中国科学: 信息科学 2014 年 第 44 卷 第 ? 期: 1–?
www.scichina.com info.scichina.com
《中国科学》杂志社
SCIENCE CHINA PRESS

论 文

中国高速铁路列控系统的形式化分析与验证

郭丹青[①], 吕继东[②], 王淑灵[①], 唐涛[②], 詹乃军[①*], 周达天[②], 邹亮[①]

① 中国科学院软件研究所计算机科学国家重点实验室, 北京100190
② 北京交通大学, 北京100044
*詹乃军. E-mail: znj@ios.ac.cn

收稿日期: 2014–01–19; 接受日期: 2014–05–08
国家重点基础研究发展计划(973计划) (批准号: 2014CB340700), 国家高技术研究发展计划(863 计划) (批准号: 2012AA112801)和国家自然科学基金(批准号: 91118007, 6110006, 61304185)项目资助

摘要 高速铁路列控系统的安全与否直接涉及人民的生命财产安全, 对高速铁路列控系统进行严格的形式化验证具有重要意义. 但是随着高速铁路列控系统软件以及硬件规模的不断增大, 系统的复杂性有了很大的提高, 直接对高速铁路列控系统进行形式化验证已经变得越来越困难. 另一方面, 由于图形化建模和仿真表现方式直观, 易于理解, 在工程实践中已经得到了广泛的应用. 因此, 为了更好的保证铁路系统的安全, 对系统进行仿真排除部分安全隐患显得尤为重要. 本文通过使用Simulink/Stateflow建模工具对高速铁路列控系统的行车许可, 等级升级及部分模式转换场景进行了建模. 该模型具有普适性, 通过修改参数信息, 可以对不同的等级转换以及模式转换的组合情况进行仿真. 通过使用该模型我们对10个组合情况进行了仿真, 最后发现在某些情况下可能会出现不正常停车或者等级转换失败的现象. 类似于测试, 仿真仅仅能够发现错误, 不能证明系统没有错误. 因为仿真的这种不完备性, 对仿真辅助形式验证在安全攸关系统设计中是非常必要的. 为此, 我们取其中一个不正常停车的场景进行了形式验证, 验证结果证明了在任何情况下都会不正常停车.

关键词 中国高速铁路列控系统 Simulink/Stateflow 仿真 模式转换 等级转换 形式化验证

1 引言

高速铁路具有节能, 环保, 大运量, 安全舒适等明显优势, 是交通运输体系中最具可持续性和环境友好性的运输模式, 客运高速化已成为世界潮流. 我国高速铁路建设列为拉动内需的“火车头” , 重点发展高速铁路, 将建成以北京为中心, 2小时大中城市圈为主节点, 8小时快速铁路交通为主干的高速铁路客运网络. 可以预见, 以高速铁路为核心的快速铁路交通网的建成, 将使我国经济社会发展步入高速铁路时代. 高速铁路是庞大复杂的系统工程, 集成了多学科, 多领域的高新技术. 其中中国高速铁路列控系统是保证列车高速, 高效, 安全运行的坚实基础, 所以确保高速铁路列控系统的安全性有非常重要的意义.

列车运行控制系统是高速铁路的核心技术之一, 是一个深度融合了计算, 通信和控制的系统, 同时又是安全攸关的系统, 即系统的任何错误都可能导致灾难性后果, 是一种典型的信息物理融合系统(Cyber-Physical Systems, CPS). 例如, 2011 年7月23日发生在温州事故导致40多位乘客失去生命. 它

引用格式: 郭丹青, 吕继东, 唐涛等. 中国高速铁路列控系统的形式化分析与验证. 中国科学: 信息科学, 2014, 44: 1–?

通过3C (Computation, Communication, Control) 技术的有机融合与深度协作, 实现列车运行过程与信息交互系统的实时感知, 动态控制和信息服务. 随着列车运行速度的不断提高(运行时速将超过500 公里/小时), 地面, 轨道网络与列车之间, 列车与列车之间的交互作用极其复杂, 使得列车运行控制系统建模, 性质刻画和性能抽取变得复杂, 最终必然导致对系统安全性分析和验证的复杂程度急剧增加. 列车运行安全控制标准是列车运行控制系统的技术标准, 是实现列车安全运行的技术保障之一. 因而必须确保制定的列车运行安全控制标准是安全可靠的, 即首先保证内部不存在互相矛盾的地方; 其次, 要保证根据标准列车能够安全运行在给定路段上; 再次, 一旦发生事故, 根据标准能够采取相应应对措施不至于发生灾难性后果.

1.1 本文主要贡献

本文主要研究如何利用信息物理融合系统方面最新结果, 特别是我们最近几年在混成系统建模, 分析和验证方面的结果, 提出一套中国高速铁路列控系统图形建模, 仿真, 形式建模和形式验证于一体的方法, 从而提高中国高速铁路列控系统的可靠性. 具体讲：

Simulink/Stateflow[1,2]是Matlab中一个重要的商业组件, 拥有Matlab的强大计算能力支持, 在工业界使用广泛, 拥有深厚的用户基础. 它是一种图形化的建模工具, 对模型的描述比较直观, 并且支持仿真, 可以检查模型是否与预计行为相符, 以便排除部分早期的设计失误, 已经成为一种事实上的工业标准. 同时, 它还可以将建立的模型转化为对应的C语言, 有助降低开发成本. 而过去, 很少有使用Simulink/Stateflow对列控系统进行建模和仿真的工作. 基于此, 我们首先针对目前中国高速铁路列控系统的规范, 提出一种一般性Simulink/Stateflow图形模型框架. 对高速铁路列控系统的行车许可, 等级升级及部分模式转换场景建立了Simlink/Stateflow模型, 并且使用该模型对二级到三级与模式转换相结合的部分情况进行了仿真, 发现在部分情况下会出现不正常停车以及等级转换失败的现象.

另一方面, 因为仿真仅仅能够根据环境的有穷输入在有限时间内观察系统行为是否满足要求, 而环境的输入通常有无穷多种, 并且这些系统的运行可能没有时间限制, 所以类似于测试, 仿真仅仅能够发现系统错误, 不能够证明系统没有错误. 特别对于安全攸关系统, 仅仅仿真无法保证系统的正确性. 因而对Simulink/Stateflow图形模型进行形式验证, 作为仿真的一种补充, 是非常必要的. 形式验证Simulink/Stateflow图形模型的前提是需要提供一个形式语义以及针对该形式语义的验证技术. 在[3,4]中, 我们考虑如何将Simulink/Stateflow图形模型转换成HCSP[5,6]形式模型, 并使用HHL[7]及其定理证明器[8]来形式验证转化后的形式模型. 本文的另一贡献是：我们以其中某个场景为例, 说明如何使用上述方法将其转换为形式模型并严格验证, 验证结果表明在任何输入下均不能正常停车.

1.2 相关工作

为了支持混成系统建模, 人们提出很多建模语言, 例如: Modelica[9], HybridUML[10], 时间自动机[11], 混成自动机[12], Esterel[13], 等等. Modelica[9]与Simulink同是图形化的建模语言. 它是一种开源语言, 支持用户自定义修改. 它同时支持图形建模和代码生成, 表达能力很强. 但是它没有类似于Stateflow这样的控制流图建模工具, 所以对控制逻辑建模支持比较薄弱. 虽然Modelica使用算法和函数对控制逻辑进行建模, 但是由于这种建模方式是基于文本的, 表现不够直观, 不利于对高层模型进行建模. HybridUML[10] 是一种对混成系统进行图形化建模的语言, 它是基于传统UML的混成扩展. UML在工业界有广泛的使用, 用户基础较好. 但是HybridUML不能进行仿真, 也不能进行验证, 使用上有很大的限制. 混成自动机[12]是自动机在混成系统方面的拓展, 基于可达集计算的各种验证工作在学

术界研究深入, 应用广泛. 基于自动机的建模技术直观, 易于理解, 但是它没有系统结构方面的信息, 组合性差, 不适合大型系统的建模. 特别是, 没有图形化工具支持, 不能进行仿真, 形式验证技术也不适用于复杂大型系统. Esterel[13]是一种基于时间同步模型的建模语言. 与常见的同步异步通讯方式不同, 当Esterel中信号被发送之后, 该时间点上其他并发进程都可以将该信号接收任意有限次. Esterel拥有成熟的商业工具Scade[14]对其进行支持, 该工具生成的代码可适用于实际系统, 减少了后期开发的工作量. **总之, Simulink/Stateflow是更实用的混成系统建模, 分析和仿真工具.**

国内外有许多关于高速铁路列控系统形式建模和验证的工作, 例如: Platzer等在[15]利用一种基于微分不变量的微分动态逻辑等技术对欧洲高速铁路列控系统进行建模, 分析和验证, 从欧洲高速铁路列控系统的非形式描述中发现了许多不严格的地方, 之后他们还进一步通过设计一个可以严格证明安全性质的形式模型, 以改进欧洲高速铁路列控系统的设计. 国内也有很多科研工作者尝试对中国高速铁路列控系统进行形式建模和验证, 例如: 在[16]中, 作者结合UML在业界的广泛应用及SMV模型检验工具的优点, 提出了一套列控系统规范的建模与验证方法. 而在[17]中, 作者使用Petri网对高速铁路列控系统信道模型和数据传输的时间特性进行建模和分析,为CTCS-3级列控系统规范中的相关参数设计提供了前提和基础.

综上分析, 目前还没有一套针对高速铁路列控系统的集图形建模, 仿真, 形式建模和验证于一体的方法. 本文填补了这方面的空白.

2 背景知识

在这一节中, 我们将介绍高速铁路列控系统, Simulink/Stateflow和HCSP以及HHL的一些背景知识, 因为我们关心等级转换和模式转换相结合的场景, 所以我们使用第一小节和第二小节分别介绍高速铁路列控系统等级和列车在不同等级下的不同模式, 第三小节介绍Simulink/Stateflow, 第四小节介绍HCSP和HHL.

2.1 高速铁路列控系统等级概述

中国列车控制系统CTCS (Chinese Train Control System) , 根据总体原则, 从国情, 路情实际出发, 共划分为5级[18], 其中我们关心如下两个等级:

1. CTCS-2 (C2) 级

   基于轨道传输信息的列车运行控制系统, 面向提速干线和高速新线, 采用车, 地一体化设计. 适用于各种限速区段, 地面可不设通过信号机, 机车凭车载信号行车. 地面子系统中增加列控中心, 根据列车占用情况及进路状态, 计算行车许可及静态列车速度曲线并传送给列车. 点式信息设备用于向车载设备传输定位信息, 进路参数, 线路参数, 限速和停车信息等.

2. CTCS-3 (C3) 级

   基于无线传输信息, 采用轨道电路等方式检查列车占用的列车运行控制系统. 面向提速干线, 高速新线或特殊线路, 基于无线通信的固定闭塞或虚拟自动闭塞. CTCS-3在CTCS-2的基础上有了很多改进. 主要体现在控制中心, 和信息交互方式的改变上.

2.2 CTCS-2级和CTCS-3级下模式概述

每个等级下列车都有不同的模式, 其中我们只关心C2下的完全监控模式（Full Supervision mode, FS）,目视行车模式(OnSight mode, OS), 部分监控模式(Partial Supervision mode, PS)模式, 以及C3下的FS, OS, 引导模式(Calling on mode, CO), 冒进模式(Trip mode, TR).

1. CTCS-2下模式介绍

   完全监控模式(FS) : 在完全监控模式下, 列控车载设备应能判断列车位置和停车位置,在保证列车速度满足线路固定限速, 车辆构造速度, 停车位置, 临时限速等条件的前提下, 生成目标距离连续速度控制模式曲线,并连续监控列车速度, 与模式速度比较, 自动输出紧急制动或常用制动命令, 同时, 应能通过DMI显示列车实际速度, 允许速度, 目标速度和目标距离等信息.

   目视行车模式(OS) : 车载设备显示停车信号或位置不确定时, 在停车状态下司机按压专用按钮可使车载设备转入目视行车模式. 在该模式下, 列控车载设备生成NBP (normal brake profile)*为20km/h 的模式曲线.

   部分监控模式(PS) : 由于应答器信息接收异常导致线路数据缺失, 或者由于其它原因列控车载设备无线路数据, 以及侧线接车和在车站办理引导接车时, 列控车载设备的工作模式都定义为部分监控模式.

2. CTCS-3下模式介绍

   完全监控模式(FS) : 当车载设备具备列车控制所需的全部基本数据(包括列车数据, 行车许可和线路数据等) 时, 车载设备生成目标距离连续速度控制模式曲线, 并通过DMI显示列车运行速度, 允许速度, 目标速度和目标距离等信息, 监控列车安全运行.

   目视行车模式(OS) : 车载设备显示停车信号或位置不确定时, 在停车状态下司机按压专用按钮可使车载设备转入目视行车模式. 目视行车模式下, 车载设备按固定限制速度40km/h监控列车运行, 列车每运行一定距离司机需确认一次.

   引导模式(CO) : 当开放引导信号进行接发车时, 车载设备生成目标距离连续速度控制模式曲线, 并通过DMI显示列车运行速度, 允许速度, 目标速度和目标距离等, 车载设备按固定限制速度40km/h监控列车运行, 司机负责在列车运行时检查轨道占用情况.

   冒进模式(TR) : 列车执行冒进防护时, 车载设备进入冒进模式. 这个模式下列车会紧急制动, 直至完全停止.

2.3 Simulink/Stateflow基本概念

2.3.1 Simulink介绍

Simulink[1]是MATLAB最重要的组件之一, 它提供一个图形化的动态系统建模, 仿真和综合分析的集成环境. 在该环境中, 无需大量书写程序, 而只需要通过简单直观的鼠标操作, 就可构造出复杂的系统. Simulink是用于动态系统和嵌入式系统的多领域仿真和基于模型的设计工具, 拥有丰富的可扩充预定义模块库. 对各种实时系统, 包括通讯, 控制, 信号处理, 视频处理和图像处理系统, Simulink提供了交互式图形化环境和可定制模块库来对其进行设计, 仿真, 执行和测试.

*常用制动介入曲线

Simulink模型是由一系列的模块和连接这些模块的边组成的. 每个模块可以是一个Simulink模块库中的基本模块, 表达了输入输出之间的一个数学关系, 也可以是由若干模块组成的子系统. 连接边反应了两个被连接模块之间的关系.

Simulink模型通常由三部分组成: 输入信号源(Source) , 系统(System) 以及接收模块(Sink) .

1. 输入信号源模块库(Source)

   输入信号源模块库用来向模型提供输入信号, 没有输入口, 但是至少有一个输出口. 主要有: Constant, Step, Ramp, Sine Wave, Signal Generator, From File, From Workspace, Clock, In.

2. 系统模块库(System)

   系统模块主要包括连续模块, 离散模块, 数学模块, 逻辑模块, 信号处理模块, 系统模块. 连续模块主要有: Integrator, Derivative, State-Space, Transfer Fcn, Zero-Pole, Transport Delay. 离散模块主要有: Discrete-time Integrator, Discrete Filter, Discrete State-Space, Discrete Transfer-Fcn, Discrete , Zero-Pole, First-Order Hold , Zero-Order Hold, Unit Delay. 数学模块主要有: Sum, Product, Dot Product, Gain, Math Function, MinMax, Abs, Sign. 逻辑模块主要有: Compare To Zero , Compare to Constant, Inteval Test, Relational Operator, Logical Operator, Bit Set, Bitwise Operator, Bit Clear. 信号处理模块主要有: IC, Bus Creator, Bus Selector, Mux, Demux, Switch, Multiport Switch, Manual Switch, Selector. 系统模块主要有: SubSystem, Triggered Subsystem, Enabled Subsystem, Function-Call Subsystem, If Action Subsystem.

3. 接收模块库(Sink)

   接收模块是用来接收模块信号的, 没有输出口, 但是至少有一个输入口. 主要有: Scope, Display, XY Graph, To File, To Workspace, Stop Simulation, Out.

Simulink模块从其行为模式上分可以分为连续模块和离散模块两种. Simulink模型中的每个模块都有一个样本时间, 它的取值范围是-1或者非负浮点数. 如果这个模块的样本时间是-1, 表示其样本时间由其来源模块的样本时间计算而来; 如果这个模块的样本时间是0, 表示这个模块是连续模块, 其计算行为具有连续性; 如果是一个正值x, 表示该模块为离散模块, 其每隔x时间长度重新计算一次.

大多数Simulink模块都包含配置参数, 用户可以通过修改这些参数来获得预想的功能. 比如说常数模块可以设置输出常数值, 积分模块可以设置初始值, Switch模块可以设置操作符号类型和阈值.

图1描述了一个Simulink模型, 其中的Add模块从In1, In2接收了两个输入, 通过加法运算产生结果, 并通过Out1输出. 该模型中In1, In2, Out1和Add分别对应于输入信号源, 输出信号源与系统.

2.3.2 Stateflow介绍

Stateflow[2]是Simulink里面的一个工具箱, 它是一个基于状态机和流程图来构建组合和时序逻辑决策模型并进行仿真的环境. Stateflow 可以将图形表示和表格表示(包括状态转换图, 流程图, 状态转换表和真值表) 结合在一起, 针对系统对事件, 基于时间的条件以及外部输入信号的反应方式进行建模. 用户可以在使用Simulink仿真时, 使用这种图形化的工具实现各个状态之间的转换, 对复杂的监控逻辑进行建模.

Stateflow模型主要由事件变量集合, 状态, 结点和状态迁移组成:

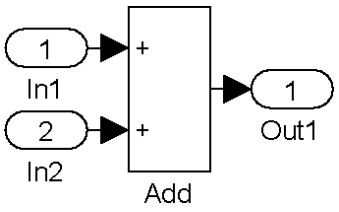

图1　Simulink 图表

Figure 1　A Simulink diagram

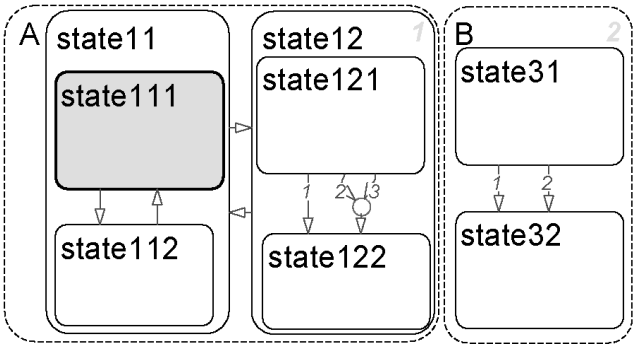

图2　Stateflow 图表

Figure 2　A Stateflow diagram

1. 事件变量集合

事件变量集合是记录Stateflow模型中使用的广播事件和变量的集合. 广播事件和变量均分为输入输出和局部三种, 分别表示从外部环境输入的广播事件或变量, 向外部输出的广播事件或变量和内部使用的广播事件或变量.

2. 状态

状态代表了系统现在处在的情况. 在Stateflow 下, 状态有两种行为: 活动的(active) 和非活动的(inactive) . 可以对一个状态进行标记, 包括给这个状态规定状态名以及这个状态在进入, 退出和处于激活状态下接收到一个事件所应执行的动作, 一般表示如下:

name

entry: entry actions

during: during actions

exit: exit actions

bind: data and events

on event_name: on event_name actions

入口动作(entry: entry actions) 是表示发生状态迁移, 激活了该状态时需要执行的动作. 中间动作(during: during actions) 表示原处于激活的状态受到一个事件的触发, 不存在从这个状态发出的状态迁移时, 此状态仍处于激活状态需要执行的动作. 出口动作(exit: exit actions) 表示存在由此状态发出的有效状态迁移时, 该状态退出时执行的动作. 数据事件绑定动作(bind: data and events) 将数据和事件绑定在此状态上. 绑定的数据只能在此状态或其子状态内被改写, 其他状态只能读取此数据. 绑定的事件由此状态或其子状态广播. 特定事件发生动作(on event_name: on event_name actions). event_name规定一个特定的事件; on event_name actions表示当该状态是激活状态且event_name规定的事件发生时需要执行的动作.

3. 结点

结点用于描述状态迁移过程中的迁移信号的分离和汇合. 通过使用结点, 状态和状态之间的迁移不仅仅是边的简单连接, 而是一个复杂的迁移网络.

4. 状态迁移

状态迁移†一般用于连接两个状态或者结点, 它可以被标记, 该标记的一般形式为如下四元组:

Event Trigger[Condition]Condition Action/Transition Action,

触发事件(Evnet trigger) 是事件集合中的一个元素, 作为迁移条件的一部分. 条件(Condition) 是一个布尔表达式, 与触发事件共同组成状态迁移的迁移条件. 触发事件(Evnet trigger) 为空或者与当前广播事件一致, 同时条件(Condition) 为真, 则该状态迁移被触发. 条件动作(Condition Action) 和迁移动作(Transition Action) 是使用Matlab的action language书写的代码片段. 条件动作在迁移被触发时被立即执行, 而迁移动作仅当迁移终点为状态时被立即执行, 否则被寄存在一个队列中, 若该迁移最终达到一个状态, 队列中的迁移动作将被顺序执行.

Stateflow模型具有层次化的特性, 任意Stateflow状态均可嵌入一个Stateflow子模型来丰富该状态的行为. 同一层次中, 所有的状态是互斥(OR) 或者并行(AND) 的, 互斥状态之间可以使用迁移进行连接, 而并行状态之间不能使用迁移连接.

Stateflow通讯采用广播机制, 当一个事件被广播时, 并行状态按照其预先定义的顺序先后接收到该广播事件, 互斥状态仅有激活状态收到该广播事件. 当状态接收到一个广播事件之后, 进行如下操作: 按照预先定义的顺序, 对状态的所有外部出口迁移进行检查, 如果该迁移网络能成功到达某个状态则执行对应的迁移动作, 并完成该轮事件广播; 否则, 按照预定的顺序对内部出口迁移进行检查, 如果该迁移网络能成功到达某个状态则执行对应的迁移动作, 并完成该轮事件广播; 如果上述两项均不成功, 则执行中间动作. 然后把广播事件对其包含的子图进行广播. 当状态迁移成功时, 需要找到源状态和目的状态之间的共同路径, 从内至外, 执行源状态出口动作至共同路径, 然后从外向内, 执行目标状态的入口动作.

图2描述了一个Stateflow的模型, 该图中有两个并行的状态A与B, 同时处于激活状态. 状态A和状态B中分别包含六个和两个互斥状态, 它们层次结构如图2所示. 图2中的迁移边包含三种特殊情况: 以实心黑点开头的迁移边为默认迁移边, 表明该箭头所指状态为默认激活状态; 状态S11到S2的迁移为层间迁移; 状态S21到S22的迁移为迁移网络, 该迁移网络包含一个结点.

2.4 混成CSP简介

2.4.1 混成CSP

HCSP (Hybrid Communicating Sequential Process) [5,6]在CSP (Communicating Sequential Process) 的基础上引入了微分方程以描述在混成系统中的连续动态行为, 并同时引入各种中断机制(条件中断和通讯中断) 来中断一个连续的微分行为, 进而使连续微分行为和离散控制行为交错运行. HCSP能同时刻画系统的连续行为和控制逻辑, 并具有CSP高可组合性的特性, 是一种很好的用来描述大型控制系统的形式化建模语言.

HCSP的语法如下所示:

$$
\begin{aligned}
P \quad &::= \quad \mathrm{skip} \mid x := e \mid \mathrm{ch}?x \mid \mathrm{ch}!e \mid P; Q \mid B \to P \mid P \sqcup Q \mid P^* \\
&\qquad \mid \langle \mathcal{F}(\dot{s}, s) = 0 \& B \rangle \mid \langle \mathcal{F}(\dot{s}, s) = 0 \& B \rangle \trianglerighteq [\!]_{i \in I} (\mathrm{io}_i \to Q_i) \\
S \quad &::= \quad P \mid S \| S.
\end{aligned}
$$

†默认迁移是一种特殊的迁移, 它仅有一个目标状态, 表示该状态为默认激活状态.

上式中P, Q, Q_i, S均为HCSP进程, x和s为进程变量, ch为通道名, io_i为通信事件(输入事件ch?x或输出事件ch!e) , B和e为布尔表达式和算术表达式. 以上语法结构的非形式化语义表示如下: skip不执行任何操作, 立即终止; $x := e$ 将表达式e的值赋给x; ch?x从通道ch中得到一个值, 并将其赋给x; ch!e将e的值发送到通道ch; $P; Q$先执行进程P, 当进程P终止时执行进程Q; $B \to P$在B为真时执行P, 否则立刻终止; $P \sqcup Q$由系统随机选择是执行进程P还是Q; P^*表示有限次重复执行进程P; $\langle \mathcal{F}(\dot{s}, s) = 0 \& B \rangle$代表微分动态过程, 该微分过程相关的变量取值必须使得布尔表达式B始终取真值, 否则该语句立即终止; $\langle \mathcal{F}(\dot{s}, s) = 0 \& B \rangle \trianglerighteq [\!]_{i \in I}(\mathrm{io}_i \to Q_i)$ 与$\langle \mathcal{F}(\dot{s}, s) = 0 \& B \rangle$ 执行过程基本一致, 但是当通信事件io_i发生时会立刻执行对应进程Q_i, 否则它会一直执行到微分过程终止为止; $S_1 \| S_2$中S_1和S_2为并发进程, 在不需要通讯时他们各自独立运行, 当需要通讯时通讯双方需要同步执行该通讯过程, 并发进程之间不能存在共享变量或是共享输入或者输出通道.

2.4.2 混成Hoare逻辑

在[7]中, 我们通过在经典的Hoare逻辑中引入历史公式构建了HCSP的证明系统. 之后, [8]将该逻辑在Isabelle/HOL中实现, 并利用该工具验证了中国铁路控制系统中的一个实际案例. 历史公式由一个时段验算公式[19,20]表达, 用于描述系统在执行时间区段内满足的性质. 在混成Hoare逻辑(HHL) 中, 顺序进程P的性质描述由三元组$\{\mathrm{pre}\} P \{\mathrm{post}; \mathrm{HF}\}$来表达, 其中pre, post, 和HF分别表示前置条件, 后置条件, 和历史公式. 前置条件和后置条件由一阶逻辑公式表达, 历史公式由时段验算公式表达. 对于并发进程其性质描述如下表示.

$\{\mathrm{pre}_1, \ldots, \mathrm{pre}_n\} P_1 \parallel \cdots \parallel P_n \{\mathrm{post}_1, \ldots, \mathrm{post}_n; \mathrm{HF}_1, \ldots, \mathrm{HF}_n\}$,

其中pre_i, post_i和HF_i分别表示第i个进程的前置条件, 后置条件和历史公式.

3 场景描述及其建模过程

我们将高速铁路列控系统在等级转换和模式转换的两个场景中涉及的主体抽象为由列控系统, 列车与司机组成. 列控系统分为多个等级, 我们只考虑列车在C2级和C3级下面的行为, 所以只涉及C2级列控系统与C3级列控系统, 分别是TCC (Train Control Center) , RBC (Radio Block Center) , 其主要作用是根据车载子系统, 地面子系统等提供的列车状态, 轨道占用, 临时限速命令, 联锁进路状态, 灾害防护等信息产生其控制范围内各个列车的行车许可等控制信息, 并且传输给车载设备以控制列车的运行. 列车运行的过程中, 司机要监控列车的运行, 需要经常对不同的情况作出反应.

图3是表示整个列控系统的Stateflow模型, 其中C2级控制器, C3级控制器, 列车和司机四个部分分别由由TCC, RBC, Train和Driver四个状态表示, 这四个部分需要并行执行, 所以这四个状态之间的关系为并行. 列车在运行过程中需要一个动力系统来计算速度, 由于动力系统是连续的, 因此我们在Stateflow外使用一个子系统来表示, 如图4所示, 这个子系统的输入是加速度, 输出是速度和距离, 使用了两个积分模块, 分别代表加速度的积分为速度, 速度的积分为距离.

运营场景是对运营中系统工作方式的简要描述, 在这个模型中, 我们考虑三个运营场景: 行车许可（Movement Authority, MA）场景, 控制器会对列车的运行过程进行监控, 并且通过MA授权控制列车的行为; 等级转换场景, 列车在运行中可以从C2级转换到C3级, 图5是等级升级场景的一个示意图; 模式转换场景, 列车在不同的模式下面也会有不同的行为模式, 不同等级下面不同的模式之间的转换有不同的行为模式. 下面我们会详细介绍这三个场景. 因为从文档中推测等级转换和模式转换相结合

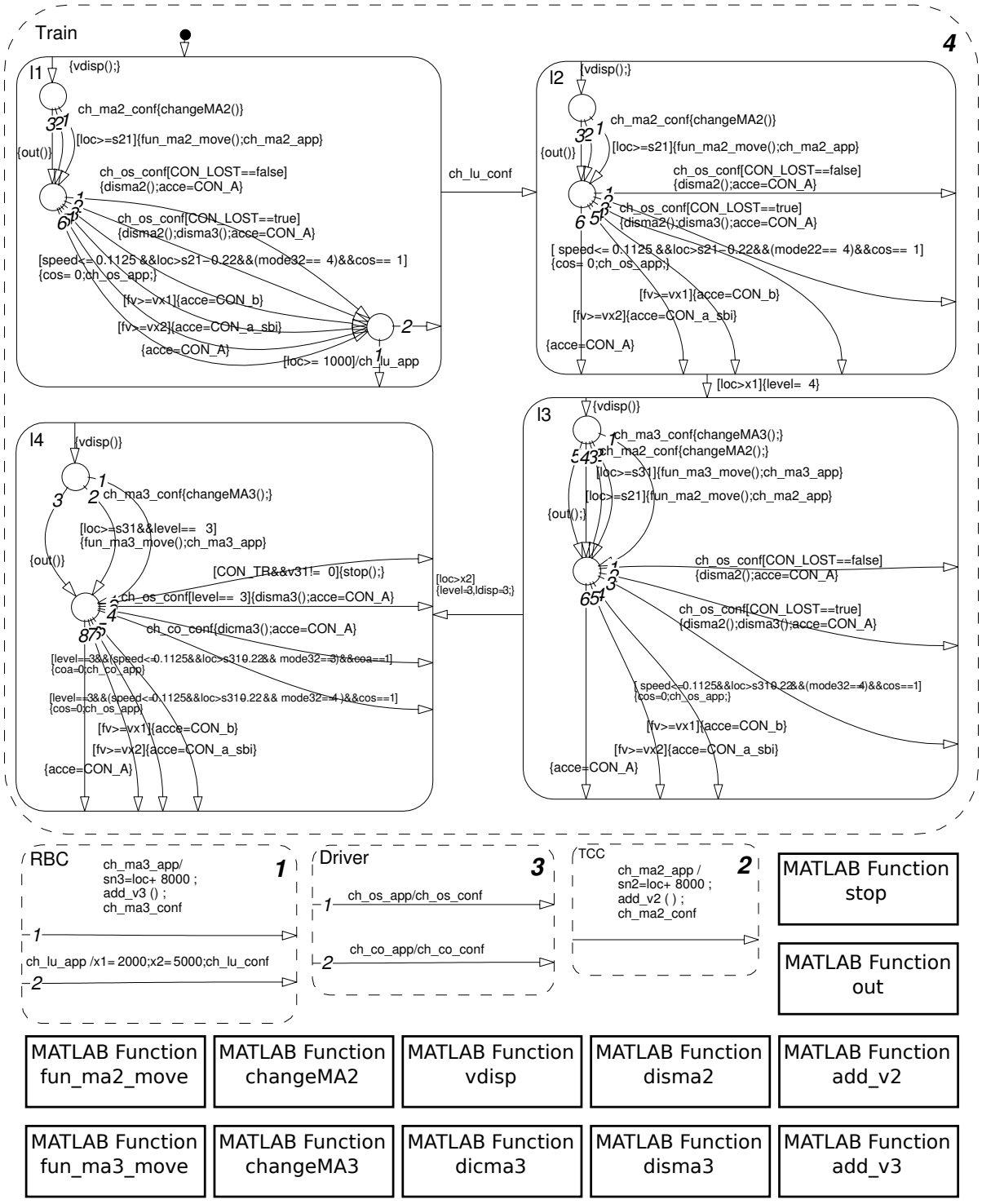

图3 列控系统

Figure 3 The Train Control System

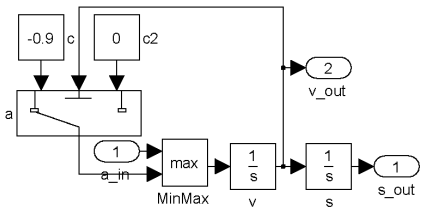

图4　动力系统

Figure 4　Speed system

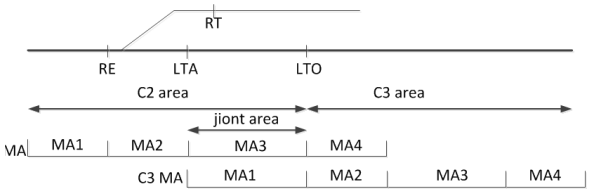

图5　等级转换场景

Figure 5　The scene of level change

可能会产生问题, 因此我们将模拟等级转换和模式转换相结合的场景, 其中等级转换点和模式转换点是同一点.

3.1　行车许可

MA是列车安全运行的行车凭证. MA终点是指列车被授权运行到的位置. 一个行车许可包括多个连续的区段, 每个区段由一个五元组表示, 具体如下:

MA区段={等级, 模式,目标距离,紧急制动介入速度,常用制动介入速度},

从之前的描述中可以知道, 列车的行为模式均由列车当时处在的等级和模式所决定, 即五元组的第一项和第二项. 模型只涉及C2级与C3级两个等级, 分别用1和2表示, 并且只考虑FS, PS, CO, OS, TR五个模式, 分别用1, 2, 3, 4, 5表示.

列车在运行过程中会不断的更新MA列表. 我们假定列车在运行完一个区段之后, 会向TCC (C3级下为RBC) 申请新的区段, 发出ch_ma2 (ch_ma3) 信号, TCC (RBC) 接到该申请之后, 将查看列车允许的位置生成新的MA, 并且发送ch_ma2_conf (ch_ma3_conf) 和新的MA给列车, 如TCC中的迁移和RBC中的迁移1所示. 列车接收到新的MA之后将覆盖原有MA.

列车会不断根据目前的MA计算列车有效加速度范围, 计算主要依据列车的两类曲线:

1. 静态速度曲线(SSP: Static Speed Profile)

   由每个区段的所有最高速度限制组成, 包括紧急制动介入速度和常用制动介入速度两个速度限制;

2. 动态速度曲线(DSP: Dynamic Speed Profile)

   包括紧急制动介入曲线和常用制动介入曲线, 由MA的目标距离和静态速度曲线计算而来. 列车在运行的过程中任意时刻需保证不能超越静态速度曲线和动态显示曲线.

   紧急制动介入曲线(EBI: Emergency brake intervention): 是根据目标距离和紧急制动介入速度计算而来的速度曲线, 如果列车当前速度超出了紧急制动介入曲线, 列车就要以最大减速度进行紧急制动;

   常用制动介入曲线(SBI: Service brake intervention): 是根据目标距离和常用制动介入速度计算而来的速度曲线, 如果列车目前的速度超出了常用制动介入曲线, 那么列车要以常用制动减速度进行制动.

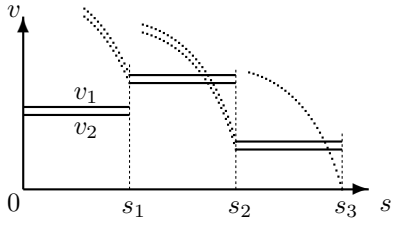

图6 MA示例

Figure 6 Example of MA

如果列车的当前速度没有超出常用制动介入曲线, 那么就把列车的加速度设置为常用制动减速度以及最大加速度之间的某个值.

在模型中我们使用out函数计算列车当前是否超过EBI或者SBI. 如果超过了EBI, 则变量eout的值设为1, 如果超过了SBI, 则变量sout的值设为1, 在之后的迁移中分别查看eout和sout的值设置加速度值a. 列车在运行中的每一刻都要计算拥有的所有MA区段的常用制动介入曲线和紧急制动介入曲线, 每一点均取所有MA区段的紧急制动介入曲线对应位置的最小值作为当前的紧急制动介入速度, 取所有MA区段的常用制动介入曲线对应位置的最小值作为当前的常用制动介入速度, 这样叠加形成该区段的紧急制动介入曲线和常用制动介入曲线, 也即列车运行中的每一刻都不可以超越目前拥有的任何一个MA区段计算得到的紧急制动介入曲线和常用制动介入曲线, 这样确保如果列车在这个区段正常运行, 就不会违反下个区段起点的速度上限.

图6是一个有三个MA区段的例子, 由点s_1, s_2, s_3分开. 这个例子中, 我们假设最后一个区段的限速曲线终点速度为零, 因此之后如果不继续拓展列车的MA区段列表, 列车就需要在s_3点停下. 图中的六条直线分别代表三个MA区段的静态速度曲线, 六条曲线(最后一段的紧急制动介入曲线和常用制动介入曲线重合, 因为末点的紧急制动介入速度和常用制动介入速度都是0) 分别代表三个MA区段各自根据上述公式计算得到的紧急制动介入曲线和常用制动介入曲线. 根据如下公式进行计算动态曲线$v^2 + 2b\,s < \mathbf{next}(\text{seg}).v_1^2 + 2b\,\text{seg}.s$, 这里的$\mathbf{next}(\text{seg})$代表列车下一个要运行的MA区段, b代表紧急制动加速度, $\text{seg}.s$代表这一段MA区段的终点位置.

3.2 等级转换

等级转换是列控系统的主要功能之一, 列车由C2级线路进入C3级线路, 需要在固定地点进行转换. 等级转换前, 应完成的工作包括注册到GSM-R（Global System Mobile-Railway）网络, 与RBC建立通信会话, 从RBC获得系统配置参数和行车许可等信息. 直至C3级控车条件具备后, 车载设备自动转入C3级工作. 如图5所示. 主要过程如下:

1. 列车与GSM-R建立链接, 与RBC建立通信会话

   从C2级控车转换为C3级控车首先应建立车载设备与GSM-R的通信链接.

   车载设备与GSM-R网络建立链接后, 从C2级转换为C3级应建立车载设备与RBC的通信会话, 为此在转换区入口处设置RBC连接应答器组(RE) , 该应答器组应向接近的列车发送用于建立通

信会话的命令. 列车之后可能进入不通往C3级的分支, 这个时候应该设置一个命令列车关闭通信会话的应答器组(RT).

2. 获得行车许可

C3级系统应只向真正进入C3级区域的列车提供行车许可. C3级系统在至转换边界前唯一进路的区域设置用于向车载设备提供准确进路的级间转换预告应答器组(Level Transition Announcement, LTA) . 当列车前端通过预告应答器组(LTA) 时, 车载设备向RBC报告列车位置, RBC据此确定列车接近的准确进路, 然后根据C3级控制区域联锁进路信息, 向车载设备提供包括线路参数的行车许可(MA) 及级间转换命令. 列车在到达转换边界前始终由C2级系统控车, 车载设备将储存此前获得的C3级行车许可(MA) 并在其通过C2/C3级边界并转换到C3级系统控车时使用. 在获得行车许可到等级转换执行这段期间内, 列车会受到C2和C3级行车许可的共同控制.

3. 执行等级转换

当列车前端通过C2/C3级边界, 根据RBC的命令并具备C3级系统控车条件时, 车载设备将自动转为C3级控车. 在C2/C3级间转换边界设置转换执行应答器组(Level Transition Operation, LTO)向列车发送消息. 为保持在等级转换边界列车能以C2级最高允许速度运行, C2级系统的行车许可必须越过C2/C3级边界至少一个完整常用制动距离. 之后在C3级系统控车过程中, C2级车载设备控制单元处于后台工作状态, 不对列车有控制作用.

图3中Train模块展示了等级转换场景, 该模块中的四个状态分别代表列车在运行至RE之前, RE到LTA之间, LTA到LTO之间, 与越过LTO之后四个阶段. 第一个阶段中列车正常启动运行, 直到到达RE建立连接. 第二个阶段中, 列车已经与RBC通过GSM-R网络建立了连接, 列车向RBC报告了目前位置, 并且收到了RBC发送的等级转换预备点和等级转换执行点位置, 列车继续在C2级MA控制下, 直到列车到达LTA. 第三个阶段中, 列车已经接收到RBC发来的MA, 列车开始受到C2级和C3级的共同控制, 直到到达LTO. 如果列车越过等级转换点则标志着等级转换成功, 列车进入C3级运行.

3.3 模式转换

C2级与C3级的模式转换有一些不同的行为方式, 这一小节我们将分别详细介绍C2级下面和C3级下面的模式转换.

1. C2级模式转换

当列车缺少线路数据, 便可进入PS模式. 当列车接收到的行车许可模式为OS时, 列车便可以转入OS, 转入的条件是列车在非OS区段末停车, 显示目视行车键, 当司机按压了目视行车键后, 便可以转入OS模式. 我们使用列车模块向司机模块发出ch_os_app信号, 司机模块在收到之后向列车模块返回ch_os_conf信号来模拟OS模式的目视行车键请求过程. 列车在引导区域可以进入引导区的PS模式, 这个时候行为模式与PS模式一致, 但是上限速度比PS模式要低. 进入这种模式列车不需要进行确认, 只需要速度降低到该模式允许速度. 当线路数据完整时, 列车只要接收到FS模式的行车许可就可以转入FS模式. 图3中I1, I2, I3均实现了这些模式转换.

2. C3级模式转换

当列车接收到的行车许可模式为CO时, 列车便可以转入CO, 转入的条件是列车速度降低到CO模式的允许速度, 并且司机进行确认. 我们使用列车模块向司机模块发出ch_co_app信号, 司机模块在收到之后向列车模块返回ch_co_conf信号来模拟CO模式的目视行车键请求过程. 当列车接收到的行车许可模式为OS时, 列车便可以转入OS, 转入的条件是列车在非OS区段末停车, 显示目视行车键, 当司机按压了目视行车键后, 便可以转入OS模式, 确认行为的模拟方式与C2级一致. 当线路数据完整时, 列车接收到FS模式的行车许可便可以转入FS模式. 当列车运行越过等级转换点的同时接收到RBC发送的缩短MA命令, 自动转到C3级下的TR模式, 这时列车进行紧急制动, 最后停在等级转换点之后的某一点. I4实现了这些模式转换.

4 模拟结果

上述模型包含了第3章节描述的所有场景, 通过修改模型参数, 我们可以模拟不同的组合情况. 需要修改的参数包括如下两个方面: 通过修改MA初始值来修改轨道参数信息, 从而对不同的组合情况进行仿真; 通过修改仿真常数(表1为仿真常数列表) 将模型的行为确定化.

表1 仿真常数列表

Table 1 List of simulation parameters

v21, v22, v23, v24	The limit speeds of emergency braking of the 4 MA segments in C2
vr21, vr22, vr23, vr24	The limit speeds of normal braking of the 4 MA segments in C2
s21, s22, s23, s24	The distances of the targets of the 4 MA segments in C2
mode21, mode22, mode23, mode24	The modes of the 4 MA segments in C2
v31, v32, v33, v34	The limit speeds of emergency braking of the 4 MA segments in C3
vr31, vr32, vr33, vr34	The limit speeds of normal braking of the 4 MA segments in C3
s31, s32, s33, s34	The distances of the targets of the 4 MA segments in C3
mode31, mode32, mode33, mode34	The modes of the 4 MA segments in C3
CON_LOST	Mark whether or not the confirming message of OS mode in C2 controller has delivered to C3 controller
CON_TR	Mark whether or not the train has received the command to shorten the MA in C3

C2级4个初始MA区段和C3级4个初始MA区段位置并不一致, 参照图5中显示. C2级MA从RE之前开始, 但是C3级MA从经过LTA之后开始, 这表示只有在经过LTA之后列车才接收到C3级MA并且对列车的运行起到控制作用.

利用建立好的模型, 我们对部分组合情况进行了仿真, 结果如图7所示. 图7由12幅子图构成, 每幅子图描述了一个组合情况的运行结果. 每幅子图横轴均表示时间, 纵轴为仿真信号的数值结果, 图中粗实线(绿线) 表示列车所在位置(单位为150m) 随时间的变化情况, 细实线(蓝线) 表示列车运行速度(单位为m/s) 随时间的变化情况, 稀疏虚线(紫线) 表示列车的常用制动介入曲线, 紧密虚线(红线) 表示列车所处等级(10表示处于C2级, 15表示处于C3级) 随时间变化情况.

1. C2→C3&&OS→TR

   为了模拟该组合情况, 我们将C2级和C3级MA模式均设置为OS, 为了表示在列车越过等级转换点的同时接收到缩短MA命令, 我们将CON_TR设置为true.

   仿真结果如图7-1所示, 列车在等级转换点之前依照C2级OS模式运行, 列车越过等级转换点的同时接收到缩短MA的消息, 立即进入TR模式, 列车紧急刹车, 停在等级转换点之后某处.

2. C2→C3&&OS→CO

   为了模拟该组合情况, 我们将C2级MA前3段模式设置为OS, 第4段的模式设置为PS; 同时将C3级MA 第1段模式设置为OS, 后3段模式设置为CO.

   仿真结果如图7-2所示, 列车在等级转换点之前依照C2级OS模式运行, 依照MA设定之后转入C3级CO 模式, 但是在C2级下无法显示列车在C3级转入CO的确认键信息, 所以司机无法对进入C3级CO模式进行确认, 因此转入CO模式失败, 列车在等级转换点处停车.

3. C2→C3&&OS→FS

   为了模拟该组合情况, 我们将C2级MA前3段模式设置为OS, 第4段模式设置为FS; C3级MA第1段模式设置为OS, 后3段模式设置为FS.

   仿真结果如图7-3所示, 列车在等级转换点之前依照C2级OS模式运行, 越过等级转换点之后转入C3级依照FS模式运行.

4. C2→ C3&&PS→ OS

   为了模拟该组合情况, 我们将C2级MA前3段模式设置为PS, 第4段模式设置为OS; C3级MA第1段模式设置为FS, 后3段模式设置为OS. 为了模拟目视行车键确认消息从C2级传递到C3级成功的情况, 我们设置CON_LOST为true, 为了模拟目视行车键确认消息从C2级传递到C3级失败的情况, 我们设置CON_LOST为false.

   目视行车键确认消息从C2级传递到C3级成功的仿真结果如图7-4所示, 列车在等级转换点之前依照C2级PS模式运行, 在等级转换点停车, 显示目视行车键, 司机确认后, 成功转入C3级OS模式, 列车继续运行. 目视行车键确认消息没有从C2级传递到C3级下的情况如图7-5所示: 列车在等级转换点之前依照C2级PS模式运行, 在等级转换点停车, 显示目视行车键, 司机的确认后, 确认消息返回给C2级控制器, 但是由于故障, 确认消息未能传递给C3级控制器, 模式转换不成功, 停在等级转换点.

5. C2→ C3&&PS→ CO

   为了模拟该组合情况, 我们将C2级MA前3段模式设置为PS, 第4段模式设置为CO; C3级MA第1段模式设置为FS, 后3段设置为CO.

该场景仿真结果如图7-6所示, 列车在等级转换点之前依照C2级PS模式运行, 依照MA设定之后要转入C3级CO模式, 但是在C2级下无法显示列车在C3级转入CO的确认键信息, 所以司机无法对进入C3级CO模式进行确认, 因此转入CO模式失败, 列车在等级转换点处停车.

6. C2→ C3&&PS→ TR

为了模拟该组合情况, 我们将C2级MA模式均设置为PS; C3级MA 模式均设置为FS, 为了表示在列车越过等级转换点的同时接收到缩短MA命令, 我们将CON_TR设置为true.

结果如图7-7所示, 列车在等级转换点之前依照PS模式运行, 越过等级转换点之后立即进入TR模式, 列车紧急制动, 停在等级转换点之后某处.

7. C2→ C3&&PS→ FS

为了模拟该组合情况, 我们将C2级MA模式前3段设置为PS, 第4段设置为FS; C3级MA模式均设置为FS.

结果如图7-8所示, 列车在等级转换点之前依照C2级PS模式运行, 越过等级转换点之后转入C3级依照FS模式运行.

8. C2→C3&&FS→TR

为了模拟该组合情况, 我们将C2级MA模式均设置为FS; C3级MA第1段模式设置为FS, 为了表示在列车越过等级转换点的同时接收到缩短MA命令, 我们将CON_TR设置为true.

结果如图7-9所示, 列车在等级转换点之前依照FS模式运行, 越过等级转换点之后立即进入TR模式, 列车紧急制动, 停在等级转换点之后某处.

9. C2→C3&&FS→OS

为了模拟该组合情况, 我们将C2级MA前3段模式设置为FS, 第4段模式设置为OS; C3级MA第1段模式设置为FS, 后3段模式设置为OS. 为了模拟目视行车键确认消息从C2级传递到C3级成功的情况, 我们设置CON_LOST为true, 为了模拟目视行车键确认消息从C2级传递到C3级失败的情况, 我们设置CON_LOST为false.

目视行车键确认消息从C2级传递到C3级成功的仿真结果如图7-10所示, 列车在等级转换点之前依照C2级FS模式运行, 在等级转换点停车, 显示目视行车键, 司机确认后, 成功转入C3级OS模式, 列车继续运行. 目视行车键确认消息没有从C2级传递到C3级下的情况如图7-11所示: 列车在等级转换点之前依照C2级FS模式运行, 在等级转换点停车, 显示目视行车键, 司机的确认后, 确认消息返回给C2级控制器, 但是由于故障, 确认消息未能传递给C3级控制器, 模式转换不成功, 停在等级转换点.

10. C2→C3&&FS→CO

为了模拟该组合情况, 我们将C2级MA前3段模式设置为FS, 第4段模式设置为CO ; C3级MA第1段模式设置为FS, 后3段模式设置为CO.

结果如图7-12所示, 列车在等级转换点之前依照C2级FS模式运行, 依照MA设定之后要转入C3级CO 模式, 但是在C2级下无法显示列车在C3级转入CO的确认键信息, 所以司机无法对进入C3级CO模式进行确认, 因此转入CO 模式失败, 列车在等级转换点处停车.

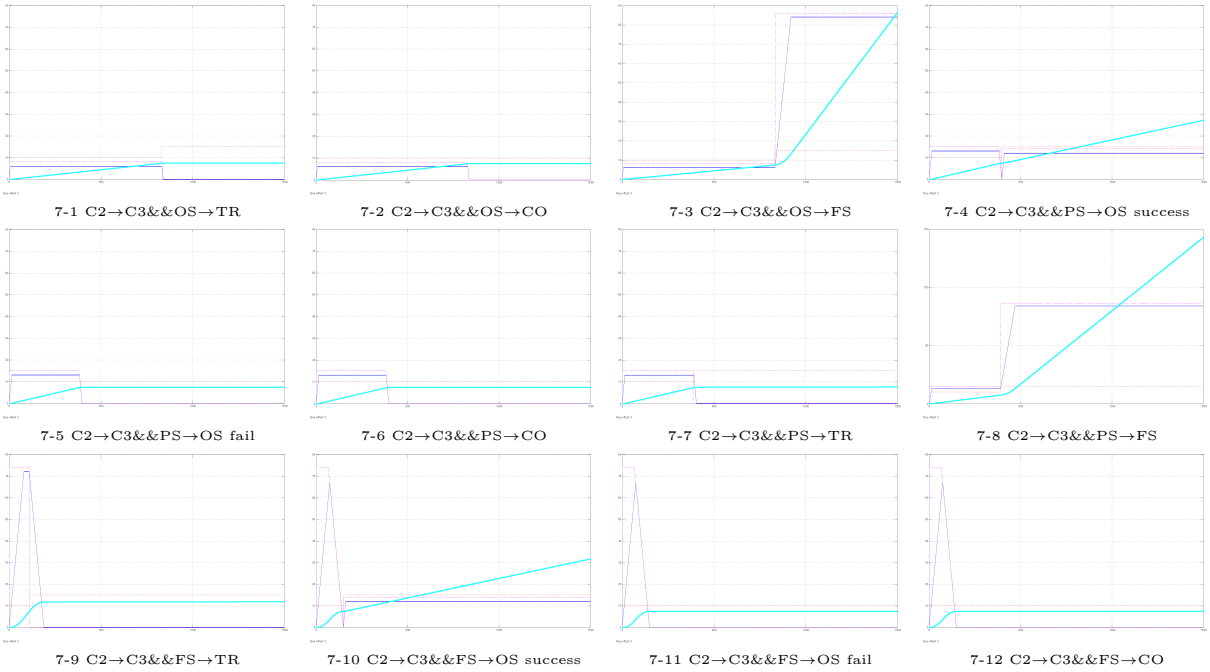

图7　仿真结果

Figure 7　Simulation result

综上所述,我们通过对上述10个情景进行仿真, 我们发现部分情景中会出现不正常停车的情况, 具体如表2所示.

表2　组合情况结果

Table 2　Result of the combination

C2→C3&&OS→TR	Stop normally
C2→C3&&OS→CO	Stop abnormally
C2→C3&&OS→FS	Run normally
C2→C3&&PS→OS	Maybe stop abnormally
C2→C3&&PS→CO	Stop abnormally
C2→C3&&PS→TR	Stop normally
C2→C3&&PS→FS	Run normally
C2→C3&&FS→TR	Stop normally
C2→C3&&FS→OS	Maybe stop abnormally
C2→C3&&FS→CO	Stop abnormally

5　形式验证

由表2可知, 在部分组合情况下会出现停车现象. 然而, 由于仿真仅能选择一种系统的可能运行情况进行测试, 并不能表明系统的所有运行情况. 例如, 在我们的模型中, 司机的行为具有很明确的确定

性, 即永远选择所有可能加速度里面最大的加速度进行行车. 从某种意义上说, 我们可以猜测这这选择是火车最危险的行车方式, 也是火车最不可能停车的情况. 由于仿真与生俱来的不完备性, 我们对其中PS到CO的情况的模型转化为HCSP进程, 并利用其定理证明器HHL Prover进行了证明. 通过证明我们确认在这种组合场景下任何情况都会发生不正常停车. 在证明过程中, 由于我们仅关心PS到CO模式转化的情况, 在我们验证的模型中我们将与此不相关的部分内容进行了简化处理.

5.1 形式转换

通过使用工具**S**im2**H**CSP[3,4], 我们获得了该模型的HCSP的进程, 它包括7个文件, 分别对应于Simulink和Stateflow的变量定义, 进程定义, 断言定义, 和最终的证明目标, 这些文件一共有1351行, 他们分别是: controlPDef.thy, controlADef.thy, controlVarDef.thy, varDef.thy, assertionDef.thy, processDef.thy 和goal.thy. 其中最后三个文件主要用于形式证明, 将在下一章节介绍. 下面, 我们简要介绍前四个文件内容:

```
theory varDef
        imports "HHL"
begin
(*Define channel names.*)
definition Ch_plant_v_1_1 :: cname where
   " Ch_plant_v_1_1 = "Ch_plant_v_1_1""
...
(*Define receiving variables.*)
definition plant_v_1_1 :: exp where
    "plant_v_1_1 =
    RVar "plant_v_1_1""
...
(*Local and sending variables.*)
definition plant_v_1 :: exp where
    "plant_v_1 = RVar "plant_v_1""
definition plant_s_1 :: exp where
    "plant_s_1 = RVar "plant_s_1""
...
end
```

processDef.thy主要结构如下, 主要定义了转化后的HCSP模型的整体框架, 该框架包含了Stateflow图对应的HCSP进程Pcontrol.

```
theory processDef
  imports "controlPDef"
begin
(*Define continuous processes*)
...
definition PC1_init :: proc where
  "PC1_init = PC1_1;assertion2;PC1_2"
definition PC1_rep :: proc where
  "PC1_rep = PC1_3;assertion3;PC1_4"
definition PC1 :: proc where
  "PC1 = PC1_init;assertion4;(PC1_rep)*"
(*Define the whole process.*)
definition P :: proc where
  "P = PC1||Pcontrol"
end
```

controlVarDef.thy主要结构如下, 主要定义了Stateflow图对应的HCSP进程中使用的变量.

```
theory controlVarDef
        imports "assertionDef"
begin
(*Define channel names.*)
definition BC_1 :: cname where
    "BC_1 = "BC_1""
```

```
definition BR_1 :: cname where
    "BR_1 == "BR_1""
definition BO_1 :: cname where
    "BO_1 == "BO_1""
definition VO1 :: cname where
    "VO1 == "VO1""
definition VI1 :: cname where
    "VI1 == "VI1""
definition vBO1 :: exp where
    "vBO1 == SVar "vBO1""
...
(*Define event variables assistent.*)
consts eL :: "exp list"
consts nL :: "exp list"
(*Define event variables.*)
definition E1 :: exp where
    "E1 == SVar "E1""
definition done1 :: exp where
    "done1 == RVar "done1""
...
    "E == SVar "E""
definition num :: exp where
    "num == RVar "num""
definition EL :: exp where
    "EL == List eL"
definition NL :: exp where
    "NL == List nL"
(*Define local and sending variables.*)
definition s :: exp where
    "s == RVar "s""
definition v :: exp where
    "v == RVar "v""
definition a :: exp where
    "a == RVar "a""
definition CONFR :: exp where
    "CONFR == RVar "CONFR"
...
end
```

controlPDef.thy主要结构如下, 主要定义了Stateflow图对应的HCSP进程, 该进程作为一个进程使用于整体HCSP模型processDef中. 其中, 进程Pcontrol7定义了一个控制进程, 用于描述Stateflow图的语义. Pcontrol11 , Pcontrol14, Pcontrol86 和Pcontrol89分别定义了RBC, TCC, 车控系统和司机的行为.

```
theory controlPDef
        imports "controlADef"
begin
(*Define the processes for MATLAB fuctions.*)
definition Fcontrol1 :: proc where
"Fcontrol1 ==
e31 := e32;assSF1;
e32 := e33;assSF2;
e33 := (e33 [+] (Real 32000));assSF3;
v311 := v321;assSF4;
v312 := v322"
...
definition FMA3 :: proc where
    "FMA3 == Fcontrol1;assSF10;Fcontrol2;assSF11;Fcontrol3"
...
definition Pcontrol7 :: proc where
    "Pcontrol7 == ((num:=(Real 0);assSF92;E:=(String " ");assSF93;(a:=(Real 0)));
        assSF94;Ch_control_1_0!!a);assSF95;(Pcontrol1;assSF96;Pcontrol2;assSF97;
        Pcontrol3;assSF98;Pcontrol4;assSF99;Pcontrol5;assSF100;Pcontrol6)*"
```

```
definition Pcontrol8 :: proc where
    "Pcontrol8 == IF ((done2[=](Real 0))[&]E2[=](String "LUA"))
        (actRBC:=(Real 0);assSF101;actRBC:=(Real 1);
        assSF102;BR_2!!(String "LU");assSF103;done2:=(Real 1))"
definition Pcontrol9 :: proc where
    "Pcontrol9 ==
        IF ((done2[=](Real 0))[&]E2[=](String "MAA3"))
        (actRBC:=(Real 0);assSF104;actRBC:=(Real 1);
        assSF105;BR_2!!(String "MAA3c");assSF106;done2:=(Real 1))"
definition Pcontrol10 :: proc where
    "Pcontrol10 == done2:=(Real 0)"
definition Pcontrol11 :: proc where
    "Pcontrol11 == Pcontrol10;assSF107;
        (BC_2??E2;assSF108;(Pcontrol8;assSF109;Pcontrol9;
        assSF110;done2:=(Real 0));assSF111;BO_2!!(String ""))*"
definition Pcontrol12 :: proc where
    "Pcontrol12 == IF ((done3[=](Real 0))[&]E3[=](String "MAA2"))
        (actTCC:=(Real 0);assSF112;actTCC:=(Real 1);
        assSF113;BR_3!!(String "MAA2c");assSF114;done3:=(Real 1))"
definition Pcontrol13 :: proc where
    "Pcontrol13 == done3:=(Real 0)"
definition Pcontrol14 :: proc where
    "Pcontrol14 == Pcontrol13;assSF115;(BC_3??E3;assSF116;
        (Pcontrol12;assSF117;done3:=(Real 0));assSF118;BO_3!!(String ""))*"
...
definition Pcontrol86 :: proc where
    "Pcontrol86 == Pcontrol85;assSF355;
        (BC_1??E1;assSF356;(Pcontrol84;assSF357;
        done1:=(Real 0));assSF358;BO_1!!(String ""))*"
definition Pcontrol87 :: proc where
    "Pcontrol87 == IF ((done4[=](Real 0))[&]E4[=](String "CONFR"))
         (actDriver:=(Real 0);assSF359;actDriver:=(Real 1);
         assSF360;BR_4!!(String "CONF");assSF361;done4:=(Real 1))"
definition Pcontrol88 :: proc where
    "Pcontrol88 == done4:=(Real 0)"
definition Pcontrol89 :: proc where
    "Pcontrol89 == Pcontrol88;assSF362;(BC_4??E4;assSF363;
    (Pcontrol87;assSF364;done4:=(Real 0));assSF365;BO_4!!(String ""))*"
definition Pcontrol :: proc where
    "Pcontrol == Pcontrol7||Pcontrol11||Pcontrol14||Pcontrol86||Pcontrol89"
end
```

5.2 形式验证

在HHL Prover中, 我们证明的结论如下:

```
lemma goal : \{T,T,T,T,T,T\} P \{plant\_s\_1 $\leq$ 4000,T,T,T,T,T;
                (l = 0) $|$(high (plant\_s\_1$\leq$4000)),T,T,T,T,T\}.
```

从该结论中, 我们可以得出火车的位置plant_s_1始终不能超过4000米, 即火车会停车.

由于文件assertionDef.thy和controlADef.thy仅提供插入断言以辅助证明, 我们在以下介绍中将忽略这两个文件. 以下为证明文件的主要结构:

```
(*Goal for the whole process.*)
lemma goal : "{WTrue,WTrue,WTrue,WTrue,WTrue,WTrue} P
    {(plant_s_1[<=](Real 4000)),WTrue,WTrue,WTrue,WTrue,WTrue;
    (l [=] Real 0) [||] (high (plant_s_1[<=](Real 4000))),
    WTrue,WTrue,WTrue,WTrue,WTrue}"
```

证明过程分为以下三步:

1. 将证明目标划分为初始化证明过程和重复计算两个过程;
2. 证明初始化过程满足性质要求:

   重复使用并发规则, 顺序组合规则, 和赋值语句规则证明该性质.
3. 证明重复过程满足性质要求:

   (a) 使用重复语句(repetition statement) 的规则消去证明目标中的重复部分,

   (b) 使用并发通讯规则消去证明目标中的通讯,

   (c) 集中证明剩余的算术性质, 该部分确保火车不会超过目标点.

由于版面所限, 我们仅给出步骤1的相关代码, 全部代码见“http://lcs.ios.ac.cn/ zoul/casestudies/p-sco.rar” .

```
apply (simp add: P_def)
apply (simp add: PC1_def Pcontrol_def)
apply (simp add: Pcontrol7_def Pcontrol11_def
    Pcontrol14_def Pcontrol86_def Pcontrol89_def)
apply (simp add: assertion4_def assSF95_def
    assSF107_def assSF115_def assSF355_def assSF362_def)
apply (cut_tac Ha="HisP1" and Hb="WTrue" and Hc="WTrue"
    and Hd="WTrue" and He="WTrue" and Hf="WTrue" in ParallelSeq6,auto)
defer defer
apply (simp add: HisP1_def)
apply (rule impR, rule LL3a, rule basic)
apply (rule Trans,auto)+
defer
```

6 总结与未来的工作

在这篇文章中, 我们提出一套高速铁路控制系统图形建模, 仿真, 形式建模与验证的方法, 包括: 使用Simulink/Stateflow 对高速铁路列控系统的进行图形建化Simulink/Stateflow图形模型转换成HCSP形式模型; 利用HHL定理证明器进行形式验证. 本文以行车许可, 等级升级及部分模式转换场景为例说明上述方法的有效性. 特别通过上述方法, 我们发现并验证原来规范中存在的一些错误. 以上工作将有助于指导改进高速铁路列控系统规范.

今后我们希望在该模型中加入RBC切换等场景,使模型更加完善, 从而可以对更加复杂的高速铁路列控系统运行情况进行仿真, 并进而能够对所有的可能存在问题的场景进行形式验证. 并且会进一步完善验证工具, 特别是利用Simulink/Stateflow模块定制一个面向高速列车控制系统的图形化建模语言.

参考文献

1 The Mathworks. Simulink User's Guide version 7, 2009

2 The Mathworks. Stateflow User's Guide version 6, 2006

3 Zou L, ZHAN N, WANG S, et al. Verifying Simulink diagrams via a Hybrid Hoare Logic Prover. In: Proc. of EMSOFT 2013. Washington: IEEE Computer Society, 2013.1-10

4 ZOU L, ZHAN N, WANG S, FRÄNZLE M. Formal Verification of Simulink/Stateflow Diagrams. SKLCS of ISCAS Technical Report ISCAS-SKLCS-13-07. 2013

5 HE J. From CSP to hybrid systems. In: Roscoe A W, eds. A Classical Mind. Prentice Hall International (UK) Ltd, 1994, 171-189

6 ZHOU C, WANG J, RAVN A P. A formal description of hybrid systems. In: Proc. of Hybrid Systems III, Lecture Notes in Computer Science 1066. Berlin, Heidelberg: Springer, 1996. 511-530

7 LIU J, LV J, QUAN Z, et al. A calculus for hybrid CSP. In: Proc. of APLAS 2010, Lecture Notes in Computer Science 6461. Berlin, Heidelberg: Springer, 2010. 1-15

8 ZOU L, LV J, WANG S, et al. Verifying Chinese train control system under a combined scenario by theorem proving. In: Proc. of VSTTE 2013, Lecture Notes in Computer Science 8164. Berlin, Heidelberg: Springer, 2013. 262-280

9 MATTSSON S E, ELMQVIST H, OTTER M. Physical system modeling with Modelica. Control Engineering Practice, 1998, 6(4) : 501-510

10 BERKENKÖTTER K, BISANZ S, HANNEMANN U, PELESKA J. The HybridUML profile for UML 2.0. International Journal on Software Tools for Technology Transfer, 2006, 8(2): 167-176

11 ALUR R, DILL D. A theory of timed automata. Theoretical Computer Science, 1994, 126(2): 183-235

12 HENZINGER T. The theory of hybrid automata. In: Proc. of LICS 1996. Washington: IEEE Computer Society, 1996. 278-292

13 BERRY G, GONTHIER G. The Esterel synchronous programming language: design, semantics, implementation. Science of Computer Programming, 1992, 19(2): 87-152

14 BERRY G. Synchronous design and verification of critical embedded systems using SCADE and Esterel. In: Proc. of FMICS 2007, Lecture Notes in Computer Science 4916. Berlin, Heidelberg: Springer, 2007. 2

15 PLATZER A, QUESEL J-D. European train control system: A case study in formal verification. In: Proc. of ICFEM 2009, Lecture Notes in Computer Science 5688. Berlin, Heidelberg: Springer, 2009. 246-265

16 TANG T and GAO C. Analysis of ETCS and CTCS. Electric Drive for Locomotives ,2005, 6:1-3 [唐涛, 郜春海. ETCS系统分析及CTCS的研究. 机车电传动, 2005, 6:1-3]

17 XU T, ZHAO H and TANG T. Reliability analysis of wireless communication of ETCS using colored Petri Net. Railway Science, 2008, 30(1):38-42 [徐田华, 赵红礼, 唐涛. 基于有色Petri网的ETCS无线通信可靠性分析. 铁道学报, 2008, 30(1):38 – 42]

18 ZHANG S. CTCS-3 Technology Specification. China Railway Publishing House. 2008 [张曙光. CTCS-3级列控系统总体技术方案. 中国铁道出版社, 2008]
19 ZHOU C, HOARE C A R, RAVN A P. A calculus of durations. Information Processing Letters, 1991, 40(5): 269-276
20 ZHOU C, HANSEN M. Duration Calculus: A formal approach to real-time systems. Springer, 2004

Formal anlysis and verification of Chinese Train Control System

GUO DanQing[1], LV JiDong[2], WANG ShuLing[2], TANG Tao[1], ZHAN NaiJun[1*], ZHOU DaTian[2], ZOU Liang[1]

1 *State Key Lab. of Computer Science, Institute of Software, CAS, Beijing* 100190*, China;*
2 *Beijing Jiaotong University, Beijing* 100044*, China;*

*E-mail: znj@ios.ac.cn

Abstract It is absolutely necessary to guarantee the correctness of high-speed train control systems by formal verification, as they are safety-critical. However, it is hard, even infeasible, to achieve the goal in practice, as these systems become more and more complicated. On the other hand, it is more convenient to model a complicated system in a graphical way. In particular, a graphical model is fairly intuitive, which has been used widely in industry. To improve the reliability of a high-speed train control system, constructing a graphical model for the system and then detecting its bugs by simulation sounds very effective. In this paper, we first show how to use Simulink/Stateflow to build graphical models for different combined scenarios of Chinese Train Control System (CTCS), in which mode conversion and level upgrade take place simultaneously. This modelling approach can be easily adapted to model other scenarios in CTCS by simply modifying the corresponding parameters. Then, we analyze these graphical models by simulation and find that under some circumstances the train will stop abnormally. Finally, in order to avoid the inherent incompleteness of simulation, we show how to translate these graphical models into formal models given by HCSP, which is a formal modelling language for hybrid systems by extending CSP, and subsequently we formally prove that abnormal stop could happen in any cases in one of these combined scenarios as an example. Formal verification of Simulink/Stateflow diagrams complements simulation, which improves the reliability of a system to be developed.

Keywords CTCS (Chinese Train Control System), Simulink/Stateflow, simulation, mode transition, level change, formal verification