid (P \langle\, b \,\rangle P) \mid EQ\ \texttt{until}\ g \\
&\quad \mid \texttt{when}\{\ G\ \} \mid \texttt{while}\ (b)\{\ P\ \} \mid Id(e_s) \\
g &::= @(s) \mid b \mid g\ \langle \texttt{and} \rangle\ g \mid g\ \langle \texttt{or} \rangle\ g \\
b &::= \texttt{true} \mid \texttt{false} \mid v \circ c \mid \sim b \mid b\ \texttt{and}\ b \mid b\ \texttt{or}\ b \\
G &::= (g\ \texttt{then}\ P) \mid G,G
\end{aligned}
$$

其中，$\circ \in \{>=, >, ==, <, <=\}$。为了方便以纯文本的方式建模，我们对第四章中的 HML 语言的做了如下修改：

- 赋值语句从形式 $v := e$ 替换为 $v = e$，与传统的编程语言如：C、Java 等一致。
- 等待语句从形式 `suspend`(e) 替换为 `wait`(e)，这样可以使 HML 语言的使用者更加容易理解。
- 程序并发从形式 $P \parallel P$ 替换为 $\{P \parallel P\}$，修改之后可以更好地与联立方程组相区分，而且，当并发程序较大时，使用括号可以增强模型代码的可读性。
- 条件选择从形式 $P \triangleleft b \triangleright P$ 改为 $(P \langle\, b \,\rangle P)$，这样更易于以文本方式建模。
- 等待选择程序从形式 `when`(G) 替换为 `when`$\{\ G\ \}$，使用大括号进行分隔可以提供更好的可读性。同样地，循环语句从形式 `while` b `do` P `od` 改为 `while` $(b)\{\ P\ \}$ 也是为了可读性。此外，选择分支的分隔符 $[\!]$ 替换为逗号，卫兵条件与子程序之间的分隔符以 `then` 代替。
- 删除了卫兵条件 ϵ，因为我们可以用布尔条件 `true` 实现同样的功能。
- 信号 s 的卫兵条件以 $@(s)$ 的形式表示，可以方便模型转换的具体实现，也可以在模型中起到强调的作用。
- 符号 $\odot$ 和 $\oplus$ 分别替换为 ⟨`and`⟩ 和 ⟨`or`⟩ 以方便文本方式建模。
- 等价关系运算符 $=$ 替换为 $==$，与赋值运算符相区分。
- 逻辑非运算符 $\neg$ 替换为波浪号 $\sim$，逻辑与运算符 $\wedge$ 替换为 `and`，逻辑或 $\vee$ 替换为 `or`。
- 增加模板实例化语句 $Id(e_s)$，其中 Id 代表模板名，e_s 为参数列表。

此外，我们为 HML 语言定义四种数据类型如下：

$$Type ::= \texttt{Signal} \mid \texttt{boolean} \mid \texttt{int} \mid \texttt{float}$$

其中，类型 Signal 用于声明信号变量，类型 boolean 用于布尔变量，类型 int 和 float 分别用于整数和实数变量。另外，对于常量的定义，我们在类型前面使用关键字 final 来表示。

为了在模型中表示变量的有界约束以及实现代码重用等功能，我们定义如下语法结构：

$$Constraint ::= v \ \texttt{in} \ [l,\ h] \tag{5.1}$$

$$Template ::= \texttt{Template} \ Id(f_s)\{\ P\ \} \tag{5.2}$$

$$Main ::= \texttt{Main}\ \{\ P\ \} \tag{5.3}$$

其中，(5.1) 用于指定 HML 模型中变量的取值范围，如：h in $[0,1]$ 表示变量 h 的取值范围是区间 $[0,1]$；(5.2) 表示模板的语法，其中 Template 是关键字，Id 是模板名称，f_s 表示形参列表，P 为模板体程序；(5.3) 用于指定 HML 程序的主方法，即程序开始执行的入口。

下面，我们给出水箱模型的代码。首先是变量定义和约束声明：

```
1  Signal  on;
2  Signal  off;
3  final  float deta=0.5, accin=1, decout=1;
4  final  int  MaxwaterLevel=10, LowestwaterLevel=5;
5  float waterLevel=8.5, clock = 0, global = 0;
6  waterLevel in [5, 10];
7  time in [0, 100];
8  clock in [0, 100];
9  global in [0, 200];
```

其中，常量 $accin$ 表示控制阀打开后水位上升的速度，$decout$ 为控制阀关闭后水位下降的速度。常量 $MaxwaterLever$ 和 $LowestwaterLevel$ 分别表示水位最高和最低值。下面的程序代码表示模板声明和主程序入口 Main：

```
1  Template  open(float h, float a, int  H, float e){
2    (dot  h = a) until  (h >= H-e)
3  }
4
5  Template  closed(float h, float b, int  L, float e) {
6     (dot  h = -b) until  (h <= L+e)
7  }
8
9  Main  {
10      while (true){
11        !on;
12        open(waterLevel,accin,MaxwaterLevel,deta);
13        !off;
14        closed(waterLevel,decout,LowestwaterLevel,deta)
15      }
16 }
```

其中，模板 *open* 和 *closed* 分别用于表示水位上升和下降的动态行为。第 12 和 14 行是模板的实例化语句。变量的导数通过在变量前加 dot来表示，如：dot h 表示变量 h 的一阶时间导数 $\dot{h}$。

5.3　模型转换

在这一节中，我们将详细阐述将 HML 模型转换为 dReal 可接受的 SMT[18] 公式的过程。首先，我们对 dReal 工具内部所采用的用于混成系统表达的逻辑公式进行简单介绍，然后，再给出从 HML 模型到 SMT 公式的转换过程。

5.3.1　$\mathcal{L}_{\mathbb{R}_\mathcal{F}}$-公式

令 $\mathcal{F}$ 为可计算实函数的一个集合，称为 Type-2[2] 可计算函数 [137]。在本文讨论的范围内，我们假设在模型中所用到的函数均是 Type-2 可计算函数，如：

[2]在本文中，Type-2 函数为形如 $f:\mathbb{R}\to\mathbb{R}$ 的函数

利普希茨连续的常微分方程的解、对数函数、指数函数、三角函数和多项式函数等等。在 dReal 中，混成系统以 $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$ 公式的形式表示，其定义如下：

定义 5.3.1 ($\mathcal{L}_{\mathbb{R}_\mathcal{F}}$ 公式 [51]). *令 $\mathcal{F}$ 为 Type-2 可计算函数（常数可作为零元函数）的一个集合，则 $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$ 公式定义如下：*

$$t ::= v \mid f(t(\vec{v})), \text{其中 } f \in \mathcal{F}$$

$$\phi ::= t(\vec{v}) > 0 \mid t(\vec{v}) \geq 0 \mid \phi \wedge \phi \mid \phi \vee \phi \mid \exists x_i.\phi \mid \forall x_i.\phi,$$

其中，v 代表变量或常量，$\vec{v} = (v_1, v_2, ..., v_n)$，$t$ 为 $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$ 的项，ϕ 为 $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$ 的公式。

含有逻辑非运算符的公式可以通过等价代换统一表示为 $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$ 公式：

$$\neg(t > 0) \equiv -t \geq 0, \qquad \neg(\phi_1 \wedge \phi_2) \equiv \neg\phi_1 \vee \neg\phi_2, \qquad \neg\exists x_i.\phi \equiv \forall x_i.\neg\phi,$$
$$\neg(t \geq 0) \equiv -t > 0, \qquad \neg(\phi_1 \vee \phi_2) \equiv \neg\phi_1 \wedge \neg\phi_2, \qquad \neg\forall x_i.\phi \equiv \exists x_i.\neg\phi.$$

逻辑蕴涵式也可以进行相应的转换：$\phi_1 \to \phi_2 \equiv \neg\phi_1 \vee \phi_2$。此外，原子公式 $(t(\vec{v}) = 0)$ 可以用 $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$ 公式 $(-|t(\vec{v})| \geq 0)$ 表示，其中 $|\cdot|$ 为取绝对值函数。含有小于关系的公式也可以统一为 $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$ 公式，即：$t(\vec{v}) < 0 \equiv -t(\vec{v}) > 0$，以及 $t(\vec{v}) \leq 0 \equiv -t(\vec{v}) \geq 0$。当关系式右侧值不为 0 时，也可以将公式统一，如：公式 $s < t$ 可以重写为 $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$ 公式 $(t - s) > 0$。

由于我们在本章中关注的是变量有界情况下的模型验证，因此，需要在公式中表示有界特性。下面，我们给出有界量词公式的定义。

定义 5.3.2 (有界量词公式 [51]). *有界存在量词（$\exists^I$）公式和有界全称量词（$\forall^I$）公式定义如下：*

$$\exists^I v.\phi =_{def} \exists v.(v \in I \wedge \phi);$$

$$\forall^I v.\phi =_{def} \forall v.(v \in I \to \phi),$$

其中，区间 $I = [l, h]$，l 和 h 均为 $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$ 的项。

一个有界 $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$ 子句的形式为 $Q_1^{I_1} v_1 \cdots Q_n^{I_n} v_n.\phi(v_1, ..., v_n)$，其中，$Q_i^{I_i}$ 为有界量词，$\phi(v_1, ..., v_n)$ 为不含量词的 $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$ 公式。dReal 解决的 SMT 判定问题为：

1. 判定完全存在量词子句的真值，称为 Σ_1-SMT 判定问题，其对应的逻辑公式形式为：$\exists^{I_1} v_1 \cdots \exists^{I_n} v_n.\phi(v_1, ..., v_n)$。

2. 判定部分全称量词子句的真值，称为 Σ_2-SMT 判定问题，其对应的逻辑公式形式为：$\exists^{I_1} v_1 \cdots \exists^{I_n} v_n \forall^{I_{n+1}} v_{n+1} \cdots \forall^{I_m} v_m.\phi(v_1, ..., v_m)$。

SMT 求解器 dReal 基于 δ 完备判定过程 [51]，可以处理含有可计算的非线性实函数的一阶逻辑公式判定问题。在该判定过程中，每个一阶公式都有相应的 δ 变式用于表示 δ 扰动的约束，其定义如下：

定义 5.3.3 (δ 变式). 令 $\mathbb{Q}^+$ 为正有理数的集合，$\delta \in \mathbb{Q}^+ \cup \{0\}$，$\mathcal{L}_{\mathbb{R}_\mathcal{F}}$ 子句 $\varphi = Q_1^{I_1} v_1 \cdots Q_m^{I_m} v_m.\phi[t_i(\vec{v}) > 0; t_j(\vec{v}) \geq 0]$，其中，$i \in \{1, ..., k\}$，$j \in \{k+1, ..., m\}$。$\delta$-弱化公式 φ^δ 定义为：

$$Q_1^{I_1} v_1 \cdots Q_m^{I_m} v_m.\phi[t_i(\vec{v}) > -\delta; t_j(\vec{v}) \geq -\delta]$$

φ^δ 是由 φ 通过将形如 $(t_i > 0)$ 的公式替换为 $(t_i > -\delta)$，以及将形如 $(t_i \geq 0)$ 的公式替换为 $(t_i \geq -\delta)$ 而得到的。

对于任何 $\mathcal{L}_{\mathbb{R}_\mathcal{F}}$ 子句 φ，dReal 将返回如下判定结果 [53]：

- UNSAT：φ 是不可满足的，此时模型无数值扰动 δ
- δ-SAT：φ^δ 是可满足的，模型有数值扰动 δ

基于上述结果，我们可以知道，在数值扰动 δ 影响下，混成系统模型是否满足某一性质或者可以到达某一状态，以及在没有扰动的情况下，证明性质的不可满足性。

5.3.2 串行模型转换

一般来说，混成系统的行为是离散和连续行为的交替往复。在 HML 模型的转换中，我们用 n 来表示最大的模型展开深度，模型的每层展开包含有离散和

连续行为所对应的逻辑公式。在 HML 程序语句转换后的公式中，我们需要表示变量在程序执行前后的变化，因此，我们需要在公式中标记变量的用途。

在本章中，我们约定，符号 ρ 用于标记变量的初值，符号 τ 用于标记程序执行后变量的新值。另外，为了区分不同展开层次的行为，程序语句所处的展开层次也需要进行标记。所以，对于公式中的变量，我们共有两个标记信息。为了表述方便，我们将这些标记信息放在变量名的下标处显示。举例来说，变量 $v_{0,\rho}$ 代表的是变量 v 在深度为 0 的模型展开时的初值。

下面，我们对五种离散动作的转换进行说明。需要注意的是，虽然实际的 SMT 公式是以前缀的形式表示的，但为了公式的可读性和表述的方便，我们在文中将以中缀的方式表示，并对某些特殊的公式以抽象的形式给出。

赋值

假设当前的模型展开深度为 m，最大深度为 n，则 $0 \leq m \leq n$。对于赋值语句 $x = e(\vec{v})$，我们将其转换为如下公式：

$$x_{m,\tau} = e(\vec{v}_{m,\rho})$$

其中，$e(\vec{v}_{m,\rho})$ 为表达式 $e(\vec{v})$ 通过将 $\vec{v}$ 替换为初值 $\vec{v}_{m,\rho}$ 而得到的新的表达式。例如，赋值语句 $x = x + y + 1$ 可以转换为公式 $x_{m,\tau} = x_{m,\rho} + y_{m,\rho} + 1$。

信号发送

信号发送程序 $!s$ 可以当作为一个离散赋值操作。信号产生时，其最重要的信息是发生的时刻。在我们的转换中，信号可以看做是类型为 `float` 的变量。同时，在我们的模型中变量 $global$ 可以表示全局时间，因而，我们可以将信号的发送转换为如下简单的公式：

$$s_m = global_{m,\rho}$$

其中，$global_{m,\rho}$ 代表信号 s 产生的时刻点。需要注意的是，我们在信号的变量名中没有下标 τ，这是因为，信号的产生时间是瞬时的，其取值也不依赖其他变量的取值，而且在连续行为中是不会产生信号发送的。

串行组合

对于两个离散动作的串行组合 $P_1;P_2$，若 P_1 转换后的公式为 $\mathcal{F}_1(\vec{x}^1_{m,\tau},\vec{x}^1_{m,\rho})\wedge(\vec{x}^1_{m,\tau}=\vec{e}_1(\vec{x}^1_{m,\rho}))$，$P_2$ 转换后的公式为 $\mathcal{F}_2(\vec{x}^2_{m,\tau},\vec{x}^2_{m,\rho})\wedge(\vec{x}^2_{m,\tau}=\vec{e}_2(\vec{x}^2_{m,\rho}))$，那么，我们有串行组合公式：

$$\mathcal{F}_1(\vec{e}_1(\vec{x}_{m,\rho}),\vec{x}_{m,\rho})\wedge\mathcal{F}_2(\vec{x}_{m,\tau},\vec{e}_1(\vec{x}_{m,\rho}))\wedge(\vec{x}_{m,\tau}=\vec{e}_2(\vec{e}_1(\vec{x}_{m,\rho})))$$

其中，$\vec{x}_{m,\rho}$ 代表 $\vec{x}$ 的初值，公式中的上标（1 和 2）只是为了区分 P_1 和 P_2 的公式用的。那么，我们有 $\vec{x}^1_{m,\rho}=\vec{x}_{m,\rho}$，$\vec{x}^2_{m,\rho}=\vec{x}^1_{m,\tau}$，以及 $\vec{x}^2_{m,\tau}=\vec{x}_{m,\tau}$。在下文中，我们将以形如 $\vec{x}_{m,\tau}=\vec{e}_2(\vec{e}_1(\vec{x}_{m,\rho}))$ 的公式来表示串行组合 $[\vec{x}^1_{m,\tau}=\vec{e}_1(\vec{x}^1_{m,\rho});\vec{x}^2_{m,\tau}=\vec{e}_2(\vec{x}^2_{m,\rho})]$ 转换后的公式。

条件选择

条件选择语句 $(P_1\ \langle\ b\ \rangle\ P_2)$ 可以表示成如下公式：

$$[b(\vec{x}_{m,\rho})\wedge\vec{x}_{m,\tau}=\vec{e}_1(\vec{x}_{m,\rho})]\vee[\neg b(\vec{x}_{m,\rho})\wedge\vec{x}_{m,\tau}=\vec{e}_2(\vec{x}_{m,\rho})]$$

其中，$b(\vec{x}_{m,\rho})$ 代表转换后的布尔表达式公式。表达式 $\vec{e}_1$ 和 $\vec{e}_2$ 分别用于程序 P_1 和 P_2 转换后的公式。一般地，条件选择对应着程序的多个可能的执行路径。

循环

对于循环语句 `while` $(b)\{\ P\ \}$，若用自然数 i 表示循环的总层数，则当 $i=0$ 时，程序转换为：

$$LP_0=\neg b(\vec{x}_{m,\rho})$$

如果 $i=1$，

$$LP_1=[b(\vec{x}_{m,\rho})\wedge\vec{x}_{m,\tau}=\vec{e}(\vec{x}_{m,\rho})]\wedge[\neg b(\vec{x}_{m,\tau})]$$

如果 $i=2$，

$$LP_2=[b(\vec{x}_{m,\rho})\wedge\vec{x}_{m,\tau}=\vec{e}(\vec{e}(\vec{x}_{m,\rho}))]\wedge[\neg b(\vec{x}_{m,\tau})]$$

因此，一般地，我们有 $LP_i=[b(\vec{x}_{m,\rho})\wedge\vec{x}_{m,\tau}=\vec{e}^{\,i}(\vec{x}_{m,\rho})]\wedge[\neg b(\vec{x}_{m,\tau})]$，其中 $\vec{e}^{\,i}(\vec{x}_{m,\rho})=\vec{e}(\vec{e}^{\,i-1}(\vec{x}_{m,\rho}))$。因为我们无法对无限循环进行完全展开，因此在本文

中，我们约定循环体中将包含有连续行为程序。

下面，我们对连续行为进行处理，主要涉及三种程序语句。

等待

对于等待程序语句 wait$(e(\vec{x}))$，我们将其转换为微分方程的形式，其中连续变量为 $clock$，一阶时间导数为 1。变量 $clock$ 的值代表了从等待程序开始后的等待时间（即时间的积分）。相应的逻辑公式，以如下抽象的形式表示：

$$[\langle clock_{m,\tau}\rangle] = (\texttt{integral}\ [0, time_m]\ clock_{m,\rho}\ flow)$$

其中，$[\langle\cdot\rangle]$ 用于连续变量取值的表示，微分方程 $flow =_{def} (\frac{d[clock]}{d[t]} = 1)$，$clock$ 的初值为 $clock_{m,\rho}$，区间 $[0, time_m]$ 代表时间的取值范围，变量 $time_m$ 为连续行为的时间上界，关键字 integral 表示积分运算，dReal 可以将上述公式转换为相应的微分方程求解。同时，在模型转换时，我们还将加入如下的约束公式：

$$\forall t \in [0, time_m].\neg(clock_{m,\tau} \geq e(\vec{x}_{m,\rho}))$$

表示在程序终止之前，变量 $clock$ 的值始终小于表达式 e 的取值。对于程序执行结束的情况，我们也需要相应的公式来表示：

$$clock_{m,\tau} \geq e(\vec{x}_{m,\rho})$$

需要注意的是，该公式没有使用量词，因为终止行为只是发生在一个时刻点上的瞬时行为。

微分方程

对于形如 $(\dot{v} = e(v)$ init v_0 until $g(\vec{x}))$ 的微分方程，我们可以将其转换为如下公式：

$$[\langle v_{m,\tau}\rangle] = (\texttt{integral}\ [0, time_m]\ v_{m,\rho}\ flow)$$

其中，$v_{m,\rho} = v_0$，方程 $flow =_{def} (\frac{d[v]}{d[t]} = e(v))$。同样地，我们还有约束公式：

$$\forall t \in [0, time_m].(\neg g(\vec{x})_{m,\tau})$$

需要注意的是，不同的卫兵条件 g，其对应的约束公式的具体形式是不同的。下面是两种主要的形式：

- 对于信号卫兵 $@(s)$，其约束公式的具体形式为 $\neg(s_m \geq global_{m,\tau})$。若在连续行为开始时信号刚刚发出，则公式 $(s_m \geq global_{m,\tau})$ 为真，连续行为将立刻终止。

- 对于布尔条件 b，我们有约束公式 $\neg b(\vec{x}_{m,\tau})$。

当微分方程为联立方程组时，各方程转换后的 SMT 公式的合取即为最终的 SMT 公式。

等待选择

对于等待选择程序 when$\{(g$ then $P)\}$，可以转换如下：

$$[\langle clock_{m,\tau}\rangle] = (\texttt{integral}\ [0, time_m]\ clock_{m,\rho}\ flow)$$

其中，微分方程 $flow =_{def} (\frac{d[clock]}{d[t]} = 1)$，约束公式为

$$\forall t \in [0, time_m].(\neg g(\vec{x})_{m,\tau})$$

子程序 P 将用于下一深度（$m+1$）的模型展开。

令 $\mathcal{F}_D$ 和 $\mathcal{F}_C$ 分别代表离散和连续行为转换后的公式，则将两种公式进行组合需要对公式 $\mathcal{F}_D$ 进行换名操作，以建立当前深度的公式与上一深度的公式之间的关系。

一般地，对于变量 v，换名操作的实现过程是，先将公式中的变量 $v_{m,\rho}$ 替换为变量 $v_{m-1,\tau}$，然后将变量 $v_{m,\tau}$ 替换为 $v_{m,\rho}$。当 $m = 0$ 时，我们直接使用变量的初值。

HML 程序的执行往往存在着多种执行路径的选择，随着模型展开的深度越来越深，可能的执行路径的数量将呈指数级增长。对于该问题，我们提出路径动态合并算法对路径进行压缩。

在算法5.1中，D_j 和 C_j 分别表示某条路径第 j 层的转换中离散和连续行为对应的公式。函数 **size** 用于计算集合中元素的数量。函数 **merge** 的作用是将两条路径合并为一条路径。

对于两条路径（p_1 和 p_2）是否可以合并，我们根据以下条件进行判

算法 5.1： 路径动态合并

输入：
路径数目 $n \geq 2$，最大展开深度 $d \geq 0$；
路径集合 $P_s = \{\bigwedge_{j=0}^{m} \langle D_j, C_j \rangle_i \mid 0 \leq i \leq n, m \leq d\}$；

输出：
合并后的路径集合 $P_s^{'}$；

1: 从 P_s 中选择并删除一条路径 p，即 $P_s := P_s - \{p\}$；
2: 在 P_s 中查找路径，如果有一天路径 $q \in P_s$ 可以与 p 合并，则从 P_s 中删除 q，即 $P_s := P_s - \{q\}$，之后合并 $p := \mathbf{merge}(p, q)$，将 p 放入 $P_s^{'}$；
3: 判定 P_s 中元素的数量，如果 $\mathbf{size}(P_s) = 1$，则唯一的路径放入 $P_s^{'}$；否则如果 $\mathbf{size}(P_s) \geq 2$ 则继续步骤1；
4: **return** $P_s^{'}$；

断：1. 长度相同：$p_1.m = p_2.m$，m 表示路径的最大深度；2. 连续公式相同：$\forall j \in \{0, ..., m\} \bullet p_1.C_j = p_2.C_j$，两条路径同一层次的公式中连续行为对应的公式相同。

基于以上条件，合并的对象是模型中离散行为所对应的那部分公式，具体合并时，可以通过逻辑析取运算符来实现。在模型转换到达最大深度之前，我们将在每层的展开完成时进行一次合并，这样可以及时地减少路径的数量。

5.3.3 并发模型转换

在 HML 模型中，我们总共只有两类动态行为，即：离散动作 D 和连续行为 C。那么，两个 HML 程序的并发有三种基本的组合方式。

离散 - 连续

对于离散动作和连续行为的并发，我们有如下规则：

$$\frac{D \parallel C}{D \ ; \ C}$$

在这种情况下，离散动作先执行，然后连续行为在离散动作结束后发生。在模型转换时，我们也是按照以上顺序，先转换离散程序然后处理连续程序的转换。

离散 - 离散

算法 5.2: HML 模型单步动态转换过程

输入： 并发程序数量 $n \geq 2$，并发程序集合 $\{P_i \mid 1 \leq i \leq n\}$
输出： SMT 公式 $\mathcal{F}_u$

```
for i = 1 to n do
  while true do
    act = P_i.getNextAct(); // 程序 P_i 的当前动作
    if act is null then
      break;
    end if
    if act is a discrete action then
      S_d = S_d ∪ {act}; //S_d 离散动作集合，初始时为 ∅
    else
      if the guard of act is unsatisfiable then
        S_c = S_c ∪ {act}; break; //S_c 连续行为的集合
      end if
    end if
  end while
end for
F_u = unroll(S_d) ∧ unroll(S_c); //将离散和连续行为转换为 SMT 公式
```

对于两个离散动作（D_1 和 D_2）的并发，我们有如下规则：

$$\frac{D_1 \parallel D_2}{(D_1 \ ; \ D_2) \ \vee \ (D_2 \ ; \ D_1)}$$

由于两个离散动作执行的先后是非确定的，因此，在转换时可以使用逻辑或操作符表示。

连续 - 连续

对于两个连续行为（C_1 和 C_2）的并发，我们有如下基本规则：

$$\frac{C_1 \parallel C_2}{[(\sim_{1,2}) \ \texttt{until} \ g_{1,2}] \ ; \ [(g_1 \wedge g_2 \wedge \texttt{skip}) \vee (g_1 \wedge \neg g_2 \wedge C_2) \vee (\neg g_1 \wedge g_2 \wedge C_1)]}$$

其中，$(\sim_{1,2})$ 为从 C_1 和 C_2 中抽取出来的方程组成的联立方程组，g_1 和 g_2 分别是 C_1 和 C_2 的卫兵条件。新的卫兵条件 $g_{1,2}$ 为 C_1 和 C_2 的合并卫兵，$g_{1,2} = g_1 \vee g_2$。分号（;）后表示程序的三种可能性，第一，两个卫兵条件都满足了则两个连续行为都结束了；第二，g_1 满足了，但 g_2 没有满足，则 C_1 结束了，C_2 还没有结束；第三中情况表示 C_2 结束了，C_1 还没有结束。

例 5.3.1. 两个连续程序的并发 $C_1 \parallel C_2$，其中，

$$C_1 =_{df} \ (\dot{p} = v \ \texttt{init} \ 0 \parallel \dot{v} = 0.5 \ \texttt{init} \ 0) \ \texttt{until} \ (p \geq 100);$$

$$C_2 =_{df} \ \texttt{wait}(2)$$

则，并发程序 $C_1 \parallel C_2$ 可以转换为：

$$[\langle clock_{m,\tau}, p_{m,\tau}, v_{m,\tau} \rangle] = (\texttt{integral} \ [0, time_m] \ clock_{m,\rho} \ p_{m,\rho} \ v_{m,\rho} \ flow)$$

其中，$flow =_{def} (\frac{d[clock]}{d[t]} = 1 \wedge \frac{d[p]}{d[t]} = v \wedge \frac{d[v]}{d[t]} = 0.5)$，相应的约束公式为：

$$\forall t \in [0, time_m].\neg(clock_{m,\tau} \geq 2) \wedge \forall t \in [0, time_m].\neg(p_{m,\tau} \geq 100)$$

变量的初值约束为：$clock_{m,\rho} = 0$，$p_{m,\rho} = 0$ 以及 $v_{m,\rho} = 0$。根据连续行为并发规则，可以对 C_1 和 C_2 进行转换并代入分号后的公式中。

5.3.4 HML 模型的动态转换

在模型的并发展开时，如果我们有 n 个并发程序，则我们在转换时可能会同时遇到 n 个并发的离散动作，n 个动作的并发将产生 $n!$ 种不同的执行顺序。当 n 的值越来越大时，完全处理所有可能的执行顺序将是一件极具挑战的任务。

在此，我们提出一个基于 SMT 求解器 dReal 的 HML 模型的动态转换过程。该转换过程是对 HML 模型进行仿真的基础。算法5.2展示了 HML 模型单步动态转换的基本过程。其中，集合 S_d 用于存储离散动作，在转换过程中，程序 P_i 的卫兵条件通过调用 dReal 来进行检查，当连续行为的卫兵条件不满足时，我们将该连续行为对应的程序加入集合 S_c。最后将集合 S_c 和 S_d 中所有程序的转换公式进行合取得到最终的 SMT 公式。

在处理 S_d 中的离散动作时，我们可以考虑各种执行顺序，也可以随机地选择一种执行顺序来做转换，但是，在本文中，我们的仿真是基于离散动作放入集合时的顺序来做的。虽然我们在算法中没有说明其他类型的程序，但是，条件选择和循环程序同样可以在转换过程中基于 dReal 来做动态检查。

5.4　案例分析

在这一节中，我们将对一个安装在一个小推车上的倒立摆（如图5.1所示）进行建模并分析。在本案例中，为了控制倒立摆的位置，使之尽可能地处于垂直于小车的位置，我们使用 PID 控制器进行控制。

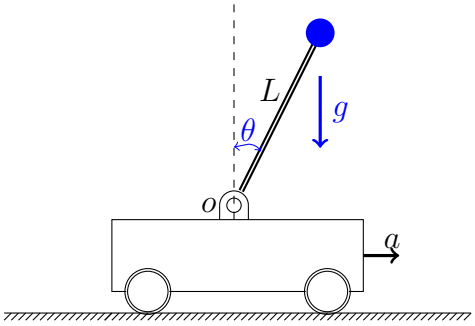


图 5.1: 倒立摆

在图5.1中，倒立摆的摆杆的一端为一球体，另一端安装在小车上，且摆杆以该点（图中点 o）为轴心能在垂直平面上自由摆动。小车可以在水平方向上前后移动。当小车处于静止状态且摆杆与竖直方向的夹角 θ 大于 0 时，摆杆将在小球的重力作用下，以点 o 为轴心下落。我们的目标是通过控制小车的前后移动，使摆杆能够稳定地处于竖直方向上，即摆杆与竖直方向的夹角 θ 要尽可能接近 0。如下是图5.1所用到的变量的解释：

- L: 摆杆的长度，1 m；
- θ: 摆杆与竖直方向的夹角；
- a: 小车的加速度；
- g: 重力加速度，9.8 $\mathrm{m/s^2}$。

以上变量之间的关系可以用如下非线性微分方程表示：

$$L\frac{d^2\theta(t)}{dt^2} = g\sin[\theta(t)] - a(t)\cos[\theta(t)]$$

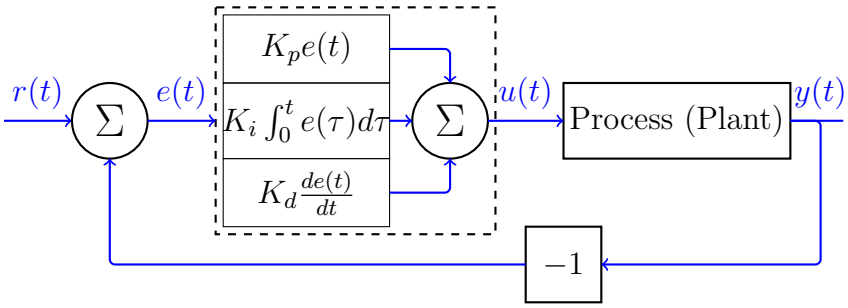


图 5.2: PID 控制器的基本结构

其中，t 代表时间。倒立摆的控制可以通过 PID 控制器 [29] 或者模糊逻辑控制器 [115, 140] 来实现。

图5.2展示的是一个处于反馈环路中的 PID 控制器。PID 控制器涉及三个项的取值：比例、积分和微分项，分别以 P、I 和 D 表示。项 P 依赖当前的控制误差，项 I 依赖于误差的积分值，项 D 基于误差变化率预测将来的控制误差。以上三个项值的和可以用于计算 PID 控制器的控制输出。在图5.2中，控制输出为 $u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt}$，其中，$e(t)$ 表示控制误差。参数 K_p、K_i 和 K_d 分别是比例、积分和微分增益 [121]。图中，过程（process/plant）的输出称为过程变量（process variable），用 $y(t)$ 表示。过程变量的期望值称为设定值（set point），用 $r(t)$ 表示。

在本案例中，我们设定 $r(t) = 0$ 表示我们希望控制倒立摆的摆杆使之处于竖直状态，即夹角 θ 等于 0。所以，控制误差可以用 $\theta(t)$ 表示。在图5.1所示模型中，倒立摆的角度是通过小车的加速度的调整来控制的，因而对于倒立摆模型，控制输出 $u(t) = a(t)$。倒立摆的 HML 模型将在5.4.1节和5.4.2节中介绍。

5.4.1　连续控制

倒立摆的 HML 模型如图5.3所示。其中，比例项参数 K_p 的取值为 80（图5.3中变量 kp），参数 K_i 和 K_d 取值为 0，因此在 HML 模型中没有给出相关的变量声明。倒立摆的初始位置 $position = 0.2$ rad（弧度），初始速度 $velocity = 0$ rad/s。因此，在图5.3的 HML 模型中，小车的加速度为 $a(t) = K_p\theta(t)$，对应于

```
1  final float g=9.8, kp=80;
2  float position=0.2, velocity=0, clock=0, global=0;
3  //变量约束
4  position in [-10, 10];
5  velocity in [-40, 40];
6  time in [0, 10];
7  clock in [0, 10];
8  global in [0, 10];
9  //模板
10 Template  Pendulum(float theta, float omega){
11     (dot  theta = omega)
12      ||
13     (dot  omega = g*sin(theta) - (kp*theta)*cos(theta))
14     until  (false)
15 }
16 //主程序
17 Main {
18       Pendulum(position, velocity)
19 }
```

图 5.3: 倒立摆系统的 HML 模型

第 13 行的 HML 程序代码 `kp*theta`。

此外，需要注意的是，在代码的第 14 行中，我们设定连续行为的终止条件为 $false$，但 $false$ 是不可满足的，因此，该连续行为也就不可终止了。还有，在连续控制中，参数的值在一开始就设定了，在控制过程中不会对参数进行重置，参数的值在控制过程中保持不变，这也是我们称该控制方式为连续控制的原因。

在 HML 模型中，模板 $Pendulum$ 的实例化参数为 $position$ 和 $velocity$。因此，形参 $theta$ 对应的是倒立摆的位置（图5.1中的 θ），$omega$ 对应的是角速度 $velocity$。通过我们的原型工具的仿真，倒立摆的位置和角速度变化情况如图5.4和5.5所示。

由仿真结果可知，倒立摆的位置和速度均呈现振荡趋势，倒立摆左右摇摆

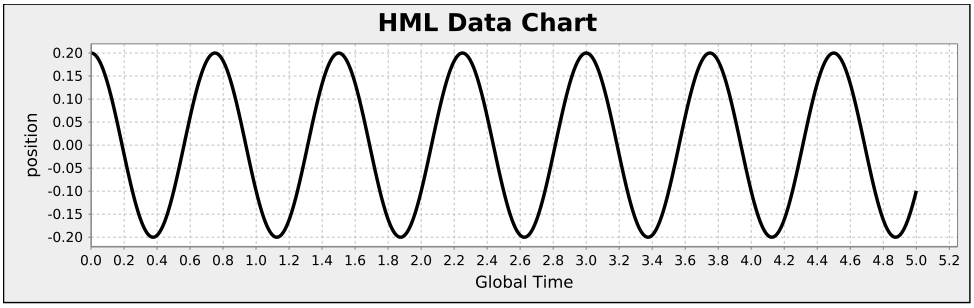


图 5.4: 倒立摆的位置。参数 $K_p = 80$，$K_i = 0$，$K_d = 0$。仿真时间 5 s

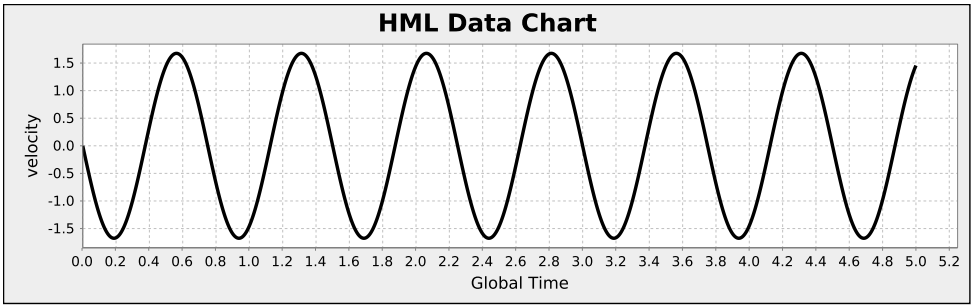


图 5.5: 倒立摆的角速度。参数 $K_p = 80$，$K_i = 0$，$K_d = 0$。仿真时间 5 s

不定。但是，我们的目标是尽快地让倒立摆稳定于竖直方向。下面，我们将修改增益 K_d 的值，相应地，修改图5.3中第 13 行代码，将 $(\texttt{kp} * \texttt{theta})$ 替换为 $(\texttt{kp} * \texttt{theta} + \texttt{kd} * \texttt{omega})$，并增加变量声明和约束条件，然后继续仿真。

图5.6为参数 $K_d = 5$ 时的仿真结果。倒立摆在开始时有轻微振荡，但是约两秒后，位置趋于 0 且没有发生再次振荡，位置较为稳定。

当我们逐渐增大 K_d，倒立摆的位置趋于稳定的速度不断加快。图5.7所示的仿真结果为 $K_d = 15$ 时倒立摆的位置变化情况。可以看出，在 0.6 s 之后，倒立摆就已稳定在 0 点位置处，而且整个控制过程中没有发生大幅的位置振荡。

当再次增大 K_d 的值，我们发现情况并没有变好，反而出现了超调现象（over control）。如图5.8所示，与图5.7相比较，控制所消耗的时间变长了。总之，随着 K_d 的增大，倒立摆控制所消耗的时间并非是可持续地缩短的。现在，基于连续控制模型的仿真结果，我们可以选定增益参数的值（$K_p = 80$，$K_i = 0$ 和 $K_d = 15$），从而进行5.4.2节中倒立摆的离散控制。

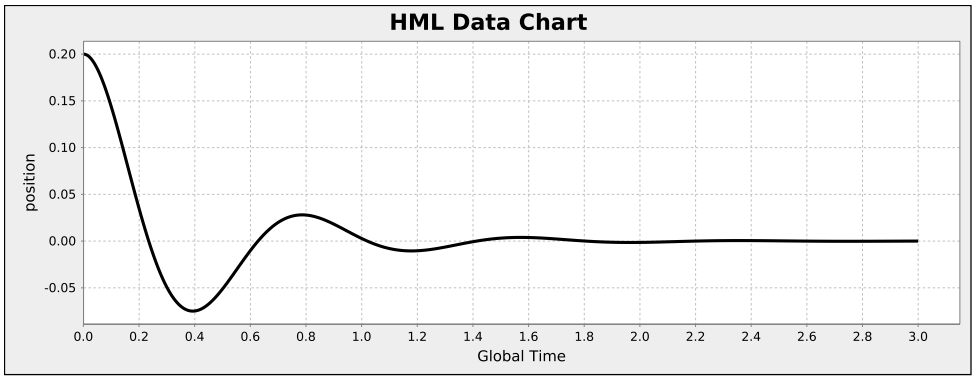


图 5.6: 倒立摆的位置。参数 $K_p = 80$，$K_i = 0$，$K_d = 5$。仿真时间 3 s

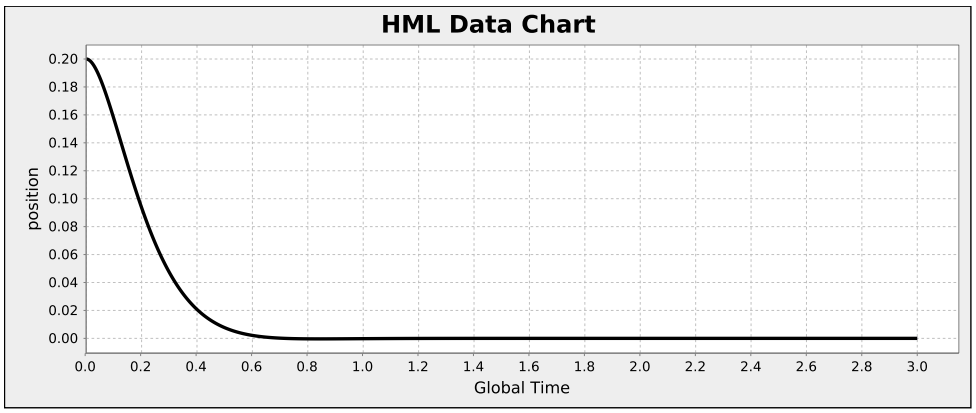


图 5.7: 倒立摆的位置。参数 $K_p = 80$，$K_i = 0$，$K_d = 15$。仿真时间 3 s

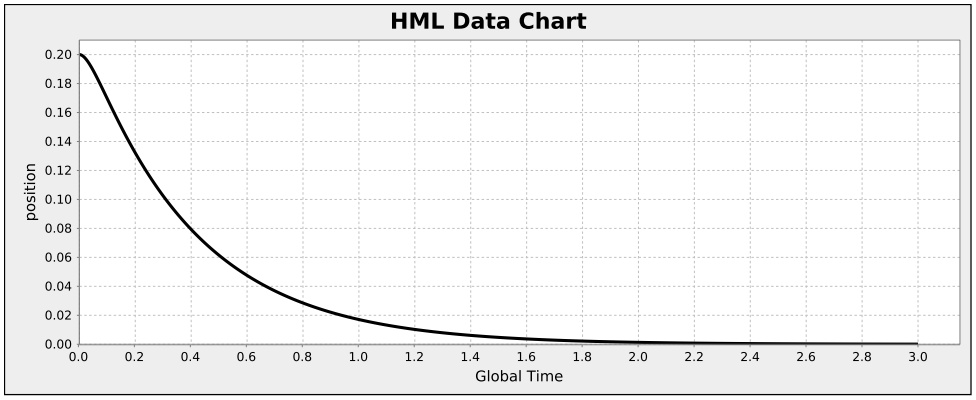


图 5.8: 倒立摆的位置。参数 $K_p = 80$，$K_i = 0$，$K_d = 30$。仿真时间 3 s

5.4.2　离散控制

图5.9显示的是倒立摆的离散控制模型。在离散控制模型中，参数 $K_p = 80$，$K_i = 0$，$K_d = 15$。我们设置的采样时间为 0.08 s，每隔 0.08 s 我们将计算一次

```
Template  Control(float a, float theta, float omega){
      while (true) {
           a = kp*theta  +  kd*omega;
           wait(0.08)
      }
}
Template  Pendulum(float theta, float omega, float a){
     (dot  a = 0)
      ||
     (dot  theta = omega)
      ||
     (dot  omega = g*sin(theta) - a*cos(theta))
     until  (false)
}
Main {
       {
                Pendulum(position, velocity, acc)
                ||
                Control(acc, position, velocity)
       }
 }
```

图 5.9: 倒立摆的离散控制模型。变量声明和约束已略去，变量 acc 代表小车的加速度

小车的加速度，并以新的加速度来控制小车的运动方向和速度，从而影响倒立摆的位置。因此，控制参数的值将可能会在控制过程中因系统状态（如倒立摆的位置、速度等）的变化而发生改变，这是离散控制与上一节的连续控制的不同之处。对于离散控制模型，我们的模型展开深度为 11，考虑以下性质验证：

1. 倒立摆的位置和角速度在离散控制下，可以处于 $-0.01 \sim +0.01$ 范围内，以谓词公式的形式可以表示如下：

$$(position \in [-0.01, 0.01]) \wedge (velocity \in [-0.01, 0.01])$$

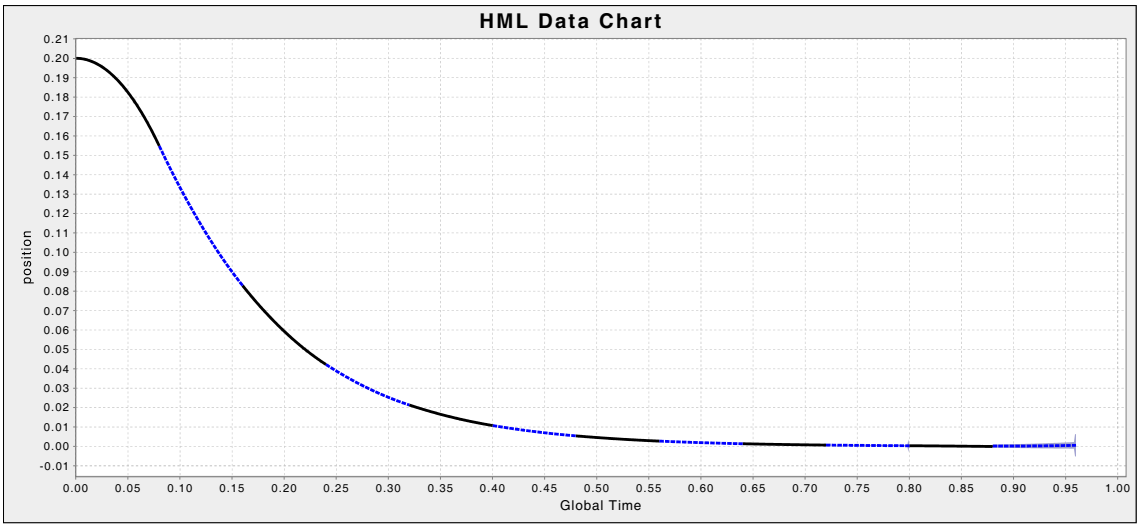


图 5.10: 离散控制下倒立摆的位置。采样时间 0.08 s，仿真时间 0.96 s。实曲线和虚曲线用于区分不同的展开深度。例如，第一段实曲线代表深度为 0 的展开所对应的倒立摆的行为，第一段虚曲线对应的深度则为 1

通过调用 dReal 进行验证，其返回的结果是 SAT（δ-SAT）。因此，上述谓词公式代表的状态在我们的离散控制模型中是可达的，说明倒立摆的位置和速度可以被控制于范围 $-0.01 \sim +0.01$ 之内。图5.10展示的是模型仿真结果，可以看出，在 0.6 s 之后，倒立摆的位置一直控制在位置 0 附近且较稳定。

2. 基于 SMT 求解器验证的特点，上述性质的可满足并不代表倒立摆的位置一定会处于所期望的范围内的，因此，我们需要判断倒立摆的位置和速度是否会超出期望的范围（在模型进行深度 11 的展开后）。考虑如下性质：

$$(position > 0.01 \vee position < -0.01) \vee (velocity > 0.01 \vee velocity < -0.01)$$

如果上述谓词公式可满足（即对应的状态可达）则表示倒立摆的位置或者速度会超出期望的数值范围。经过验证，dReal 返回的结果是 UNSAT（即不可满足的），因此，我们可以确认在离散控制下，倒立摆的位置和速度将稳定于范围 $[-0.01, 0.01]$ 内。从图5.10也可以看出，在 0.6 s 之后，倒立摆将趋于稳定。我们将模型展开的深度设为 8，然后继续对上述公式进行检验，dReal 的可达性分析结果同样显示为 UNSAT。

总之，第一个性质分析表示倒立摆是可以稳定的，而第二个性质分析则表示倒立摆是一定可以趋于稳定的。在本案例分析中，我们所用的实验设备是一台普通的 PC 机，其中，处理器：3.2GHz Quad-Core Intel Core i5-4460，内存：24GB RAM，操作系统：64-bit Ubuntu 14.10 (Utopic)。

5.5　本章小结

本章主要通过扩展 HML 语言，使之提供代码重用的功能并且更易于在实际应用中进行建模。同时，我们还将 HML 语言转换到 SMT 公式，并整合 SMT 求解器 dReal 的功能，开发了相应的原型工具以支持模型的仿真和可达性分析。通过倒立摆模型的分析，一方面检查了我们提出的工具的有效性，另一方面也可说明，我们的建模语言和工具可以对工业界中的实际系统进行建模和仿真，同时还可支持非线性混成系统的形式化分析。

第六章　仿真与验证原型工具实现

6.1　工具框架

基于混成关系理论，我们将 HML 语言建立的混成系统模型转换为 SMT 公式，在 SMT 求解器 dReal 的基础上，实现模型的仿真和形式化验证。图6.1 展示的是原型工具的基本框架。

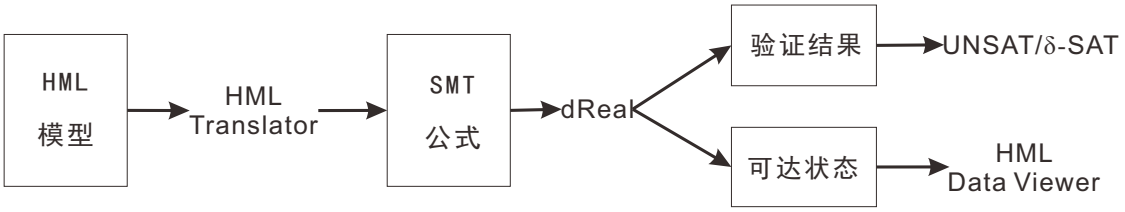


图 6.1: HML 原型工具基本框架

在图6.1中，工具的输入为使用 HML 语言建立的混成系统模型，输出为 dReal 的验证结果，以及系统行为的仿真图形。从 HML 模型到 SMT 公式的转换是通过我们开发的工具中的转换器（HML Translator）自动实现的。当 HML 模型转换为 SMT 公式后，我们调用 dReal 进行验证，dReal 会返回验证结果：UNSAT 或者 δ-SAT，并生成可达状态的数据文件（格式为 JSON[1]），通过分析可达状态数据，我们开发了数据的可视化图形展示工具（HML Data Viewer）。

使用 HML Data Viewer，可以很容易地让系统设计和开发人员了解系统变量随时间演变的方式，从而对系统的行为有较直观的认识，可以作为形式化验证的辅助分析手段。另外，通过在 SMT 公式中附加性质公式，我们可以验证相关

[1] http://www.json.org/

性质的可满足性和状态的可达性。

本章余下部分内容将涉及建模语言分析技术的介绍、工具的中间结果（SMT公式）的表达以及案例分析等方面。

6.2 语言分析技术

ANTLR（ANother Tool for Language Recognition）[113] 是一个基于 $LL(*)$[2] [5, 59] 的语言识别器。ANTLR 通过解析用户定义的文法，可以自动生成词法分析器（lexer）、语法分析器（parser ）以及树分析器（tree parser）。从 1988 年起，ANTLR 是在旧金山大学（University of San Francisco）Terence Parr 教授的领导之下逐步完成的。目前，很多大公司都基于 ANTLR 来开发自己的产品，如：Twitter、Google、Oracle、IBM 以及 Yahoo 等等。

ANTLR 使用上下文无关文法（context-free grammar）来描述语言，最新的 ANTLR（v4）是一个基于自上而下的分析器 $ALL(*)$ 的语言识别器，称为自适应 $LL(*)$。$ALL(*)$ 是对 ANTLR v3 中使用的 $LL(*)$ 的扩展，$LL(*)$ 实现的是语法的静态分析，而 $ALL(*)$ 可以实现动态语法分析。此外，ANTLR v4 在文法上也较 v3 版简单。此外，ANTLR v4 还支持左递归的自动重写，能够自动生成语法树，以及语法树的遍历器等等 [112]。

图6.2显示的是语言识别器的基本数据流图。对于语言识别器来说，其最开始的输入不管整个程序代码还是一条程序语句，均被当成字符串来处理。图中，输入为一个赋值语句，通过词法分析后拆分为词法单元，然后输入给语法分析器，最后生成语法树。

整个过程在 ANTLR 中是完全自动的。语法树生成之后，我们就可以通过树的遍历操作实现复杂的应用需求，比如，我们可以计算某个表达式的取值或者将程序转换成中间代码或其他语言的程序等等。我们在对混成系统建模语言 HML 进行分析时，就是基于 ANTLR 自动生成的语法树及其遍历器机制（Listener 和

[2]两个 “L" 分别表示自左而右扫描和最左推导，“*’’ 表示扫描的字符（token）不限定特定数目

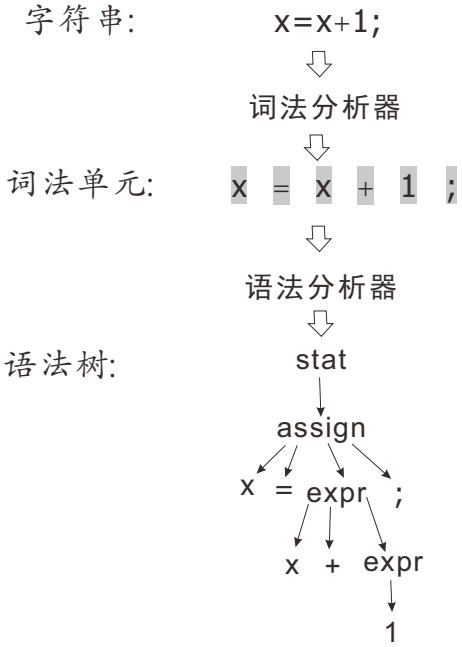


图 6.2: 语言识别器的基本数据流图示例

Visitor）实现 HML 程序到 SMT 公式的转换的。下面主要介绍 ANTLR 中对于语法树遍历所采用的 Listener 和 Visitor 机制以及混成系统建模语言 HML 的文法。

ANTLR Listener

ANTLR 默认会生成一个语法树监听接口，可以对 ANTLR 内置的树遍历器所触发的事件进行反应。当 ANTLR 运行时依据语言的文法规则进行解析时，遍历器会产生进入（enter）和退出（exit）事件，并附加相应的规则上下文信息。通过实现监听接口，我们可以在规则的进入或退出事件的回调函数中实现具体的操作。所以，Listener 机制是一种被动的遍历语法树的方式，在具体应用的实现时，我们不用自己去实现语法树的遍历算法，而只需在事件发生时对具体应用中所关心的语法元素做出合理的操作即可。

ANTLR Visitor

Visitor 机制与 Listener 机制的被动模式是不同的。Visitor 机制采用的是主动遍历的方式对语法树进行遍历。通过扩展 ANTLR 自动生成的 Visitor 类，可以在遍历语法树时根据需要选定语法元素进行访问，而对于语法树中其他的语法

元素，则使用 ANTLR 默认的遍历和访问方法。与 Listener 机制相比，Visitor 可以不受 ANTLR 的默认实现的影响，可以更加灵活方便地实现具体应用的需求。当应用不需要访问语法树中所有节点或者访问依赖于上下文环境，则 Visitor 机制将是我们首选的方案。

HML 语言文法定义

首先，我们需要定义文法的名称。对于 HML 语言，我们直接使用 HML 作为文法名称，具体的定义代码如下所示：

```
grammar HML;
```

其中，grammar 为 ANTLR 文法语言中的关键字。

HML 模型有变量定义、约束条件、模板声明和主程序等组成部分，相应的文法定义为：

```
1  hybridModel
2      : signalDeclaration*
3        variableDeclaration*
4        variableConstraint*
5        template*
6        program
7        EOF
8      ;
```

其中，冒号（:）之前的字符串表示规则的名称。ANTLR 规定，以小写字母开头的规则名称为语法规则，以大写字母开头的规则名称是词法规则。EOF 是 ANTLR 预定义的字符表示文件结束，可以表示在模型的解析时需要对整个文件内容（直到文件结束）进行处理。第 2 ∼ 6 行分别为信号声明、变量声明、变量约束、模板结构和主程序的规则名称，其中 ∗ 表示 0 次或多次匹配。

下面，我们主要列出与主程序密切相关的规则，其中主程序的定义如下：

```
program : 'Main' '{' blockStatement '}';
```

其中，以符号 ' 标识出的字符串是需要在识别时完全匹配的，而不是对其他规则的引用。程序主体部分引用了规则 blockStatement，表示需要对程序语句进行匹配。规则 blockStatement 的部分定义如下：

```
1 blockStatement
2   : atom                                                    #AtomPro
3   | '{' blockStatement ('||' blockStatement)+ '}'           #ParaCom
4   | blockStatement ';'?  blockStatement                     #SeqCom
5   | '(' blockStatement '<' expr '>' blockStatement ')'      #ConChoice
6   ... ...
7   ;
```

其中，# 用于指示 ANTLR 在生成监听器时为每一个可选的匹配子规则生成独立的进入和退出方法。比如，对于第二行的规则 atom，ANTLR 会生成 enterAtomPro 和 exitAtomPro 方法。由于 HML 语言的完整文法代码量较多，我们在此不再做详细介绍。

6.3　原型工具的中间结果

我们的原型工具的输入为 HML 语言建立的模型，这在前面已经做了介绍。现在，我们主要介绍 HML 原型验证工具的中间结果，也就是 HML 模型转换之后的 SMT 公式 [18]，这也是 SMT 求解器 dReal 的输入模型。

HML 模型转换之后的 SMT 公式会存储于文本文件中，该文件包含变量的声明、约束以及微分方程定义等逻辑公式。下面，我们依次对 HML 原型验证工具所涉及的主要 SMT 公式进行说明：

逻辑设置

首先需要定义的是求解器所用的逻辑。SMT 求解器 dReal 所用的逻辑为 QF_NRA_ODE，这是求解器内部实现的特殊逻辑。

```
(set-logic QF_NRA_ODE)
```

其中，set-logic 为 SMT 标准语言定义的关键词。逻辑 QF_NRA_ODE 是 SMT 求解器 dReal 内置的可支持非线性微分方程约束求解的逻辑。

变量声明

对变量进行声明是 SMT 公式的基本功能。如下所示公式，分别对变量 p 和 v 进行声明，并制定了变量的类型为实数类型 Real。

```
(declare-fun p () Real)
(declare-fun v () Real)
```

同时，对于每个变量，还需要定义每个深度展开时所用到的变量，用于表示模型在该深度展开时所对应的系统行为在执行前与开始执行后的取值情况。

```
(declare-fun p_0_0 () Real)
(declare-fun p_0_t () Real)
(declare-fun v_0_0 () Real)
(declare-fun v_0_t () Real)
```

其中，变量 p_0_0 用于表示在深度为 0 时，系统行为在执行前的取值，变量 p_0_t 用于表示在深度为 0 时，系统行为在开始执行后的取值。若模型展开的最大深度是 d，则形如 a_b_c 的变量名中，b $\in \{0, ..., d\}$，c 取 0 或字符 t。

变量约束

变量约束用于指示变量的取值范围，可以看出公式中使用的前缀的形式表示约束的，即运算符（<=）在前，操作数在后。

```
(assert (<= -10  p_0_0))
(assert (<= p_0_0 10))
```

模态变量

模态变量以整数的方式用于区分系统中不同的连续行为：

```
(declare-fun mode_0 () Int)
```

相应的变量取值范围与系统中含有的连续行为的种类有关，从下面的约束可以知道，该模型只有一种连续行为：

```
(assert (<= 1  mode_0))
(assert (<= mode_0  1))
```

微分方程

在我们的模型中，连续行为均可转换为微分方程的形式进行表达，在 SMT 公式中，微分方程可以表示如下：

```
(define-ode flow_1 ((= d/dt[p] v)(= d/dt[v] 0.5)))
```

其中，flow_1 为该方程的标识符或可称为名称，数字 1 表示方程的序号，从 1 开始递增。不同的连续行为所对应的方程使用不同的序号以示区分。上述微分方程定义公式表示如下微分方程组：$\frac{dp}{dt} = v$，$\frac{dv}{dt} = 0.5$。

同时，我们在模型转换时，还需要指出微分方程的初值 [p_0_0 v_0_0]，以及积分时间区间（起始时间 0，终止时间 time_0），以及表示微分方程演化过程中的变量取值（[p_0_t v_0_t]）：

```
(= [p_0_t v_0_t] (integral 0. time_0 [p_0_0 v_0_0] flow_1))
```

此外，还可以约定连续行为的不变式约束。如下公式表示在在连续行为演化过程中，变量 的取值始终小于或等于 0：

```
(forall_t 1 [0 time_0] (<= v_0_t 0))
```

其中，数字 1 表示该不变式约束所对应的微分方程的序号（flow_1 中的数字 1）。

6.4　混成系统案例展示

这一节，我们主要介绍反弹球（Bouncing ball）和水箱水位控制的 HML 模型、模型转换后的 SMT 公式以及通过原型工具分析之后的结果。

反弹球案例

如图6.3所示，反弹球初始处于一定高度（h），由于受地球引力作用，会逐渐下落，其速度（v）也会逐渐变大，当球与地面接触时，球体会因弹力而反弹，反弹后的速度会比刚刚接触地面时的速度要小，我们使用参数 K 来计算反弹后

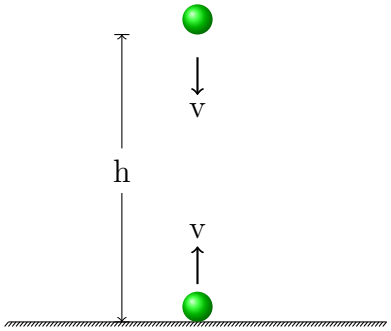


图 6.3: 反弹球示意图

```
final  float D = 0.45, K = 0.9, g = 9.8;
float height = 10, velocity = 0, clock = 0, global = 0;

height in [0, 15];
velocity in [-18, 18];
time in [0, 20];
clock in [0, 20];
global in [0, 1000];

Template  falling(float x, float v) {
  (dot  x = v) || (dot  v = -g + (-D * v )) until  (x<=0)
}
Template  bouncing(float x, float v) {
  (dot  x = v) || (dot  v = -g + (-D * v)) until  (v<=0)
}
Main  {
    while (true) {
        falling(height, velocity);
        velocity = -K * velocity;
        bouncing(height, velocity)
    }
}
```

图 6.4: 反弹球的 HML 模型

的速度值。此外，球体在下落和反弹过程中会受空气阻力影响，相应的，我们有参数 D 来表示空气阻力对球的运动速度的作用。

球体的高度和速度变化遵循下列方程：$\frac{dh}{dt} = v$，$\frac{dv}{dt} = -g + (-D * v)$。

另外，当球体反弹时，速度的变化为 $v' = -K * v$，其中 $K \in (0, 1)$，所以，

```
1 (set-logic QF_NRA_ODE)
2 (declare-fun height () Real)
3 ...
4 (define-ode flow_1 ((= d/dt[height] velocity)
5 (= d/dt[velocity] (+ (- 0 9.8) (* (- 0 0.45) velocity)))
6 (= d/dt[clock] 1)(= d/dt[global] 1)))
7 (assert (<= 0  height_0_0))
8 ...
9 (assert (and
10 (= mode_0 1) ... (= [velocity_0_t height_0_t
11 clock_0_t global_0_t](integral 0. time_0 [velocity_0_0 height_0_0
12 clock_0_0 global_0_0]  flow_1)) ... (= mode_0 1)
13
14 (= mode_1 1)(= global_1_0 global_0_t)(= velocity_1_0
15 (* (- 0 0.9) velocity_0_t)) ... (forall_t 1 [0 time_1] (not
16 (<= velocity_1_t 0)))(= mode_1 1)
17 ...
18 (= mode_10 1)(= global_10_0 global_9_t)(= velocity_10_0
19 velocity_9_t)(= clock_10_0 0) ... (forall_t 1 [0 time_10]
20 (not (<= height_10_t 0)))(= mode_10 1)))
21 (check-sat)
22 (exit)
```

图 6.5: 反弹球 HML 模型转换后的 SMT 公式

新的速度方向与原来的相反，速度的值会变小。反弹球的 HML 模型，如图6.4所示。在该模型中，球体的高度和速度分别用变量 `height` 和 `velocity` 表示。模板 `falling` 和 `bouncing` 分别表示球下落和从地面反弹上升的连续行为。在模板 `falling` 中的连续行为终止条件 (`x <= 0`) 表示，当球到达地面时，下落过程结束，模板 `bouncing` 中连续行为的终止条件 (`v <= 0`) 表示，当球的速度降为 0 时，球无法再上升。第 19 行的赋值语句用于更新球反弹后的速度。

图6.5显示的是反弹球模型以深度 10 进行展开转换为 SMT 公式。由于转换后的公式数量庞大，我们没有将所有的公式列出来，在图6.5 中，省略的公式用 '...' 代替。需要指出的是，我们在 SMT 公式中，只声明了一个微分方程公式 `flow_1`，因为在 HML 模型中，`falling` 和 `bouncing` 的区别只在终止条件上，而方程本身的约束是相同的，我们在 HML 模型转换时将约束相同的微分方程统一为一个微分方程公式。

通过 dReal 的分析，我们可以得到球体高度和速度随时间变化的状态数据。通过 HML Data Viewer 可以对状态数据进行分析并以图形方式展示。

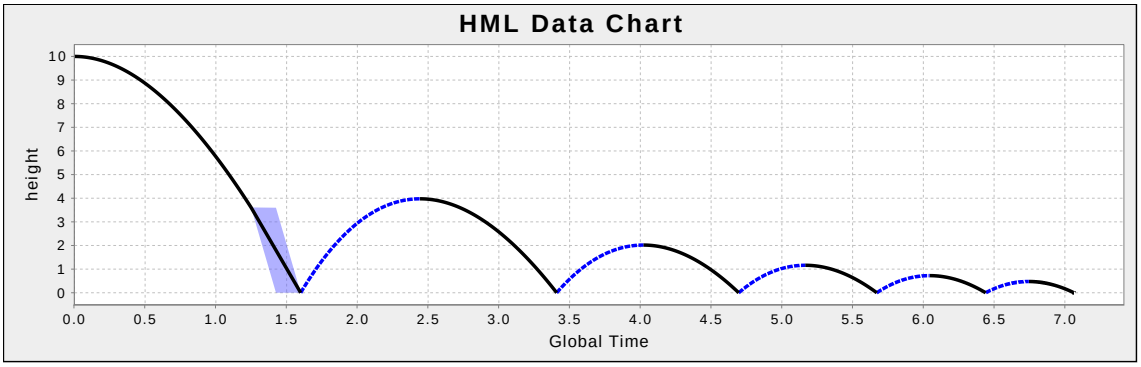


图 6.6: 球的高度随时间变化情况

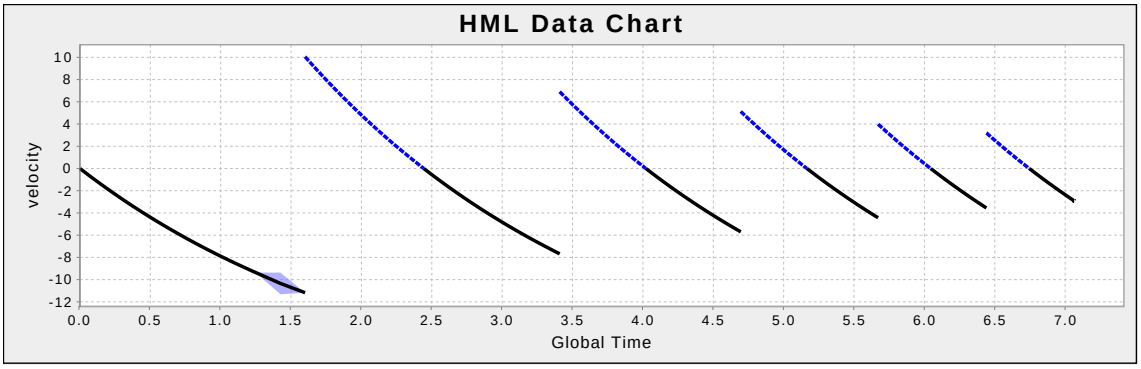


图 6.7: 球的速度随时间变化情况

图6.6和6.7展示的分别是球体高度和速度随时间变化的情况。从总体趋势上可以看出，高度和速度的数值随时间逐渐变小。这表明，球体在运动过程中，势能和动能在逐渐减少。此外，在 HML Data Viewer 对状态数据进行展示时，我们使用实线和虚线对不同深度的展开进行区分。这样，通过数据图，我们可以在图中看出模态之间的切换，实线与虚线的交接点（或间断点）即表示在该点处有离散迁移发生。

水箱水位控制案例

本案例中，水箱水位控制系统涉及两种水位变化。当水箱控制阀打开时，水箱处于进水模式，水位逐渐上升，当控制阀关闭时，水箱处于出水模式，水位将下降。因此，在水箱模型中，水位是我们所关注的最主要的连续变量。令变量 h 代表水位，以 a 代表进水时水位上升的速度，b 为水位下降的速度，则水位的连续变化遵循如下微分方程：

```
Signal  on;
Signal  off;
final  float deta = 0.5, accin = 1, decout = 1;
final  int  MaxwaterLevel = 10, LowestwaterLevel = 5;
float waterLevel = 8, clock = 0, global = 0;

waterLevel in [5, 10];
time in [0, 100];
clock in [0, 100];
global in [0, 200];

Template  open(float h, float a, int  H, float e){
    (dot  h = a) until  (h >= H-e)
}
Template  closed(float h, float b, int  L, float e) {
    (dot  h = -b) until  (h <= L+e)
}
Main  {
    while (true){
        !on;
        open(waterLevel,accin,MaxwaterLevel,deta);
        !off;
        closed(waterLevel,decout,LowestwaterLevel,deta)
    }
}
```

图 6.8: 水箱水位控制 HML 模型

1. 水位上升：$\frac{dh}{dt} = a$

2. 水位下降：$\frac{dh}{dt} = -b$

另外，我们设定一个阈值 e，当水位上升与最高水位相差 e 时，控制阀会关闭，系统会切换到出水模式。当水位下降到离最低水位相差 e 时，控制阀会打开，系统会切换到进水模式。

图6.8展示的是水箱水位控制的 HML 模型。变量 `waterLevel` 代表水位，`accin` 和 `decout` 分别表示进水和出水速度，`MaxwaterLevel` 和 `LowestwaterLevel` 分别表示最高和最低水位，`deta` 代表阈值。

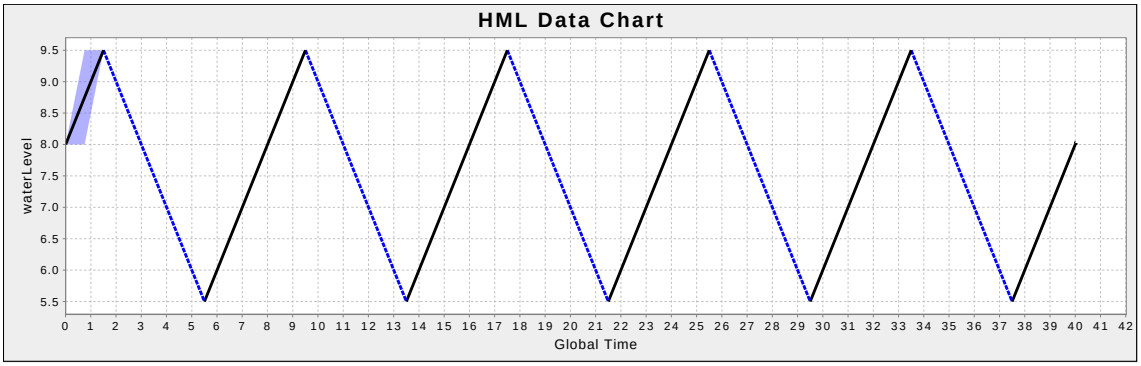


图 6.9: 水箱水位随时间变化情况

使用原型工具，我们将 HML 模型进行深度为 10 的展开，水位变化情况如图6.9所示。如果要验证系统某个性质是否满足，则可以在转换后的 SMT 公式中附加 SMT 公式来表示性质，然后使用 dReal 进行验证。例如，要检查水位是否会高于 9.6，则在深度为 10 的展开公式中可以加入公式：`(>= waterLevel_10_t 9.6)`。经检验，该公式是不可满足的。对于图6.9，我们验证的性质是 `(>= waterLevel_10_t 8.0)`，显然，该性质是可满足的。

6.5　本章小结

本章主要介绍了适用于混成系统建模语言 HML 仿真与验证的原型工具，具体内容包括工具的基本框架、语言分析技术的介绍以及 SMT 公式的说明和简单的案例分析。通过案例分析，可以认识到该原型工具目前的功能主要包括系统行为的仿真以及性质的验证。此外，目前该工具可以对一定深度的模型展开进行分析，但是，由于 dReal 的约束求解往往在模型较大时无法在可接受的时间内给出验证结果，所以未来还需在工具的效率上面做更多的优化工作。

第七章　总结与展望

7.1　总结

本论文主要研究了混成系统的建模语言、仿真技术和形式化验证理论和实现。本文提出基于仿真和验证相结合的协同验证方法，以适应规模较大的混成系统的开发和验证。在协同验证方法中，通过模型仿真结果的反馈，可对形式化模型进行指导性优化，另外，基于验证技术在模型检验方面的优势，可以对模型的正确性和安全性提供保障。此外，在混成系统建模语言方面，我们扩展了华东师范大学何积丰院士最新提出的混成系统建模语言，该语言在语法结构上具有设计良好、语法简单等特性。我们在该语言中引入类型和模板结构声明，使之能够支持模型代码重用，为大规模混成系统的建模提供基本的设计和表达能力。同时，为了支持该语言的形式化验证，我们提出基于混成关系理论的霍尔逻辑推理规则，并将该语言建立的混成系统模型自动转换为 SMT 公式进行约束求解，基于 SMT 求解器 dReal 实现了仿真和验证的自动化。

在方法学方面，我们提出仿真与验证互反馈实现混成系统的渐进开发方法。仿真支持大规模复杂系统的分析，但是无法完全覆盖系统所有可能的状态和执行路径，所以容易产生遗漏和失误。而形式化验证技术可以对系统的所有可能的状态进行分析和检查，从而能够保证系统的正确性和安全性。但是，形式化验证的缺点是无法处理大规模复杂系统的验证问题。本论文提出的反馈演进方法充分利用仿真与验证的优势，通过互反馈的方式能够逐步实现大规模复杂混成系统的可信设计与开发。

在理论方面，本论文将混成关系理论与霍尔逻辑相结合，提出新的霍尔逻辑推理规则，可以支持混成系统建模语言 HML 的推理验证。本论文提出的推理系统，是一个形式化分析验证的基本框架，对于微分方程，在推理时可以按照具体需求选择成熟的数值求解技术或者微分不变式进行分析。逻辑规则与数学方法的分离原则，极大地简化了推理规则的复杂度，并保持了推理系统的可扩展性，这样可以让系统行为推理与微分方程求解的研究工作独立的进行，降低了二者的耦合程度，增强了推理系统的灵活性。

在实践方面，本论文包含丰富的案例分析，这些案例分析在一定程度上检验了本论文提出的理论和方法，体现了理论和方法的可行性与有效性。在混成系统分析工具实现方面，本论文基于混成关系理论的基本思想，提出将基于 HML 语言建立的混成系统模型转换为 SMT 公式的方法，通过整合 SMT 求解器 dReal 的约束求解技术，开发了相应的原型工具，实现了 HML 模型仿真和验证的自动化。

7.2 展望

下一步，我们将在如下与混成系统相关的几个方面深入开展研究：

1. 在 HML 模型仿真方面，我们还需设计更好的系统执行路径展开方式，支持路径覆盖率的分析。为了避免 SMT 公式静态表达方式的缺陷，我们将提出直接基于数值求解和区间分析技术的验证方法，并实现相关的分析验证工具。

2. 在 HML 模型中引入离散动作权值规约，实现概率特性的表达，并基于动态路径展开和覆盖技术，实现概率分布求解和时态性质验证，并开发相应的统计模型检验工具。

3. 优化 HML 建模语言，增加数组和其他简单实用的数据结构，以增强模型的表达能力和可读性。同时，研究控制器代码自动生成技术，逐步实现 HML

在工业界的推广和应用。

4. 在 HML 模型中引入随机微分方程，为具有随机过程特性的混成系统提供模型和验证支撑，并应用于实际的系统开发和分析过程中。

5. 研究混成系统模型的抽象精化方法，实现不同类型的混成系统模型之间的抽象精化关系的构建与证明。该研究工作可为模型一致性分析和自动代码生成技术提供理论基础。

附录 A　共享轨道列车防碰撞系统 HML 程序代码

HML 程序 $Near$ 表示列车接近站点时的动态行为：

```
1  fcnear_i := true; l_i := ⌊p_i/250⌋; !g_{i,l_i}; l_i := l_i + 1;
2  acc_i := −0.5 ∗ v^2/(S_i − p_i);
3  while fcnear_i do
4     (v̇_i = acc_i || ṗ_i = v_i) until (p_i = 250 ∗ l_i) ⊕ (S_i = p_i) ⊕ urstop_i;
5     when((p_i = 250 ∗ l_i)&(!g_{i,l_i}; l_i := (l_i + 1) mod 56) [] ((S_i = p_i) ⊕ urstop_i)&(fcnear_i := false))
6  od;
7  when((S_i = p_i)&(frun_i := false) [] urstop_i&Urde)
```

HML 程序 $Srun$ 表示列车处于稳定行驶时的动态行为：

```
1  fcsrun_i := true; l_i := ⌊p_i/250⌋; !g_{i,l_i}; l_i := l_i + 1;
2  while fcsrun_i do
3     (v̇_i = 0 || ṗ_i = v_i) until (p_i = 250 ∗ l_i) ⊕ (S_i = p_i + 500) ⊕ urstop_i;
4     when((p_i = 250 ∗ l_i)&(!g_{i,l_i}; l_i := (l_i + 1) mod 56)
5           [](((S_i = p_i + 500) ⊕ urstop_i)&(fcsrun_i := false))
6  od;
7  when((S_i = p_i + 500)&Near [] urstop_i&Urde)
```

HML 程序 $Urde$ 用于表示列车紧急减速的动态行为:

```
1  furde_i = true; l_i := ⌊p_i/250⌋; !g_{i,l_i};
2  while furde_i do
3     (v̇_i = −1.5 || ṗ_i = v_i) until v_i = 0;
4     l_i := ⌊p_i/250⌋; !g_{i,l_i};
5     (v̇_i = 0 || ṗ_i = 0) until urstart_i;
6     when((S_i > p_i + 800)&(furde_i := false) [] (S_i ≤ p_i + 800)&Urin)
7  od;
```

HML 程序 $Urin$ 用于表示列车从紧急停车状态恢复正常行驶的行为：

```
1  fcurin_i := true; l_i := ⌊p_i/250⌋; !g_{i,l_i}; l_i := l_i + 1;
2  while fcurin_i do
3    (v̇_i = 0.5 || ṗ_i = v_i) until (p_i = 250 * l_i) ⊕ urstop_i ⊕ (p_i = 0.5 * (S_i + p_i));
4    when((p_i = 250 * l_i)&(!g_{i,l_i}; l_i := (l_i + 1) mod 56)
5          [](urstop_i ⊕ (p_i = 0.5 * (S_i + p_i)))&(fcurin_i := false))
6  od;
7  when(urstop_i&skip [] p_i = 0.5 * (S_i + p_i)&Urec)
```

HML 程序 $Urec$ 与 $Near$ 类似，主要用于紧急停车后的恢复行驶，但同时列车与前方站点距离又比较近的情况:

```
1  fcurec_i := true; l_i := ⌊p_i/250⌋; !g_{i,l_i}; l_i := l_i + 1;
2  while fcurec_i do
3    (v̇_i = −0.5 || ṗ_i = v_i) until (p_i = 250 * l_i) ⊕ (S_i = p_i) ⊕ urstop_i;
4    when((p_i = 250 * l_i)&(!g_{i,l_i}; l_i := (l_i + 1) mod 56) [] ((S_i = p_i) ⊕ urstop_i))&(fcurec_i := false))
5  od;
6  when(S_i = p_i&(furde_i := false; frun_i := false) [] urstop_i&skip)
```

参考文献

[1] J.-R. Abrial, M.K.O. Lee, D.S. Neilson, P.N. Scharbach, and I.H. Sørensen. The B-method. In Proceedings of VDM, volume 552 of Lecture Notes in Computer Science, pages 398–405. Springer-Verlag, 1991.

[2] Jean-Raymond Abrial. The B-book: Assigning programs to meanings. Cambridge University Press, Cambridge, 2005.

[3] Jean-Raymond Abrial. Modeling in Event-B: System and Software Engineering. Cambridge University Press, Cambridge, 2010.

[4] Accellera. Property specification language reference manual, version 1.1, 2004. `http://www.eda.org`.

[5] Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. Compilers: Principles, Techniques, and Tools (2nd Edition). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.

[6] Matthias Althoff, Olaf Stursberg, and Martin Buss. Computing Reachable Sets of Hybrid Systems Using A Combination of Zonotopes and Polytopes. Nonlinear Analysis: Hybrid Systems, 4(2):233–249, 2010.

[7] Rajeev Alur, Costas Courcoubetis, Thomas A. Henzinger, and Pei-Hsin Ho. Hybrid Automata: An Algorithmic Approach to the Specification and Verifi-

cation of Hybrid Systems. In Hybrid Systems, volume 736 of Lecture Notes in Computer Science, pages 209–229. Springer, 1993.

[8] Rajeev Alur, Thao Dang, Joel Esposito, Yerang Hur, Franjo Ivancic, Vijay Kumar, Insup Lee, Pradyumna Mishra, George J. Pappas, and Oleg Sokolsky. Hierarchical Modeling and Analysis of Embedded Systems. Proceedings of the IEEE, 91(1):11–28, 2003.

[9] Rajeev Alur, Radu Grosu, Yerang Hur, Vijay Kumar, and Insup Lee. Modular Specifications of Hybrid systems in CHARON. In Hybrid Systems: Computation and Control, volume 1790 of Lecture Notes in Computer Science, pages 6–19. Springer-Verlag, 2000.

[10] Rajeev Alur, Thomas A. Henzinger, and Pei-Hsin Ho. Automatic Symbolic Verification of Embedded Systems. IEEE Transactions on Software Engineering, 22(3):181–201, March 1996.

[11] Krzysztof R. Apt. Ten Years of Hoare's Logic: A Survey–Part I. ACM Transactions on Programming Languages and Systems, 3(4):431–483, October 1981.

[12] Eugene Asarin, Olivier Bournez, Thao Dang, and Oded Maler. Approximate reachability analysis of piecewise-linear dynamical systems. In Proceedings of HSCC, volume 1790 of Lecture Notes in Computer Science, pages 20–31. Springer-Verlag, 2000.

[13] Eugene Asarin, Thao Dang, and Oded Maler. d/dt: A Tool For Reachability Analysis Of Continuous and Hybrid Systems. Proceedings of NOLCOS, pages 3–34, 2001.

[14] Eugene Asarin, Thao Dang, Oded Maler, and Romain Testylier. Using Redundant Constraints for Refinement. In Proceedings of ATVA, volume 6252 of Lecture Notes in Computer Science, pages 37–51. Springer-Verlag, 2010.

[15] J. C. M. Baeten and W. P. Weijland. Process Algebra. Cambridge University Press, New York, NY, USA, 1990.

[16] Roberto Bagnara, Elisa Ricci, Enea Zaffanella, and Patricia M. Hill. Possibly Not Closed Convex Polyhedra and the Parma Polyhedra Library. In Proceedings of SAS, volume 2477 of Lecture Notes in Computer Science, pages 299–315. Springer-Verlag, 2002.

[17] Thomas Ball and SriramK. Rajamani. The SLAM Toolkit. In Proceedings of CAV, volume 2102 of Lecture Notes in Computer Science, pages 260–264. Springer Berlin Heidelberg, 2001.

[18] Clark Barrett, Aaron Stump, and Cesare Tinelli. The SMT-LIB Standard: Version 2.0. Technical report, Department of Computer Science, The University of Iowa, 2010. Available at `www.SMT-LIB.org`.

[19] Albert Benveniste, Timothy Bourke, Benoît Caillaud, and Marc Pouzet. Non-standard semantics of hybrid systems modelers. Journal of Computer and System Sciences, 78(3):877–910, 2012.

[20] Dirk Beyer, Thomas A. Henzinger, Ranjit Jhala, and Rupak Majumdar. The software model checker BLAST. International Journal on Software Tools for Technology Transfer, 9(5-6):505–525, 2007.

[21] Roderick Bloem, Alessandro Cimatti, Karin Greimel, Georg Hofferek, Robert Könighofer, Marco Roveri, Viktor Schuppan, and Richard Seeber. RATSY–

A New Requirements Analysis Tool with Synthesis. In Proceedings of CAV, pages 425–429. Springer-Verlag, 2010.

[22] Clifford F. Bonnett. Practical Railway Engineering. Imperial College Press, 2005.

[23] Oleg Botchkarev and Stavros Tripakis. Verification of Hybrid Systems with Linear Differential Inclusions Using Ellipsoidal Approximations. In Proceedings of HSCC, volume 1790 of Lecture Notes in Computer Science, pages 73–88. Springer Berlin Heidelberg, 2000.

[24] Robert S. Boyer and J Strother Moore. A Theorem Prover for A Computational Logic. In Proceedings of CADE, volume 449 of Lecture Notes in Computer Science, pages 1–15. Springer Berlin Heidelberg, 1990.

[25] Riccardo Bresciani and Andrew Butterfield. From Distributions to Probabilistic Reactive Programs. In Proceedings of ICTAC, volume 8049 of Lecture Notes in Computer Science, pages 94–111. Springer Berlin Heidelberg, 2013.

[26] Roberto Bruttomesso, Edgar Pek, Natasha Sharygina, and Aliaksei Tsitovich. The OpenSMT Solver. In Proceedings of TACAS, volume 6015 of Lecture Notes in Computer Science, pages 150–153. Springer Berlin Heidelberg, 2010.

[27] Lei Bu, Jianhua Zhao, and Xuandong Li. Path-oriented Reachability Verification of a Class of Nonlinear Hybrid Automata Using Convex Programming. In Proceedings of VMCAI, volume 5944 of Lecture Notes in Computer Science, pages 78–94, Berlin, Heidelberg, 2010. Springer-Verlag.

[28] Ana Cavalcanti, Andy Wellings, and Jim Woodcock. The Safety-Critical Java Memory Model: A Formal Account. In Proceedings of FM, volume 6664 of Lecture Notes in Computer Science, pages 246–261. Springer Berlin Heidelberg, 2011.

[29] Wei-Der Chang and Shun-Peng Shih. PID controller design of nonlinear systems using an improved particle swarm optimization approach. Communications in Nonlinear Science and Numerical Simulation, 15(11):3632–3639, 2010.

[30] Zhou Chaochen, C. A. R. Hoare, and Anders P. Ravn. A Calculus of Durations. Information Process Letter, 40(5):269–276, 1991.

[31] Zhou Chaochen, Wang Ji, and Anders P. Ravn. A Formal Description of Hybrid systems. In Hybrid Systems III, volume 1066 of Lecture Notes in Computer Science, pages 511–530. Springer-Verlag, 1996.

[32] Zhou Chaochen, Anders P. Ravn, and Michael R. Hansen. An Extended Duration Calculus for Hybrid Real-Time Systems. In Hybrid Systems, volume 736 of Lecture Notes in Computer Science, pages 36–59. Springer, 1992.

[33] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. Taylor Model Flowpipe Construction for Non-linear Hybrid Systems. Proceedings of RTSS, pages 183–192, 2012.

[34] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. Flow*: An Analyzer for Non-linear Hybrid Systems. In Proceedings of CAV, volume 8044 of Lecture Notes in Computer Science, pages 258–263. Springer Berlin Heidelberg, 2013.

[35] ClearSy. ClearSy–A French SME company, 2013. http://www.clearsy.com/?lang=en.

[36] ClearSy. COPPILOT System, 2013. http://www.coppilot.fr/en/coppilot/.

[37] ClearSy. Tools and applications at ClearSy, 2013. http://www.tools.clearsy.com.

[38] Thomas H. Cormen, Charles Eric Leiserson, and Ronald L. Rivest. Introduction to Algorithms. MIT Press, 2000.

[39] P. J. L. Cuijpers and M. A. Reniers. Hybrid Process Algebra. The Journal of Logic and Algebraic Programming, 62(2):191–245, 2005.

[40] René David. Modeling of hybrid systems using continuous and hybrid petri nets. In Proceedings of the Seventh International Workshop on Petri Nets and Performance Models, pages 47–58. IEEE, 1997.

[41] Martin Davis, George Logemann, and Donald Loveland. A Machine Program for Theorem-proving. Communications of the ACM, 5(7):394–397, July 1962.

[42] Martin D. Davis, Ron Sigal, and Elaine J. Weyuker. Computability, Complexity, and Languages: Fundamentals of Theoretical Computer Science (2nd Edition). Academic Press Professional, Inc., San Diego, CA, USA, 1994.

[43] E.W. Dijkstra and C.S. Scholten. Predicate Calculus and Program Semantics. Texts and monographs in computer science. Springer-Verlag, 1990.

[44] Laurent Doyen, Thomas A. Henzinger, and Jean-François Raskin. Automatic Rectangular Refinement of Affine Hybrid Systems. In Proceedings of FORMATS, volume 3829 of Lecture Notes in Computer Science, pages 144–161. Springer-Verlag, 2005.

[45] Alessio Ferrari, Alessandro Fantechi, Gianluca Magnani, Daniele Grasso, and Matteo Tempestini. The Metrô Rio Case Study. Science of Computer Programming, 78(7):828–842, 2013.

[46] Goran Frehse. Compositional Verification of Hybrid Systems using Simulation Relation. PhD thesis, Radboud University Nijmegen, 2005.

[47] Goran Frehse. PHAVer: Algorithmic Verification of Hybrid Systems Past HyTech. In Proceedings of HSCC, volume 3414 of Lecture Notes in Computer Science, pages 258–273. Springer-Verlag, 2005.

[48] Goran Frehse. PHAVer: Algorithmic Verification of Hybrid Systems Past HyTech. International Journal on Software Tools for Technology Transfer, 10(3):263–279, 2008.

[49] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. SpaceEx: Scalable Verification of Hybrid Systems. In Proceedings of CAV, volume 6806 of Lecture Notes in Computer Science, pages 379–395. Springer-Verlag, 2011.

[50] Sicun Gao, Jeremy Avigad, and Edmund M. Clarke. Delta-Complete Decision Procedures for SatisfiabilityăOverăthe Reals. In Automated Reasoning, volume 7364 of Lecture Notes in Computer Science, pages 286–300. Springer Berlin Heidelberg, 2012.

[51] Sicun Gao, Jeremy Avigad, and Edmund M Clarke. Delta-Decidability Over the Reals. In Proceedings of LICS, pages 305–314. IEEE, 2012.

[52] Sicun Gao, Malay Ganai, Franjo Ivančić, Aarti Gupta, Sriram Sankaranarayanan, and Edmund M. Clarke. Integrating ICP and LRA Solvers for

Deciding Nonlinear Real Arithmetic Problems. In Proceedings of FMCAD, pages 81–89. IEEE, 2010.

[53] Sicun Gao, Soonho Kong, and Edmund M. Clarke. dReal: An SMT solver for nonlinear theories over the reals. In Proceedings of CADE, volume 7898 of Lecture Notes in Computer Science, pages 208–214. Springer Berlin Heidelberg, 2013.

[54] Antoine Girard. Reachability of Uncertain Linear Systems Using Zonotopes. In Proceedings of HSCC, volume 3414 of Lecture Notes in Computer Science, pages 291–305. Springer Berlin Heidelberg, 2005.

[55] Antoine Girard and Colas Le Guernic. Zonotope/Hyperplane Intersection for Hybrid Systems Reachability Analysis. In Proceedings of HSCC, volume 4981 of Lecture Notes in Computer Science, pages 215–228. Springer Berlin Heidelberg, 2008.

[56] Michael J. C. Gordon. Introduction to the HOL system. In Proceedings of the International Workshop on the HOL Theorem Proving System and its Applications, Davis, California, USA, pages 2–3, 1991.

[57] T. Granlund and K. Ryde. The GNU Multiple Precision Arithmetic Library Version 4.0, 2001.

[58] Laurent Granvilliers and Frédéric Benhamou. Algorithm 852: RealPaver: An Interval Solver Using Constraint Satisfaction Techniques. ACM Transactions on Mathematical Software, 32(1):138–156, 2006.

[59] Dick Grune, Ceriel Jacobs, Dick Grune, and Ceriel J. Jacobs. Parsing Techniques: A Practical Guide. Springer-Verlag New York, 2008.

[60] Colas Le Guernic and Antoine Girard. Reachability Analysis of Hybrid Systems Using Support Functions. In Proceedings of CAV, volume 5643 of Lecture Notes in Computer Science, pages 540–554. Springer Berlin Heidelberg, 2009.

[61] Colas Le Guernic and Antoine Girard. Reachability Analysis of Linear Systems Using Support Functions. Nonlinear Analysis: Hybrid Systems, 4(2):250–262, 2010.

[62] N. Halbwachs, Y. Proy, and P. Raymond. Verification of Linear Hybrid Systems By Means of Convex Approximations. In Proceedings of SAS, volume 864 of Lecture Notes in Computer Science, pages 223–237. Springer-Verlag, 1994.

[63] David Harel. Statecharts: A Visual Formalism for Complex Systems. Science of Computer Programming, 8(3):231–274, 1987.

[64] David Harel and Amnon Naamad. The STATEMATE Semantics of Statecharts. ACM Transactions on Software Engineering and Methodology, 5(4):293–333, 1996.

[65] Thomas A. Henzinger. The Theory of Hybrid Automata. In Proceedings of LICS, pages 278–292. IEEE, 1996.

[66] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. HyTech: A Model Checker for Hybrid Systems. International Journal on Software Tools for Technology Transfer, 1(1-2):110–122, 1997.

[67] Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What's Decidable about Hybrid Automata? Journal of Computer and System Sciences, 57(1):94–124, 1998.

[68] Thomas A. Henzinger and Howard Wong-Toi. Using HyTech to Synthesize Control Parameters for A Steam Boiler. In Formal Methods for Industrial Applications, volume 1165 of Lecture Notes in Computer Science, pages 265–282. Springer-Verlag, 1996.

[69] C. A. R. Hoare. An Axiomatic Basis for Computer Programming. Communications of the ACM, 12(10):576–580, 1969.

[70] C. A. R. Hoare. Communicating Sequential Processes. Prentice Hall, 1985.

[71] C. A. R. Hoare and He Jifeng. Unifying Theories of Programming. Prentice Hall, 1998.

[72] Philip Holmes. Poincaré, celestial mechanics, dynamical-systems theory and "chaos". Physics Reports, 193(3):137–163, 1990.

[73] Jonathan Jacky. The Way of Z: Practical Programming with Formal Methods. Cambridge University Press, 1996.

[74] He Jifeng. From CSP to Hybrid Systems. In A Classical Mind, pages 171–189. Prentice Hall, Hertfordshire, UK, 1994.

[75] He Jifeng. A Clock-Based Framework for Construction of Hybrid Systems. In Proceedings of ICTAC, volume 8049 of Lecture Notes in Computer Science, pages 22–41. Springer Berlin Heidelberg, 2013.

[76] He Jifeng. Hybrid Relation Calculus. In Proceedings of ICECCS, page 2. IEEE, 2013.

[77] He Jifeng, Xiaoshan Li, and Zhiming Liu. rCOS: A Refinement Calculus for Object Systems. Theoretical Computer Science, 365(1-2):109–142, 2006.

[78] He Jifeng and Xu Qiwen. Advanced features of duration calculus and their applications in sequential hybrid programs. Formal Aspects of Computing, 15(1):84–99, 2003.

[79] He Jifeng, Karen Seidel, and Annabelle McIver. Probabilistic Models for the Guarded Command Language. Science of Computer Programming, 28(2-3):171–192, 1997.

[80] Hyun-Jeong Jo, Jong-Gyu Hwang, and Yong-Ki Yoon. Development of Formal Method Application for Ensuring Safety in Train Control System, 2008. http://www.railway-research.org/IMG/pdf/o.3.4.2.3.pdf.

[81] C. B. Jones. Development Methods for Computer Programs including a Notion of Interference. PhD thesis, Oxford University, 1981.

[82] E.W. Kamen. Industrial Controls and Manufacturing. Elsevier Science, 1999.

[83] Ker-I Ko. Complexity Theory of Real Functions. Birkhauser Boston Inc., Cambridge, MA, USA, 1991.

[84] Daniel Kroening and Ofer Strichman. Decision Procedures: An Algorithmic Point of View. Springer Berlin Heidelberg, 2008.

[85] Alexander B. Kurzhanski and Pravin Varaiya. Ellipsoidal Techniques for Reachability Analysis. In Proceedings of HSCC, volume 1790 of Lecture Notes in Computer Science, pages 202–214. Springer-Verlag, 2000.

[86] W. Kühn. Rigorously Computed Orbits of Dynamical Systems without the Wrapping Effect. Computing, 61(1):47–67, 1998.

[87] Gérard Le Lann. An Analysis of the Ariane 5 Flight 501 Failure - A System Engineering Perspective. In Proceedings of ECBS, pages 339–346. IEEE, 1997.

[88] Thierry Lecomte. Safe and Reliable Metro Platform Screen Doors Control/-Command Systems. In Proceedings of FM, volume 5014 of Lecture Notes in Computer Science, pages 430–434. Springer-Verlag, 2008.

[89] Thierry Lecomte. Applying A Formal Method in Industry: A 15-Year Trajectory. In Proceedings of FMICS, volume 5825 of Lecture Notes in Computer Science, pages 26–34. Springer-Verlag, 2009.

[90] Edward A. Lee. Cyber Physical Systems: Design Challenges. In Proceedings of ISORC, pages 363–369, 2008.

[91] Bing Liu, Soonho Kong, Sicun Gao, Paolo Zuliani, and Edmund Clarke. Towards Personalized Cancer Therapy Using Delta-Reachability Analysis. In Proceedings of HSCC, pages 227–232. ACM, 2015.

[92] Bing Liu, Soonho Kong, Sicun Gao, Paolo Zuliani, and EdmundM. Clarke. Parameter Synthesis for Cardiac Cell Hybrid Models Using Delta-Decisions. In Computational Methods in Systems Biology, volume 8859 of Lecture Notes in Computer Science, pages 99–113. Springer International Publishing, 2014.

[93] Nancy Lynch, Roberto Segala, and Frits Vaandrager. Hybrid I/O Automata. Information and Computation, 185(1):105–157, 2003.

[94] Nancy A. Lynch and Frits W. Vaandrager. Forward and Backward Simulations, II: Timing-Based Systems. Information and Computation, 128(1):1–25, 1996.

[95] Oded Maler, Zohar Manna, and Amir Pnueli. From Timed to Hybrid Systems. Lecture Notes in Computer Science, 600(12):447–484, 1992.

[96] S. Marrone, R. Nardone, A. Orazzo, I. Petrone, and L. Velardi. Improving Verification Process in Driverless Metro Systems: The MBAT Project. In Proceedings of ISoLA, volume 7610 of Lecture Notes in Computer Science, pages 231–245. Springer-Verlag, 2012.

[97] MathWorks. Simulink, 2013. http://www.mathworks.com/products/simulink/.

[98] MathWorks. Stateflow, 2013. http://www.mathworks.com/products/stateflow/.

[99] MBAT Consortium. ARTEMIS Project MBAT, 2013. http://www.mbat-artemis.eu.

[100] Modelica Association. Modelica, 2013. https://modelica.org.

[101] Pieter J. Mosterman. Hybrid Dynamic Systems: A Hybrid Bond Graph Modeling Paradigm and its Application in Diagnosis. PhD thesis, Vanderbilt University, 1997.

[102] National Institute of Standards and Technology (NIST). Fire Dynamics Simulator and Smokeview Code. http://code.google.com/p/fds-smv/.

[103] Greg Nelson and Derek C. Oppen. Simplification by Cooperating Decision Procedures. ACM Transactions on Programming Languages and Systems, 1(2):245–257, 1979.

[104] Greg Nelson and Derek C. Oppen. Fast Decision Procedures Based on Congruence Closure. Journal of the ACM, 27(2):356–364, 1980.

[105] X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. An Approach to the Description and Analysis of Hybrid Systems. In Proceedings of Hybrid Systems, volume 736 of Lecture Notes in Computer Science, pages 149–178. Springer-Verlag, 1993.

[106] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving SAT and SAT Modulo Theories: From an Abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T). Journal of the ACM, 53(6):937–977, 2006.

[107] R. Nikoukhah. Hybrid Dynamics in Modelica: Should all Events be Considered Synchronous? In First International Workshop on Equation-Based Object Oriented Languages and Tools (EOOLT 2007), Berlin, Germany, pages 37–48. Berlin, Germany, 2007.

[108] Tobias Nipkow, Markus Wenzel, and Lawrence C. Paulson. Isabelle/HOL: A Proof Assistant for Higher-order Logic. Springer-Verlag, Berlin, Heidelberg, 2002.

[109] Marcel Oliveira, Ana Cavalcanti, and Jim Woodcock. A UTP Semantics for Circus. Formal Aspects of Computing, 21(1-2):3–32, 2009.

[110] Marcel Oliveira, Ana Cavalcanti, and Jim Woodcock. Unifying theories in ProofPower-Z. Formal Aspects of Computing, 25(1):133–158, 2013.

[111] S. Owre, J. M. Rushby, and N. Shankar. PVS: A Prototype Verification System. In Proceedings of CADE, volume 607 of Lecture Notes in Artificial Intelligence, pages 748–752, Saratoga, NY, June 1992. Springer-Verlag.

[112] Terence Parr. The Definitive ANTLR 4 Reference. Pragmatic Bookshelf, 2nd edition, 2013.

[113] Terence John Parr and Russell W. Quong. ANTLR: A Predicated-LL(k) Parser Generator. Software: Practice and Experience, 25(7):789–810, 1995.

[114] Henry A. Paynter. Analysis and Design of Engineering Systems. The MIT Press, 1960.

[115] Phi A. Phan and Timothy J. Gale. Direct Adaptive Fuzzy Control With a Self-Structuring Algorithm. Fuzzy Sets and Systems, 159(8):871–899, 2008.

[116] André Platzer and Jan-David Quesel. KeYmaera: A Hybrid Theorem Prover for Hybrid Systems (System Description). In Proceedings of Automated Reasoning, volume 5195 of Lecture Notes in Computer Science, pages 171–178, 2008.

[117] André Platzer. Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics. Springer-Verlag, 2010.

[118] H. Poincaré. New methods of celestial mechanics. Vol. II: Methods of Newcomb, Gyldén, Lindstedt, and Bohlin. Translated from the French. National Aeronautics and Space Administration, Washington, D.C., 1967.

[119] Roland Priemer. Introductory Signal Processing. World Scientific, 1991.

[120] L. Qu and W.K. Chow. Platform Screen Doors on Emergency Evacuation in Underground Railway Stations. Tunnelling and Underground Space Technology, 30(4):1–9, 2012.

[121] Karl Johan Åström and Tore Hägglund. Advanced PID Control. ISA-The Instrumentation, Systems, and Automation Society, Research Triangle Park, NC 27709, 2006.

[122] John C. Reynolds. Separation Logic: A Logic for Shared Mutable Data Structures. In Proceedings of LICS, pages 55–74. IEEE, 2002.

[123] Jae Seong Roh, Hong Sun Ryou, Won Hee Park, and Yong Jun Jang. CFD Simulation and Assessment of Life Safety in a Subway Train Fire. Tunnelling and Underground Space Technology, 24(4):447–453, 2009.

[124] Mauno Rönkkö, Anders P. Ravn, and Kaisa Sere. Hybrid Action Systems. Theoretical Computer Science, 290(1):937–973, 2003.

[125] Ricardo Sanfelice, David Copp, and Pablo Nanez. A Toolbox for Simulation of Hybrid Systems in Matlab/Simulink: Hybrid Equations (HyEQ) Toolbox. In Proceedings of HSCC, pages 101–106, New York, NY, USA, 2013. ACM.

[126] Roberto Sebastiani. Lazy Satisfiability Modulo Theories. Journal on Satisfiability, Boolean Modeling and Computation, 3(3–4):141–224, 2007.

[127] Sanjit A. Seshia. Adaptive Eager Boolean Encoding for Arithmetic Reasoning in Verification. PhD thesis, Carnegie Mellon University, May 2005.

[128] Adnan Sherif, Ana Cavalcanti, He Jifeng, and Augusto Sampaio. A Process Algebraic Framework for Specification and Validation of Real-Time Systems. Formal Aspects of Computing, 22(2):153–191, 2010.

[129] Robert E. Shostak. An Algorithm for Reasoning About Equality. Communications of the ACM, 21(7):583–585, 1978.

[130] Robert E. Shostak. A Practical Decision Procedure for Arithmetic with Function Symbols. Journal of the ACM, 26(2):351–360, 1979.

[131] Robert E. Shostak. Deciding Combinations of Theories. Journal of the ACM, 31(1):1–12, January 1984.

[132] Wen Su, J.-R. Abrial, and Huibiao Zhu. Complementary Methodologies for Developing Hybrid Systems with Event-B. In Proceedings of ICFEM, volume

7635 of Lecture Notes in Computer Science, pages 230–248. Springer-Verlag, 2012.

[133] Alfred Tarski. On the calculus of relations. J. Symbolic Logic, 6(3):73–89, 1941.

[134] Alfred Tarski. A Lattice-Theoretical Fixpoint Theorem and Its Applications. Pacific Journal of Mathematics, 5(2):285–309, 1955.

[135] Alfred Tarski. A Decision Method for Elementary Algebra and Geometry. In Quantifier Elimination and Cylindrical Algebraic Decomposition, Texts and Monographs in Symbolic Computation, pages 24–84. Springer Vienna, 1998.

[136] Viktor Vafeiadis and Matthew Parkinson. A Marriage of Rely/Guarantee and Separation Logic. In Proceedings of CONCUR, volume 4703 of Lecture Notes in Computer Science, pages 256–271. Springer Berlin Heidelberg, 2007.

[137] Klaus Weihrauch. Computable Analysis: An Introduction. Springer, 2000.

[138] Jim Woodcock and Ana Cavalcanti. A Concurrent Language for Refinement. In 5th Irish Workshop on Formal Methods, pages 182–196, July 2001.

[139] Jim Woodcock, Peter Gorm Larsen, Juan Bicarregui, and John S. Fitzgerald. Formal methods: Practice and experience. ACM Computing Surveys, 41(4):1–36, 2009.

[140] J Yi and N Yubazaki. Stabilization Fuzzy Control of Inverted Pendulum Systems. Artificial Intelligence in Engineering, 14(2):153–163, 2000.

[141] Lin Zhao, Tao Tang, Ruijun Cheng, and Liyun He. Property Based Requirements Analysis for Train Control System. Journal of Computer Information Systems, 9(3):915–922, 2013.

[142] Huibiao Zhu, Jifeng He, and Jonathan P. Bowen. From Algebraic Semantics to Denotational Semantics for Verilog. Innovations in Systems and Software Engineering, 4(4):341–360, 2008.

[143] Huibiao Zhu, Shengchao Qin, Jifeng He, and Jonathan P. Bowen. PTSC: Probability, Time and Shared-Variable Concurrency. Innovations in Systems and Software Engineering, 5(4):271–284, 2009.

[144] Huibiao Zhu, Jeff W. Sanders, Jifeng He, and Shengchao Qin. Denotational Semantics for a Probabilistic Timed Shared-Variable Language. In Unifying Theories of Programming, volume 7681 of Lecture Notes in Computer Science, pages 224–247. Springer Berlin Heidelberg, 2013.

[145] 卜磊, 解定宝. 混成系统形式化验证. 软件学报, 25:219–233, 2013.

[146] 黎作鹏, 张天驰, 张菁. 信息物理融合系统 (CPS) 研究综述. 计算机科学 ISTIC PKU, 38:25–31, 2011.

致　　谢

不愤不启，不悱不发，在华东师范大学软件学院，前后长达五年的学习和研究历程中，我适时地得到了诸多老师、同学以及挚友的启发与帮助。在此，向所有曾经帮助过我的人们表达诚挚的谢意。

自从二零一零年进入软件学院学习，首先，我就受到郭建副教授的指导。在郭老师多方面的启发、关怀和指导之下，我逐步开始从事混成系统方面的研究工作。在学习和研究方面，每次点滴的收获和进步，都凝聚着郭老师的心血。在此，我衷心感谢郭老师给予的无私帮助。

之后，从二零一二年始，我转为朱惠彪教授的博士生，开始了硕博连读的学习和科研生涯。朱老师对科学事业永不停歇的探索精神和一丝不苟的治学态度深深地影响了我。在朱老师的悉心指导下，我不仅学到了理论知识，还学到了思考问题和创新科研的方法。在这里，我想表达对朱老师的深深感激和无限崇敬。

衷心感谢华东师范大学软件学院何积丰院士在混成系统建模语言和语义研究方面的杰出贡献，正是在何积丰院士对物联网的积极倡导之下，我才有机会接触该领域的研究工作。感谢何积丰院士在形式化方法的历次研讨会中的谆谆教诲，本文的研究在很多方面深受何积丰院士诸多思想的启发。

衷心感谢澳门大学徐启文老师为了科研事业和学术梦想，在澳门和上海之间不辞劳苦，来往奔波。感谢徐老师在混成系统建模语言和逻辑证明系统等方面的谆谆教诲和无私帮助。

感谢法国 VERIMAG 实验室助理教授 Goran Frehse 在混成系统形式化验证方面的建议和耐心的指导。Goran Frehse 是混成系统领域的专家, 我经常与他就混成系统的研究进行交流，在论文撰写、系统分析等方面也深受他的启发和帮助。荷兰埃因霍芬理工大学（TU/e）助理教授 P.J.L. Cuijpers 在混成进程代数方面也给予我莫大的帮忙，为了让我更好地理解混成进程代数，他还专门从荷兰把他的博士论文寄给我阅读。对于 Cuijpers 教授的无私帮助，在此，我表达对他的深深感激。

感谢丹麦奥尔堡大学 Kim G. Larsen 教授邀请我去奥尔堡大学交流学习。在奥尔堡学习期间，我深深地领略了大师的卓越风采，并深受启发和鼓励。感谢奥尔堡大学 Alexandre David 副教授在生活和学习上面给予的帮助。感谢奥尔堡大学 ZHANG Zhengkui 博士和 XUE Bingtian 博士在日常生活和研究方面给予的诸多建议和支持，你们无私和热情的帮助使我受益颇多。

感谢陈仪香教授、陈勇教授、王长波教授、蒲戈光教授、刘静教授、张立臣教授在专业知识上的悉心指导。感谢曾振柄教授、郁文生教授、杨争锋副教授、章玥副教授在研究过程中给予的宝贵建议和帮助。感谢吴玲颖、庞荣、叶林娟、曾万聃老师在学习和生活中给予的关心和帮助。

我还要感谢实验室所有曾经或者此刻正奋斗在科研事业上的全体同仁：李钦、赵涌鑫、史建琦、王政、苏雯、黄滟鸿、朱龙飞、吴晓峰、刘鹏、吴成成、王梦莹、吴茜、孙再亮、严海星、袁婷、唐翊婷、苏亭、叶昕、李新、费媛、王路遥、李思祺、刘艾伦、卢建宇、向霜晴、谢宛玲、徐元敏、宋迦陵等。感谢你们的一路相伴。

最后，感谢亲爱的家人及亲友的理解和鼎力支持。你们的关爱和默默付出使我能安心学业并顺利完成论文。

方 徽 星

二零一五年五月

攻读博士学位期间发表论文和科研情况

■ 已公开发表论文

1. Fang Huixing, Zhu Huibiao, He Jifeng: SMT-Based Symbolic Encoding and Formal Analysis of HML Models. MONET: Mobile Networks and Applications, Springer US, Volume 21, Issue 1, pp.35-52, Feb 2016 (期刊, SCIE, CCF-C)

2. Fang Huixing, Zhu Huibiao, Shi Jianqi: An Object-Oriented Language for Modeling of Hybrid Systems. HASE 2015: 16th IEEE International Symposium on High Assurance Systems Engineering, IEEE Computer Society, pp.1-9, 8-10 Jan 2015, Daytona Beach, Florida, USA (会议, EI)

3. Fang Huixing, Shi Jianqi, Zhu Huibiao, Guo Jian, Kim Guldstrand Larsen, Alexandre David: Formal Verification and Simulation for Platform Screen Doors and Collision Avoidance in Subway Control Systems. STTT: International Journal on Software Tools for Technology Transfer, Springer-Verlag, Volume 16, Issue 4, pp.339-361, August 2014 (期刊, EI, CCF-C)

4. Fang Huixing, Guo Jian, Zhu Huibiao, Shi Jianqi: Formal Verification and Simulation: Co-verification for Subway Control Systems. TASE 2012: 6th International Symposium on Theoretical Aspects of Software Engineering, IEEE Computer Society, pp.145-152, 4-6 July 2012, Beijing, China (会议, EI, CCF-C)

5. Fei Yuan, Zhu Huibiao, Wu Xi, Fang Huixing: Comparative Modeling and Verification of Pthreads and Dthreads. HASE 2016: 17th IEEE International Symposium on High Assurance Systems Engineering, IEEE Computer

Society, pp.132-140, 7-9 Jan, 2016, Orlando, FL, USA (会议, EI)

6. Yan Haixing, Fang Huixing, Christian Kuka, Zhu Huibiao: Verification for OAuth Using ASLan++. HASE 2015: 16th IEEE International Symposium on High Assurance Systems Engineering, IEEE Computer Society, pp.76-84, 8-10 Jan 2015, Daytona Beach, Florida, USA (会议, EI)

7. Alexandre David, Fang Huixing, Kim Guldstrand Larsen, Zhang Zhengkui: Verification and Performance Evaluation of Timed Game Strategies. FORMATS 2014: 12th International Conference on Formal Modeling and Analysis of Timed Systems, Springer-Verlag, LNCS, Volume 8711, pp.100-114, 8-10 Sep 2014, Florence, Italy (会议, EI)

8. Shi Jianqi, Zhu Longfei, Fang Huixing, Guo Jian, Zhu Huibiao, Ye Xin: xBIL - A Hardware Resource Oriented Binary Intermediate Language. ICECCS 2012: 17th IEEE International Conference on Engineering of Complex Computer Systems, IEEE Computer Society, pp.211-219, 18-20 July 2012, Paris, France (会议, EI, CCF-C)

9. Shi Jianqi, He Jifeng, Zhu Huibiao, Fang Huixing, Huang Yanhong, Zhang Xiaoxian: ORIENTAIS: Formal Verified OSEK/VDX Real-Time Operating System. ICECCS 2012: 17th IEEE International Conference on Engineering of Complex Computer Systems, IEEE Computer Society, pp.293-301, 18-20 July 2012, Paris, France (会议, EI, CCF-C)

10. Shi Jianqi, Zhu Longfei, Huang Yanhong, Guo Jian, Zhu Huibiao, Fang Huixing, Ye Xin: Binary Code Level Verification for Interrupt Safety Properties of Real-Time Operating System. TASE 2012: 6th International Symposium on Theoretical Aspects of Software Engineering, IEEE Computer Society, pp.223-226, 4-6 July 2012, Beijing, China (会议, EI, CCF-C)

■ 主持或参与的科研课题

1. 国家 973 项目（编号：2011CB302900）“物联网的基础理论与实践研究”课题 4“物联网可信软件设计理论与方法研究”，参与理论研究，2011.01-2013.08

2. 国家自然科学基金委员会 (NSFC) 与丹麦国家研究基金会 (DNRF) 共同资助合作研究项目（编号：613111085，61361136002）“信息物理融合系统的基础研究”，参与理论研究，2011.01-2013.12，2014.01-2016.12