软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2020,31(5):1374−1391 [doi: 10.13328/j.cnki.jos.005952]

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

区域控制器的安全需求建模与自动验证*

刘筱珊[1], 袁正恒[1], 陈小红[1], 陈铭松[1], 刘 静[1], 周庭梁[2]

[1](上海市高可信计算重点实验室(华东师范大学),上海 200062)

[2](卡斯柯信号有限公司,上海 200071)

通讯作者: 陈小红, E-mail: xhchen@sei.ecnu.edu.cn

摘 要: 轨道交通区域控制器是我国轨道交通信号系统选型的主流制式——基于通信的列车控制系统的核心子系统,其突出的安全性使得安全需求的形式化验证成为一个非常重要的问题.但是区域控制器自身的复杂性以及领域知识的繁杂难以掌握,使得形式化方法很难应用到安全需求的验证中去.针对这些问题,提出一种安全需求的自动验证方法,使用半形式化的问题框架方法来建模和分解安全需求,根据需求模型自动生成安全需求的验证模型和验证性质,在此基础上自动生成验证模型的 Scade 语言实现,并通过 Design Verifier 验证器对需求进行组合验证.最后,使用某个实际案例区域控制器的一个子问题 CAL_EOA 进行了研究,实验结果证明了该方法的可行性与有效性.它能够自动地将安全需求模型进行组合验证,改善了验证的效率.

关键词: 区域控制器;安全需求建模;自动验证;需求分解;组合验证

中图法分类号: TP311

中文引用格式: 刘筱珊,袁正恒,陈小红,陈铭松,刘静,周庭梁.区域控制器的安全需求建模与自动验证.软件学报,2020,31(5):1374−1391. http://www.jos.org.cn/1000-9825/5952.htm

英文引用格式: Liu XS, Yuan ZH, Chen XH, Chen MS, Liu J, Zhou TL. Safety requirements modeling and automatic verification for zone controllers. Ruan Jian Xue Bao/Journal of Software, 2020,31(5):1374−1391 (in Chinese). http://www.jos.org.cn/1000-9825/5952.htm

Safety Requirements Modeling and Automatic Verification for Zone Controllers

LIU Xiao-Shan[1], YUAN Zheng-Heng[1], CHEN Xiao-Hong[1], CHEN Ming-Song[1], LIU Jing[1], ZHOU Ting-Liang[2]

[1](Shanghai Key Laboratory of Trustworthy Computing (East China Normal University), Shanghai 200062, China)

[2](CASCO Signal Co., Ltd., Shanghai 200071, China)

Abstract: The rail transit zone controller is a core sub-system of the communication-based train control system, which is the mainstream choice of China's rail transit systems. Its outstanding safety makes formal verification of safety requirements a very important issue. However, the complexity of ZC itself and the rail transit domain knowledge make it difficult to apply formal methods to the verification of safety requirements. To solve these problems, this study proposes an automated verification approach for safety requirements. It models and decomposes safety requirements using a semi-formal Problem Frames approach, automatically generates verification model and verification properties, implements the verification problem with Scade automatically, and finally performs formal verification with a model checker Design Verifier. Finally, a sub-problem of zone controller, CAL_EOA from real case is studied. The

* 基金项目: 国家重点研发计划(2018YFB2101300); 国家自然科学基金(61332008, 61872147, 61572195, 61802251); 上海市经济和信息化委员会专项资金(160306)

Foundation item: National Key Research and Development Program of China (2018YFB2101300); National Natural Science Foundation of China (61332008, 61872147, 61572195, 61802251); Special Fund of Shanghai Municipal Commission of Economy and Informatization (160306)

本文由"系统软件构造与验证技术"专题特约编辑赵永望副教授、刘杨教授、王戟教授推荐.

收稿时间: 2019-09-01; 修改时间: 2019-10-24; 采用时间: 2019-12-24; jos 在线出版时间: 2020-04-07

experiment results show the feasibility and effectiveness of the proposed approach. The safety requirements are automatically decomposed and compositionally verified, thus greatly improving the verification efficiency.

Key words: zone controller; safety requirement modeling; automatic verification; requirement decomposition; composition verfication

基于通信的列车控制系统(communication based train control,简称 CBTC)是用来保证轨道交通安全、高效运行的控制系统.它打破了传统的轨道交通领域的固有限制,可以在更精确的运行间隔中追踪列车,已经成为我国轨道交通信号系统选型的主流制式[1].区域控制器(zone controller,简称 ZC)是 CBTC 系统的重要组成部分,负责控制多个车站之间的一部分轨道上的列车及所涉及到的设备,其核心功能是计算所控制的所有列车的移动权限(MA),并将计算结果发送给相应的列车.在我们与卡斯柯信号有限公司的合作项目中,ZC 系统还具有其他功能,例如计算列车的安全位置、对区域内的列车进行分类以及更新轨道占用状态等.

作为 CBTC 的地面核心子系统,ZC 的安全性直接关系到整个系统的成败.据统计,在安全攸关系统的安全类问题中,软件需求缺陷导致的问题已占到 60%~80%[2],需求类错误占到软件缺陷和错误的 70%[3],美军四代战机 F-22 项目办公室认为,这一数字甚至高达 85%[3].所以,对安全攸关场景下的 ZC 系统进行安全需求的建模与验证至关重要,是必须解决的问题.常见的需求验证包括需求的正确性、一致性和可满足性[4].本文关注需求的可满足性,特指验证规约是否满足需求.在轨道交通的国际标准 EN50128[5]、EN50129[6]中,强烈推荐在软件开发生命周期的每部分使用形式化方法.因此,有必要对 ZC 的安全需求进行形式化建模与验证.但在建模与验证过程中,需要解决以下问题.

(1) ZC 本身的复杂度:根据文献[7],ZC 可以分为大约 20 个子系统,在实现中可以进一步分为更多的子模块,这些模块与不同的设备交互.所谓设备,我们指的是与 ZC(子)系统交互的外部实体,不仅包括信号灯等物理世界中的设备,而且包括计算机联锁系统等现有软件,以及虚拟列车保护(VTP)等虚拟节点.根据 IEEE1474.1 标准[8],ZC 可以与 1 000 多个这样的设备交互.由于设备在形式化的验证中由一组变量表示,一个设备可能有数千个可能的值.所有这些值使得系统的模型过于庞大,无法由现有的验证器进行计算,导致了状态爆炸问题的出现.

(2) 形式化方法难以应用:形式化方法是使用一套明确定义的数学概念的符号来书写,有严格的语法规范和确定的语义定义,对数学基础的要求较高,因此其学习成本较高,非专业人士难以应用.在轨道交通领域,涉及很多的领域知识,对这类系统进行形式化,不仅要求分析员有着深厚的形式化方法的知识,也要熟悉领域知识.但通常来说,领域专家不了解形式化,而形式化专家缺乏充分的领域知识,很难对 ZC 这类系统的安全需求进行直接的形式化建模与验证.

在现有的安全需求验证工作中,主要有两种方式:一种是将安全需求直接建模为形式化语言并进行形式化验证,例如文献[9]中使用 ANSI/ISO-C 规范语言(ACSL)将其形式化并实现自动化验证,文献[10]中使用 SCADE 进行形式化的建模与验证,文献[11]将 Petri 网与基于域的上下文推理相结合来实现形式化建模并最终实现验证等,但是这些工作都无法避免非专业人士难以使用形式化方法的难题;另一种就是基于半形式化需求模型的验证,例如文献[12]中使用 UML 进行建模并用 BACH 进行验证,文献[13]中使用 SysML/KAOS 进行建模并用 Rodin 进行验证,文献[14]中使用消息顺序图进行建模并用 UPPAAL 进行验证等.这些半形式化模型最终都将转换为形式化语义模型进行验证.这类方法使用 UML 等图形化语言,降低了形式化方法的使用成本,但是并不能直接应用到 ZC 系统中,它也没有处理验证问题的复杂性.总之,现有的工作并不能解决全部解决上述问题.

因此,本文提出一种安全需求的自动验证方法,使用半形式化的问题框架方法(PF)[15,16]来建模安全需求,根据需求模型自动生成安全需求验证问题模型,并根据需求模型和验证问题模型中的每个部分自动生成 SCADE[17]工具中的 Scade 语言模型进行组合验证,有效降低验证复杂度.选择 PF 有两个原因:(1) PF 是一种基于环境建模的需求工程方法,它将需求映射为环境(如 ZC 中的设备)中预期的变化,特别适合建模 ZC 这类系统[18];(2) PF 使用投影进行问题分解,能够有效降低需求问题的复杂度,我们前期的工作[19]就提出了一种基于情景的自动投影方法.选择 SCADE 工具是因为它是一种典型的模型驱动开发支持工具,常用于支持安全关键系统开发的整个过程,涵盖了从建模、仿真到验证.它通过了轨道交通标准 EN 50128 认证,达到 SIL 3/4[5,6],能够减少开

发过程中的成本、风险并能缩短验证所需的时间,在轨道交通领域被广泛应用.

本文首先利用 PF(problem frames)中的问题图和情景图建模 ZC 的安全需求问题,并利用文献[19]中的自动投影方法将安全需求分解为若干个较小的子问题,根据每个子问题的需求模型自动生成安全需求的验证问题模型;然后,根据需求模型和验证问题模型自动生成 Scade 模型并进行需求可满足性的组合验证.围绕上述工作,本文第 1 节介绍本文涉及到的 PF 和 SCADE 验证方法.第 2 节对区域控制器(ZC)的安全需求使用 PF 进行建模与分解,在一定程度上缓解状态空间爆炸的问题.第 3 节提出安全需求验证问题模型自动生成并转化为 Scade 模型的方法,并由此生成 ZC 的验证模型.第 4 节以 ZC 的核心功能之一的移动授权终点计算子系统 CAL_EOA 为例,将本文所提出的方法应用到实际的工业案例中,实现对安全需求的建模、分解与自动验证过程.第 5 节介绍相关工作,最后第 6 节总结全文并提出下一步工作.

1 预备知识

1.1 问题框架方法

问题框架方法(PF)是 Jackson 提出的一种以环境为视角的需求工程方法[15,16].它强调对与软件系统相交互的现实世界(即环境)进行刻画,将需求的含义指称到对现实世界相关领域的描述上.其建模的基本概念是现实世界的“问题领域”,以及软件系统与领域的“交互”.它使用问题图[16]来对需求进行静态的描述,使用情景图[19]刻画如何动态实现需求.

在问题图中,用领域(domain)、需求(requirement)和交互(interaction)来对问题进行描述,其中,领域可以被分为两类:机器领域(machine domain)和问题领域(problem domain).机器领域与问题领域之间共享的现象称为交互.交互分为两种类型:行为交互为机器领域和问题领域之间的接口交互(interface),期望交互为问题领域和需求之间的引用(reference)以及需求与问题领域之间的约束引用(constraint).根据文献[16],需求问题 P 可以定义为一个四元组:$P=\langle M,E,IS,R\rangle$,其中,机器领域 M 表示要开发的软件;环境领域 E 是所有与 M 交互的物理实体的集合;IS 为交互(interaction)集;R 是一组需求,通常用自然语言描述.而交互可以定义为一个三元组:$Interaction=\langle Init,Recv,Phe\rangle$,$Init$ 为交互的发送方,$Recv$ 为交互的接收方,Phe 为共享的现象.具体细节详见文献[15].

情景图是交互流,表示交互之间的关系.其基本表示采用活动图的形式.每个情景图都可以从 3 部分来理解:行为交互部分、期望交互部分以及行为交互和期望交互的使能关系部分.在行为交互部分中,用活动图来对行为交互的变化流程进行描述,用行为序关系(*BehOrder*)来描述行为交互之间的先后关系;在期望交互部分中,同样使用活动图来对期望交互的变化流程进行描述,用期望序关系(*ExpOrder*)来描述期望交互之间的先后关系;而在使能部分中,可以使用行为交互和期望交互之间的连接来表示交互之间的使能关系,包括同步关系(*Syn*)、行为使能关系(*BehEna*)和期望使能关系(*ExpEna*).根据文献[19],情景图可以定义为一个三元组 $Scenario=\langle NodeSet,AssSet,FlowSet\rangle$,其中,$NodeSet=IntSet\cup CtrlSet$ 表示节点集合,可以分为两类:交互节点(*IntNode*)和控制节点(*CtrlNode*).其中,交互节点包括行为交互(*BehIntNode*)和期望交互(*ExpIntNode*);控制节点可分为以下 5 种:开始节点(*StartNode*)、结束节点(*EndNode*)、决策节点(*DecisionNode*)、合并节点(*MergeNode*)和分支节点(*BranchNode*).$AssSet=\{AssType(int_i,int_j)|int_i,int_j\in IntSet\}$表示交互之间的关系,包括以下 5 种类型:行为使能关系 $BehInt\rightarrow ExpInt$(*BehEna*)、行为序关系 $BehInt\rightarrow BehInt$(*BehOrd*)、同步关系 $BehInt\leftrightarrow ExpInt$(*Syn*)、期望使能关系 $ExpInt\rightarrow BehInt$(*ExpEna*)和期望序关系 $ExpInt\rightarrow ExpInt$(*ExpOrd*).$FlowSet=\{CtrlFlow(ctrl_i,node_j)|ctrl_i\in CtrlSet, node_j\in NodeSet\}$表示控制节点和其他节点之间的转换关系.具体细节详见文献[19].

1.2 SCADE Suite工具

SCADE Suite[17]是 ANSYS 公司嵌入式软件家族中的一个产品线.它整合了形式化建模语言 Scade,内置了由 Prover 公司开发的模型检验器 Design Verifier,可以对 Scade 模型进行形式化验证.它是一个对关键应用进行综合设计的环境,可以进行模型驱动设计,被用于设计关键软件,例如飞行控制系统、自动驾驶系统、轨道联锁系统、自动列车操控系统、紧急刹车系统、超速保护系统、列车空位检测系统以及其他众多航空航天、轨道交通、自动化领域的工业应用.除此之外,它还支持仿真、验证和代码生成.

在 Scade 中,模型的基本结构是操作符,不同的操作符之间支持并行或嵌套的关系,每个操作符需要定义它的名称、输入变量、输出变量和本地变量.不同的操作符之间通过实体线连接,表示数据的传输路径.也就是说,一根实线两端分别连接一个操作符的输出变量和另一个操作符的输入变量.在对模型进行验证时,在被验证模型之外,需要增加一个操作符观察者(observer),用来描述所需要验证的性质.性质的描述语法和建模一样,仅要求输出结果为一个布尔变量.

2 区域控制器的安全需求建模与分解

ZC 是 CBTC 系统的地面核心子系统,其核心功能是用来计算列车的移动权限(MA),对于 CBTC 系统起着至关重要的作用.根据 PF 中的问题定义,首先获取待开发的软件 ZC 为机器 *M*.ZC 子系统通过车-地设备的无线通信实现车-地双向信息交互,从车载计算机(VOBC)、自动列车监控(ATS)和计算机联锁系统(CI)以及轨道沿线的许多传感器接收信息,由此实现为列车分配移动授权的功能.ZC 系统通过与这些外部实体进行交互,以此获取相应的位置信息,从而完成对列车移动权限的分配.每个实体我们称为设备,这些设备不仅包括很多物理世界中的物理实体,例如信号机、列车、轨道、传感器等,还包括与 ZC 交互的其他现有系统,比如 VOBC、ATS 等,甚至包括虚拟列车保护等的逻辑概念.这些设备组成 ZC 的环境 *E*,其中,设备的每个状态都可以用更多的变量来表示.例如,在轨道上有一种叫做道岔(point)的设备需要 5 个变量来描述:一个状态变量表示它的状态是正常、反向或未知的值之一;另 4 个变量,即 *nextidx*_1~*nextidx*_4,表示它旁边的块的 ID.因此,*E* 的形式化描述由 ZC 环境中的这些设备定义,每个设备由一组变量定义.由于在有限的空间中存在太多的设备(超过 1 000 多个),因此使用 D_i 来对设备进行表示.*E* 可以形式化地表示如下.

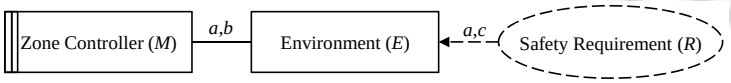
定义 1(环境 *E*). 是所有与 *M* 交互的设备 D_i 的集合,而每个设备 D_i 是变量 v_{ij} 的集合,即

$$E=\{D_i|1\leqslant i\leqslant n\},\ D_i=\{v_{ij}|1\leqslant i\leqslant n,1\leqslant j\leqslant m\}.$$

根据发送者不同,*M* 和 *E* 之间的相互作用可分为两组:*a*,*c* 和 *b*(遵循 PF 的约定).*a*,*c* 中的交互是由环境 *E* 发送的现象,而 *b* 中的交互是由机器 *M* 发送的现象.*a*,*b* 和 *c* 中的交互是 *E* 中变量 $v_{ij}(1\leqslant i\leqslant n,1\leqslant j\leqslant m)$的函数表示:

$$f_i(V_{11},\ldots,V_{nm}),\ g_j(V_{11},\ldots,V_{nm})\text{和}\ h_k(V_{11},\ldots,V_{nm}).$$

R 是一组安全需求.本文中的安全需求采用文献[18]中的定义,为机器应满足或者不违反的条件,以确保人员安全.安全需求的一个例子是“一列火车决不能撞到另一列车”.一般来说,*R* 通常是用自然语言编写的,可以分解成若干个子需求.例如,上面的例子可以分解为 2 个子需求:(1) ZC 必须确保 MA 中没有另一列车;(2) VOBC 必须确保列车速度不超过速度曲线.在大多数情况下,子属性可以在领域标准中找到,如 IEEE1474.3[20].根据以上描述,我们得到 ZC 的安全需求问题,如图 1 所示.



a:{$int_1,\ldots,int_m$},b:{$int_{m+1},\ldots,int_n$},c:{$int_{n+1},\ldots,int_p$};int_i=<$E,M,f_i(v_{11},\ldots,v_{nm})$>$(1\leqslant i\leqslant m)$,
int_j=<$M,E,g_j(v_{11},\ldots,v_{nm})$>$(m+1\leqslant j\leqslant n)$,$int_k$=<$E,M,h_k(v_{11},\ldots,v_{nm})$>,$n+1\leqslant k\leqslant p$

Fig.1 A schematic diagram of the problem diagram in ZC

图 1 ZC 问题图的示意图

在问题图的基础上,定义 ZC 里面的情景来表示每个 *R* 的子需求是如何实现的.ZC 和设备(如列车、相应传感器等)之间紧密结合,在接收到物理实体发送的信息后,对列车前方轨道的可运行情况进行计算,并将计算结果返回列车,以便对列车进行控制.以 ZC 的核心功能移动授权的计算与分配来举例,在单个计算周期内,ZC 需与多个设备(如列车、轨道、信号机、其他 ZC 等)进行交互.一般来说,列车首先向 ZC 发起移动授权请求,然后 ZC 接受包括列车在内的多个设备的相关信息,根据这些信息进行移动授权的计算,并把计算的授权终点坐标结合其他相关信息组成移动授权报告,再发送给列车.

根据上述的描述,可以大概描述每个 *R* 的子需求对应的情景图.在行为交互部分中,外部设备将环境信息发送给 ZC,在对列车前方轨道的安全情况进行计算后,将结果发送给控制设备.在期望交互的部分中,期望外部设

备将环境信息发送给 ZC,ZC 依据需求再从环境中获取其他有效信息,然后控制设备的状态发生变化.在交互使能的部分中,当外部设备将环境信息发送给 ZC 时,期望交互部分中同时期望这一现象发生,因此它们之间存在同步关系;期望 ZC 从环境中感知其他有效信息从而触发接下来的行为交互,因此存在期望使能关系;当 ZC 将计算结果发送给控制设备后,期望控制设备的状态发生改变,因此行为交互和期望交互之间存在行为使能关系.据此,得到 ZC 的子需求情景图,如图 2 所示.

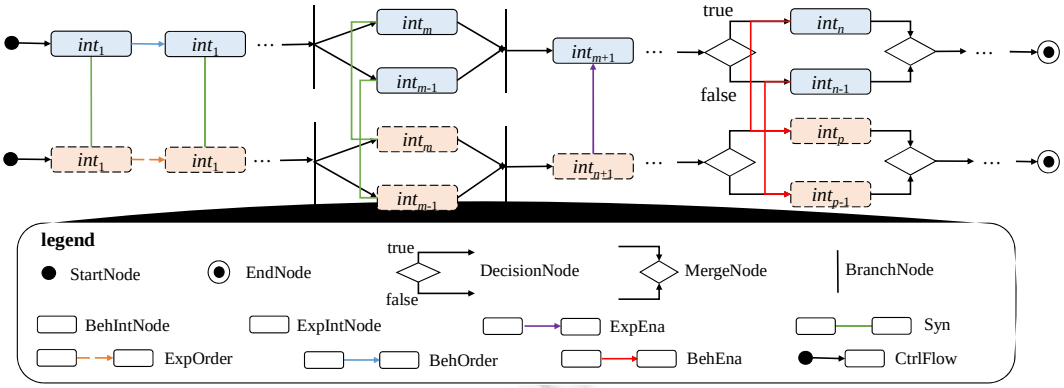


Fig.2 A schematic diagram of the scenario graph in ZC's sub-requirements

图 2 ZC 的子需求情景图示意

在图 1 问题图和图 2 类似(假设)的 n 个情景图的基础上,根据文献[19],对问题 ZC 使用需求分解方法可以得到多个子需求问题,即 $P=\{P_1,\ldots,P_n\}$,其中,$P_i=\langle M_i,E_i,IS_i,R_i\rangle(1\leqslant i\leqslant n)$.详情这里不再阐述.

3 区域控制器的安全需求验证

3.1 验证模型的生成

本文采用 SCADE 来完成对区域控制器的安全需求自动验证.在自动生成 Scade 模型之前,需要先生成需求的验证模型.本文把需要建模和实现的机器称为验证机器(verification machine,简称 VM).*VM* 通过观察机器 *M* 的输入变量和输出变量,即 *M* 和环境 *E* 的交互,来判断机器 *M* 是否在所有可达状态上满足安全需求,并且依此生成验证报告(verification reprot,简称 VR),值得注意的是,*VR* 也是一个问题域,在问题框架方法中,将要构建的数据存储可以建模为一种称为"设计域"的问题域类型,由问题图中带有一条竖线的矩形表示.因此,验证系统的环境 E'由机器 *M*、环境 *E*、验证报告 *VR* 构成,即 $E'=E\cup\{M\}\cup\{VR\}$.

环境 E'之间的交互关系如下:首先,根据图 1,机器 *M* 和环境 *E* 之间的交互集为 *a*,*b* 和 *c*.从机器 *M* 的角度来看,*a*,*c* 是输入变量,*b* 是输出变量.验证机器 *VM* 也需要观察 *a*,*b* 和 *c* 来判断需求 *R* 是否得到满足.因此,环境 *E* 和验证机器 *VM* 之间的交互 a',c'在内容上和 *a*,*c* 相同,验证机器 *VM* 和机器 *M* 之间的交互 b'在内容上和 *b* 相同.a',b',c'和 *a*,*b*,*c* 的区别在于接收方不同.这样一来,在需求的验证模型中就可以不包括 *a*,*b* 和 *c*.而对于验证机器 *VM* 和验证报告 *VR* 之间的交互,可以用一个布尔变量 *bValid* 和一个字符串变量 *cExample* 来表示,分别代表验证的结果,以及当验证失败时需要提供的反例.

综上,需求的验证模型包含了验证机器 *VM*、验证环境 E'以及交互 a',b',c'和 *d*,其中,*VM* 是用来验证 *M* 的验证系统;E'是 *VM* 的运行环境,即 *E*,*M*,*VR* 的并集,*VR* 是该验证问题的验证报告,包括验证结果 *bValid* 和失败时反例 *cExample*;$IS'=a'\cup b'\cup c'\cup d$ 是该验证问题和 E'之间的交互集合,其中,$d=\{int_{2p+1}=\langle VM,VR,bValid\rangle\}\cup\{int_{2p+2}=\langle VM,VR,cExample\rangle\}$.由此,可以得到需求的验证模型的静态视图,如图 3 所示.

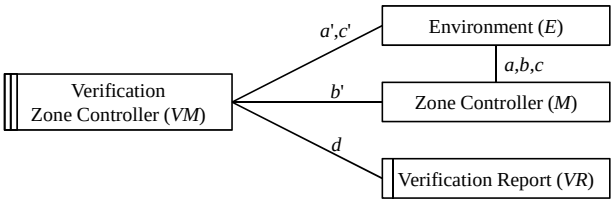


Fig.3 Static verification model of the requirements

图 3 需求的验证模型的静态视图

根据上述描述,我们设计了需求验证模型的自动构造规则,见表 1.该转换规则是根据需求问题 *P* 自动生成为需求的验证模型,从 *P* 中读取机器 *M* 来构建验证机器 *VM*;根据需求的验证结果来构建验证报告 *VR*,*E*′为环境 *E*、机器 *M* 和验证报告 *VR* 的并集;根据 *IS* 中交互来构建 *IS*′中新的交互:交互的发送方为原来的交互发送方,交互的接收方为验证机器 *VM*,交互的内容与原来的交互内容相同;另外,新建与验证报告之间的交互:交互的发送方为验证机器(*VM*),接收方为验证报告(*VR*),内容包括验证结果(*bValid*)和验证失败时需要提供的反例(*cExample*).

Table 1 mapping rules from requirement problem to verification model

表 1 从需求问题到需求验证模型静态视图的构造规则

需求问题 *P*	需求问题的验证模型
M	*VM*: Verify *M*
E	*E*′=*E*∪{*M*}∪{*VR*} *VR*:验证报告
IS	*IS*′=*IS*′∪{*d*}; **for** $int_i \in IS$ **then** $int_{i+p}.Init=int_i.Init$; $int_{i+p}.Recv=VM$; $int_{i+p}.Info=int_i.Info$; *IS*′=*IS*′∪{$int_i$}; **end for** *d* ={$int_{2p+1}$=⟨*VM*,*VR*,*bValid*⟩}∪{$int_{2p+2}$=⟨*VM*,*VR*,*cExample*⟩};

基于静态视图,要研究其行为交互过程,以此构建其动态视图.根据图 2 中机器 *M* 和设备 *E* 交互的情况,其验证过程可以设计为:首先,验证机器 *VM* 观察到外部设备发送环境信息;然后,*VM* 会接收到机器 *M* 发送的交互信息;接下来,*VM* 能够观察到外部设备发送其状态的改变情况;最后,根据获得的信息,*VM* 判断需求 *R* 的满足情况,将验证结果和验证失败时的反例发送给验证报告 *VR*.具体过程如图 4 的上半部分所示.

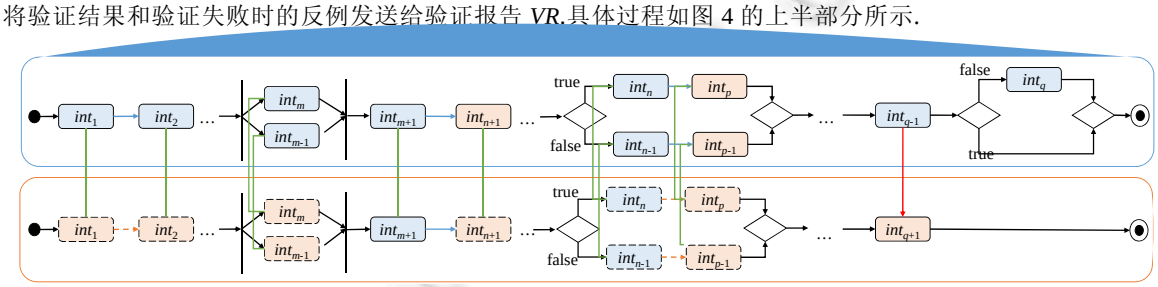


Fig.4 Scenario graph of the verification system

图 4 验证系统的情景图

3.2 验证性质的生成

待验证的性质 R'实际上就是要验证机器 M 是否满足需求 R.按照问题框架的表示,实际上就是要增加验证问题的需求、需求引用和约束.其中,验证问题的需求就是 R',下面按 E'领域来考虑需求对其现象的引用和约束.

首先,对于环境 E,由于 E 与 VM 的交互为 a',c',与 M 的交互为 a,b 和 c,其中,a',c'来自 a,c,a,c 中的交互是由环境 E 发送的现象,这种现象是要观察的,因此验证需求 R'要引用交互 a',c'中的现象,在 R'和 E 之间存在需求引用 a',c'.对于机器 M 来说,它与 VM 的交互为 b',与 E 的交互为 a,b 和 c,其中,交互 b'来自于交互 b,其在内容上和 b 相同且由机器 M 发送到验证机器 VM,R'要通过验证机器 VM 来验证机器 M 是否满足需求 R,因此需要引用 b'.对于验证报告 VR 而言,其与 VM 之间的交互 d 是 VM 发送给 VR 的需求验证结果和验证失败时的反例,R'期望验证报告的报告效果(*reportEfforts*)发生改变,因此,R'和 VR 之间存在约束 $e=\{int_{2p+3}=\langle VR,VM,reportEfforts\rangle\}$,因此,$IS'=IS'\cup e$.综上,我们在图 3 的基础上,增加 R'、需求约束与需求引用,得到验证系统的问题图来对其进行静态描述,如图 5 所示.

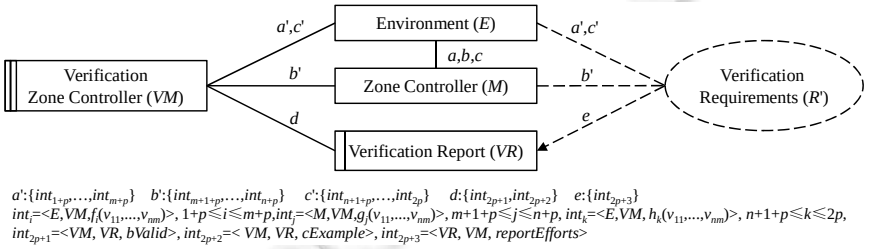


Fig.5 Problem diagram of verification system

图 5 验证系统的问题图

基于验证系统的问题图,根据图 2 中需求问题的引用和约束的先后顺序关系,设计验证系统的需求约束和约束引用的先后顺序关系,可以得到相应的动态视图,如图 4 的下半部分所示.首先,R'期望 VM 能观察到外部设备发送的环境信息;然后,VM 接收到 M 发送的交互信息;接下来,期望 VM 能观察到外部设备发送的其状态的变化;然后,期望验证报告 VR 的报告效果发生变化.

3.3 验证的Scade实现

对于一个验证问题 VP_i,需要使用 SCADE Suite 对其进行建模并用内嵌的 Design Verifier 进行验证.根据 Scade 语言特点,就是要设计图 5 验证问题中 E、M、VR 和 VM 以及验证性质 R'.这些设计中必须包含其领域特性,这些领域特性更多包含在图 2 的 ZC 系统的情景图中,只有 VR 的性质是体现在图 4 的验证系统的情景图中.因此,本文从图 5 验证系统的问题图和图 2 系统的情景图出发,设计了自动生成 Scade 模型的规则,如图 6 所示.

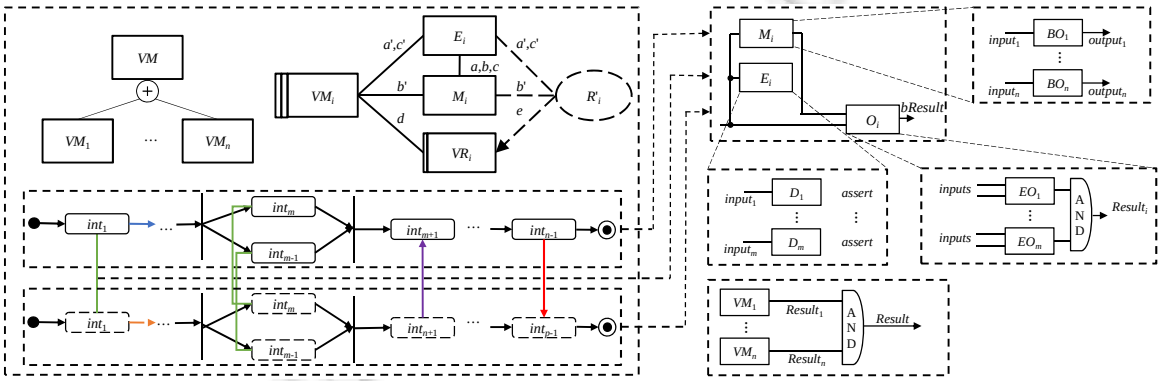


Fig.6 SCADE model conversion diagram for verification problem

图 6 验证问题 SCADE 模型转换示意图

在 Scade 模型中,分为两个层级:首先是顶层操作符,主要描述问题图中不同领域之间的关系,即 *M*,*E*,*VR* 之间的连接方式,对应在 Scade 中分别为操作符 *M*、操作符 *E* 和操作符 *O*.操作符 *O*,即观察者,被用来描述所需要验证的性质.对于该验证问题中操作符 *O* 的构建,其实就是要判断需求的期望交互是否能发生.根据情景图对于期望交互的描述,操作符 *O* 需要判断情景图中所有的期望交互是否被满足.值得注意的是,在问题图中,并没有描述交互之间的先后顺序.根据情景图以及 Scade 的语义模型,可以得到如下所述的每个模块的输入输出集合:操作符 *M* 的输入为所有 *E* 发出的、*M* 接受的变量集合,输出为 *M* 发出的、*E* 接受的变量集合.操作符 *E* 的输入为操作符 *M* 的输入变量集合,并以 Scade 语言中断言的形式将操作符 *E* 对这些变量的影响反馈给操作符 *M*,即没有显示输出.在实际的建模过程中,这部分变量往往以感应器(sensor)的方式进行定义.操作符 *O* 的输入为操作符 *M* 的输入变量和输出变量,输出为单个布尔变量,用来表示 *M* 是否满足 *R*′.通过实线可以将这 3 个操作符进行连接.具体的设计如算法 1 所示.

- *M* 根据行为交互进行设计,也就是说,为每个交互设计一个操作符 BO_i,该操作符的输入输出由情景图中对于该交互的建模得到.同时,不同 BO_i 之间仅有并行关系,而没有实体线对它们进行连接.在对 BO_i 进行进一步的设计时,也遵循行为交互所包含的信息,设计其数据流模型,完成 BO_i 操作符的实现.具体细节见算法 1 的第 3 行~第 14 行.
- *E* 根据交互使能进行设计,首先,根据问题图中建模的问题领域,对每个问题领域创建一个操作符 D_i,该操作符的输入由问题图中该领域相关的交互来设计;然后,在每个操作符 D_i 中,根据交互使能对其进行进一步设计.具体来说,在问题图和情景图中,由 *E* 发起的交互的变量,在 Scade 模型中设定为感应器(sensor)类型的变量,通过对该变量进行断言(assert),把问题领域由交互产生的影响反馈给 *M*.变量来自单个问题领域的断言在具体 D_i 中进行描述;变量来自多个问题领域的断言在 *E* 中进行描述.具体细节见算法 1 的第 15 行~第 26 行.
- *O* 根据期望交互进行设计,根据情景图中建模的期望交互,在 *O* 中设计相对应的操作符 EO_i,对每个期望交互进行设计建模.操作符 EO_i 的输入由相对应的交互而来,输出则是一个布尔变量,用来表示 *M* 是否满足该期望交互.并且,所有 EO_i 的输出由 and 操作符连接,作为 *O* 的输出变量.此外,当存在多个 *R*′ 时,会建立多个 *O*,可以把这些 *O* 的输出变量通过 and 操作符连接,从而同时验证多个属性.具体细节见算法 1 的第 27 行~第 35 行.

算法 1. Scade 模型生成算法.

Input:验证系统问题 $VP=\langle VM,E',IS',R'\rangle$;

系统情景图:$Scenario=\langle NodeSet,AssSet,FlowSet\rangle$,$NodeSet=\langle BehSet,ExpSet\rangle$.

Output:Scade 模型 $N=\{operators\}$.

1: **Begin**;
2: N=NULL;
3: 构建验证系统操作符 DV,$N=N\cup\{DV\}$:
4: **for** 问题图和情景图中所有交互变量 *Variable*
5: $DV.input=\{int|int\in NodeSet\}\cup DV.input$; //*DV* 的输入变量为问题图和情景图中所有交互变量
6: $DV.output=\{BooleanVariable\}$; //*DV* 的输出变量为一个布尔变量,用来表示 *M* 是否满足 *R*′
7: 构建操作符 M,$N=N\cup\{M\}$;
8: $M.input=\{variable|int=\langle *,M,variable\rangle,int\in BehSet\}$; //*M* 的输入为所有行为交互中 *M* 接受的交互变量集合
9: $M.output=\{variable|int=\langle M,*,variable\rangle,int\in BehSet\}$; //*M* 的输出变量为所有行为交互中 *M* 发出的交互变量集合
10: **for** (each $BehEna_i$ or $ReqEna_i\in NodeSet$)
11: 构建操作符 BO_i in *M*;

12: BO_i.*input* 为该使能中所涉及到的交互变量集合;
13: BO_i.*output* 的输出为该使能所需要发出的交互变量集合;
14: **Endfor**
15: 构建操作符 E,$N=N\cup\{E\}$
16: 定义 E 的输入变量集合:所有交互使能中涉及到的交互变量集合;
17: 定义 E 的输出变量集合:无;
18: **for** (each $domain\in E'$) //构建操作符 E 中的 D_i
19: 构建操作符 D_i,$E=E\cup D_i$
20: **for** (each $Ass\in AssSet$)
21: **if** (*Ass* 的变量来自单个 D_i)
22: 在 D_i 中创建表达式或操作符,描述 *Ass*;
23: **else** 在 E 中创建表达式或操作符,描述 *Ass*;
24: **endif**
25: **endfor**
26: **endfor**
27: 构建操作符 O,$N=N\cup\{O\}$;
28: 定义 O 的输入变量集合:M,E 的输入变量集合和输出变量集合的并集
29: 定义 O 的输出变量集合:一个布尔型变量 *Result*;
30: **for** (each $expBeh\in ExpSet$)
31: 在 O 中根据期望使能构建操作符 EO_i,$O=O\cup EO_i$
32: EO_i 的输入为$\{variable|int=\langle *,*,variable\rangle\}$
33: EO_i 的输出为布尔变量 *Result*
34: **endfor**
35: 把所有 EO_i 的输出变量用 AND 操作符连接,并连至 O 的输出变量 *Result*;
36: 把 M,E,O 拖入 *DV* 中;
37: 连接 *DV* 的输入变量至 M,E,O 的输入变量;
38: 连接 M 的输出变量至 E 的输入变量;
39: 连接 O 的输出变量至 *DV* 的输出变量;
40: **End**;

4 案例研究

本节用 ZC 的核心子系统 CAL_EOA 作为案例.限于篇幅,我们将案例做了一定的简化.CAL_EOA 负责计算并分配移动授权.移动授权(movement authority,简称 MA)指的是轨旁设备分配给列车对于该区域的行驶许可,即告诉车辆前方的轨道在多大程度上能够安全运行.计算过程如下:随着列车的移动,列车必须和轨旁设备保持连续的通信;在通信过程中,列车的位置信息和刹车曲线也在被连续地计算,并从列车发送到轨旁设备.根据这些信息,轨旁设备能够建立保护区域,确保其他列车不会进入.这个区域被称为移动授权区域.这个区域的终点被称为授权终点(end of authority,简称 EOA),列车在一个移动授权中被允许行驶到该点,但速度必须降为 0.

4.1 需求的建模和分解

CAL_EOA 就是我们要开发的软件,它涉及到如下设备:列车(train,简称 TR)、区块(block,简称 BL)、分支(branch,简称 BR)、区块布局(block layout,简称 LO)、分支布局(branch layout,简称 RO)、其他列车(other train,简称 OT)、信号机(signal,简称 SI)、区域控制器边界(ZC boundary,简称 ZB)、行驶方向(traffic direction,简称 TD)、缓冲区(buffer zone,简称 BZ)、重叠区(overlap,简称 OL)等.这些设备共同构成了 CAL_EOA 的环境,即

E={TR,BL,BR,LO,RO,OT,SI,ZB,TD,BZ,OL}.

在需求 R 方面,IEEE 标准 1474.3 的第 6.3 节中提到移动保护的限制和目标点确认(limit of movement protection and target point determination)[20].根据该节内容及相关文档(如 IEEE 标准 1474.1[8]的第 6.1.2 节),可以得到一条 SIL-4 安全级别(即最高安全级别)的功能需求 R_{CAL_EOA}:CAL_EOA 需要为注册在当前区域控制器下的列车计算移动授权终点,并将其发送给相应列车.根据移动授权不同的情况,可以分为 8 个子需求,见表 2.

Table 2 List of safety sub-requirements in CAL_EOA

表 2 CAL_EOA 中的安全子需求列表

序号	子需求描述
R_1	在列车行驶方向上搜索移动授权终点,在没找到合适的结果时遇到了轨道尽头,以该点作为移动授权终点
R_2	在列车行驶方向上搜索移动授权终点,在没找到合适的结果时遇到了其他列车尾部,以该点作为移动授权终点
R_3	在列车行驶方向上搜索移动授权终点,在没找到合适的结果时遇到了非受控道岔,以该点作为移动授权终点
R_4	在列车行驶方向上搜索移动授权终点,在没找到合适的结果时遇到了区域控制器边界,以该点作为移动授权终点
R_5	在列车行驶方向上搜索移动授权终点,在没找到合适的结果时遇到了非连续行驶方向,以该点作为移动授权终点
R_6	在列车行驶方向上搜索移动授权终点,在没找到合适的结果时遇到了不允许该列车通过的缓冲区,以该点为移动授权终点
R_7	在列车行驶方向上搜索移动授权终点,在没找到合适的结果时遇到了断链点,以该点为移动授权终点
R_8	在列车行驶方向上搜索移动授权终点,在没找到合适的结果时遇到了不允许该列车通过的重叠区,以该点为移动授权终点

对于需求 R_1,CAL_EOA 从 Train、Block、Branch、Block Layout 和 Branch Layout 中分别读取列车定位信息、Block 逻辑分布、Branch 逻辑分布、Block 实体分布以及 Branch 实体分布,然后从最近的 Block 开始,一个 Block 一个 Block 地寻找是否存在轨道尽头,如果找到,作为 EOA 发送.

类似地,需求 R_2~R_8 中,也都是从不同的设备集合中读取列车定位信息、Block 逻辑分布、Branch 逻辑分布等信息,从最近的 Block 开始,一个 Block 一个 Block 地寻找是否存在其他列车(R_2)、是否存在非受控道岔(R_3)、是否存在区域控制器边界(R_4)、是否存在非连续 TD(R_5)、是否存在缓冲区(R_6)、是否存在断链点(R_7)、是否存在重叠区(R_8),如果找到,作为 EOA 发送.

从上述描述中,我们可以得到交互集合 IS,IS={int_1,int_2,...,int_{48}},其中,{int_1,…,int_9}为 Train 发送给 CAL_EOA 的列车定位信息,{int_{10},int_{11}}为 CAL_EOA 发送给列车的 EOA 报文,{int_{12},…,int_{15}}为 Block 发送给 CAL_EOA 的 BLOCK 逻辑分布,{int_{16},…,int_{18}}为 Branch 发送给 CAL_EOA 的 Branch 逻辑分布,{int_{19},…,int_{22}}为 Block Layout 发送给 CAL_EOA 的 Block 实体分布,{int_{23},…,int_{25}}为 Branch Layout 发送给 CAL_EOA 的 Branch 实体分布,{int_{26},…,int_{34}}为 Other Train 发送给 CAL_EOA 的其他列车定位信息,{int_{35},…,int_{37}}为 Signal 发送给 CAL_EOA 的信号机位置和状态,{int_{38},int_{39}}为 ZC boundary 发送给 CAL_EOA 的区域控制器边界信息,{int_{40},…,int_{42}}为 Traffic Direction 发送给 CAL_EOA 的 Block 的可行驶方向信息,{int_{43},…,int_{45}}为 Buffer Zone 发送给 CAL_EOA 的 SI 后的缓冲区信息,{int_{46},int_{47}}为 Overlap 发送给 CAL_EOA 的重叠区信息,int_{48} 为 Train 发送给 CAL_EOA 的报文接受状态.上述交互 int_i($1 \leqslant i \leqslant 48$)的定义及变量的含义具体见表 3.

根据上述信息,可以得到 CAL_EOA 的问题图,如图 7 所示.

接下来,对每个子需求绘制情景图.以 R_1 为例,在行为交互部分中,Train、Block、Branch、Block Layout 和 Branch Layout 分别向 CAL_EOA 发送列车定位信息(int_1,…,int_9)、Block 逻辑分布(int_{12},…,int_{15})、Branch 逻辑分布(int_{16},…,int_{18})、Block 实体分布(int_{19},…,int_{22})以及 Branch 实体分布(int_{23},…,int_{25}),在接收到信息之后,CAL_EOA 将移动授权终点的计算结果发送给列车(int_{10},int_{11}).在期望交互部分中,Train、Block、Branch、Block Layout 和 Branch Layout 分别向 CAL_EOA 发送列车定位信息(int_1,…,int_9)、Block 逻辑分布(int_{12},…,int_{15})、Branch 逻辑分布(int_{16},…,int_{18})、Block 实体分布(int_{19},…,int_{22})以及 Branch 实体分布(int_{23},…,int_{25}),然后期望列车的消息状态变为 Received(int_{48}).在交互使能部分中,由于行为交互和期望交互都需要获取 Train、Block、Branch、Block Layout 和 Branch Layout 向 CAL_EOA 发送的信息,因此它们之间存在同步关系;当列车接收到 CAL_EOA 发送的结果报告后,消息接收状态变为 Received,因此,int_{11} 和 int_{48} 之间存在行为使能关系.

最终,得到一个情景图,如图 8 所示.

Table 3 Interactive list of problem diagrams for CAL_EOA and its validation model

表 3 CAL_EOA 及其验证模型的问题图的交互列表

交互	发起方	接收方	现象	含义
int_1 (int_{49})	*TR*	CAL_EOA(*VM*)	*THaPo*	最大列车车头的坐标
int_2 (int_{50})	*TR*	CAL_EOA(*VM*)	*THiPo*	最小列车车头的坐标
int_3 (int_{51})	*TR*	CAL_EOA(*VM*)	*TTaPo*	最大列车车尾的坐标
int_4 (int_{52})	*TR*	CAL_EOA(*VM*)	*TTiPo*	最小列车车尾的坐标
int_5 (int_{53})	*TR*	CAL_EOA(*VM*)	*VHaPo*	最大虚拟列车防护头的坐标
int_6 (int_{54})	*TR*	CAL_EOA(*VM*)	*VHiPo*	最小虚拟列车防护头的坐标
int_7 (int_{55})	*TR*	CAL_EOA(*VM*)	*VTaPo*	最大虚拟列车防护尾的坐标
int_8 (int_{56})	*TR*	CAL_EOA(*VM*)	*VTiPo*	最小虚拟列车防护尾的坐标
int_9 (int_{57})	*TR*	CAL_EOA(*VM*)	*TRLen*	列车的长度
int_{10} (int_{58})	CAL_EOA	TR(*VM*)	*EOATyp*	EOA 的类型
int_{11} (int_{59})	CAL_EOA	TR(*VM*)	*EOAPos*	EOA 的坐标
int_{12} (int_{60})	*BL*	CAL_EOA(*VM*)	*BLpID*	在 up 方向的下一个区块的编号
int_{13} (int_{61})	*BL*	CAL_EOA(*VM*)	*BLnID*	在 down 方向的下一个区块的编号
int_{14} (int_{62})	*BL*	CAL_EOA(*VM*)	*BLLen*	区块的长度
int_{15} (int_{63})	*BL*	CAL_EOA(*VM*)	*BLiBR*	区块所属分支的编号
int_{16} (int_{64})	*BR*	CAL_EOA(*VM*)	*BRpID*	在 up 方向的下一个分支的编号
int_{17} (int_{65})	*BR*	CAL_EOA(*VM*)	*BRnID*	在 down 方向的下一个分支的编号
int_{18} (int_{66})	*BR*	CAL_EOA(*VM*)	*BRLen*	分支的长度
int_{19} (int_{67})	*LO*	CAL_EOA(*VM*)	*LOpID*	在 up 方向的下一个区块实体的编号
int_{20} (int_{68})	*LO*	CAL_EOA(*VM*)	*LOnID*	在 down 方向的下一个区块实体的编号
int_{21} (int_{69})	*LO*	CAL_EOA(*VM*)	*LOLen*	区块实体的长度
int_{22} (int_{70})	*LO*	CAL_EOA(*VM*)	*LOiBR*	区块实体所属分支的编号
int_{23} (int_{71})	*RO*	CAL_EOA(*VM*)	*ROpID*	在 up 方向的下一个分支实体的编号
int_{24} (int_{72})	*RO*	CAL_EOA(*VM*)	*ROnID*	在 down 方向的下一个分支实体的编号
int_{25} (int_{73})	*RO*	CAL_EOA(*VM*)	*ROLen*	分支实体的长度
int_{26} (int_{74})	*OT*	CAL_EOA(*VM*)	*OTHaPo*	其他列车最大列车车头的坐标
int_{27} (int_{75})	*OT*	CAL_EOA(*VM*)	*OTHiPo*	其他列车最小列车车头的坐标
int_{28} (int_{76})	*OT*	CAL_EOA(*VM*)	*OTTaPo*	其他列车最大列车车尾的坐标
int_{29} (int_{77})	*OT*	CAL_EOA(*VM*)	*OTTiPo*	其他列车最小列车车尾的坐标
int_{30} (int_{78})	*OT*	CAL_EOA(*VM*)	*OVHaPo*	其他列车最大虚拟列车防护头的坐标
int_{31} (int_{79})	*OT*	CAL_EOA(*VM*)	*OVHiPo*	其他列车最小虚拟列车防护头的坐标
int_{32} (int_{80})	*OT*	CAL_EOA(*VM*)	*OVTaPo*	其他列车最大虚拟列车防护尾的坐标
int_{33} (int_{81})	*OT*	CAL_EOA(*VM*)	*OVTiPo*	其他列车最小虚拟列车防护尾的坐标
int_{34} (int_{82})	*OT*	CAL_EOA(*VM*)	*OTLen*	其他列车的长度
int_{35} (int_{83})	*SI*	CAL_EOA(*VM*)	*SIPos*	信号机的坐标
int_{36} (int_{84})	*SI*	CAL_EOA(*VM*)	*SIDir*	信号机的有效方向
int_{37} (int_{85})	*SI*	CAL_EOA(*VM*)	*SISta*	信号机的状态
int_{38} (int_{86})	*ZB*	CAL_EOA(*VM*)	*ZBPos*	区域控制器边界的坐标
int_{39} (int_{87})	*ZB*	CAL_EOA(*VM*)	*ZBDir*	区域控制器边界的有效方向
int_{40} (int_{88})	*TD*	CAL_EOA(*VM*)	*TDPos*	可行驶方向标识的坐标
int_{41} (int_{89})	*TD*	CAL_EOA(*VM*)	*TDDir*	可行驶方向标识的有效方向
int_{42} (int_{90})	*TD*	CAL_EOA(*VM*)	*TDSta*	可行驶方向标识的状态
int_{43} (int_{91})	*BZ*	CAL_EOA(*VM*)	*BZPos*	缓冲区的坐标
int_{44} (int_{92})	*BZ*	CAL_EOA(*VM*)	*BZDir*	缓冲区的有效方向
int_{45} (int_{93})	*BZ*	CAL_EOA(*VM*)	*BZSta*	缓冲区的状态
int_{46} (int_{94})	*OL*	CAL_EOA(*VM*)	*OLPos*	搭接区的坐标
int_{47} (int_{95})	*OL*	CAL_EOA(*VM*)	*OLDir*	搭接区的有效方向
int_{48} (int_{96})	*TR*	CAL_EOA(*VM*)	*Received*	已接收 EOA_Report
int_{97}	*VM*	*VR*	*bValid*	验证结果
int_{98}	*VM*	*VR*	*cExample*	系统当前状态
int_{99}	*VR*	*VM*	*reportEfforts*	报告效果

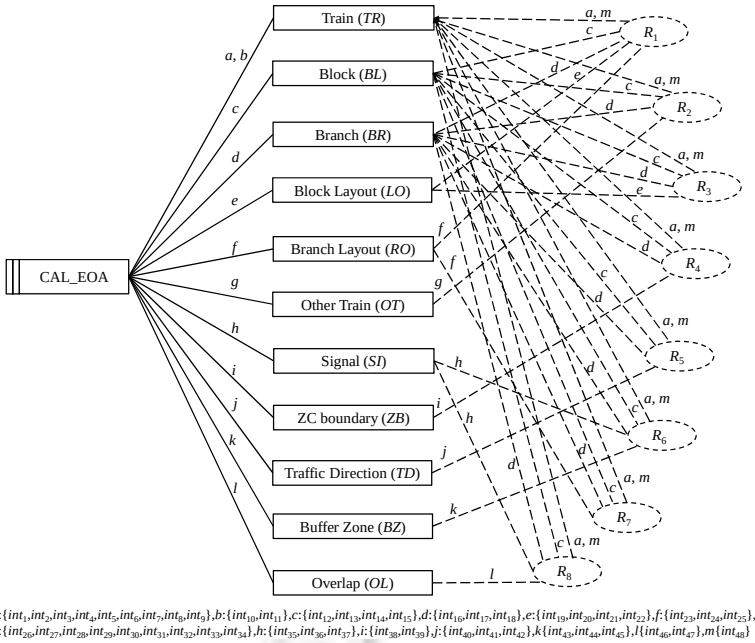


Fig.7 Problem diagram of CAL_EOA

图 7 CAL_EOA 的问题图

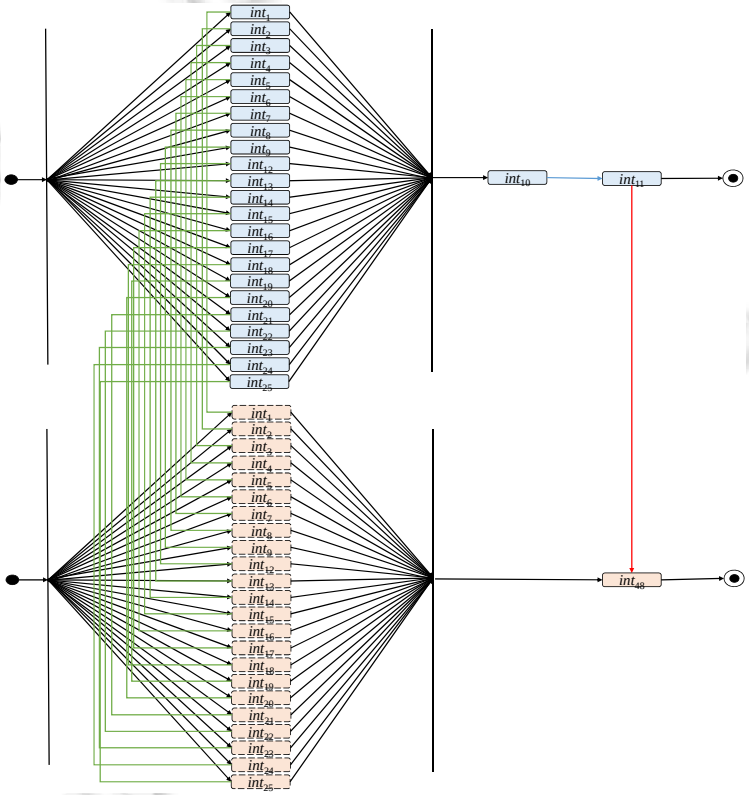


Fig.8 Scenario graph of R_1

图 8 R_1 的情景图

类似地,可以得到其他的 7 个子需求的情景图,限于篇幅,详见网址 https://github.com/lxsbamboo/ZC-Verification.git.基于根据这些情景图,使用文献[19]中的基于情景的需求投影方法,将问题图 5 分解,得到 8 个子问题.每个子问题都对应解决一个子需求,有自己的子环境和子交互.这 8 个问题是类似的.以子问题 P_1 为例,其用来对子需求 R_1 进行验证,在待开发软件中,P_1 涉及到的子机器为 M_1,和 M_1 进行交互的外部实体为子环境 E_1,包括 Train、Block、Branch、Block Layout 和 Branch Layout,IS_1 为 M_1 和 E_1 之间发生的交互,其中,$\{int_1,\ldots,int_9\}$ 为 Train 发送给 M_1 的列车定位信息,$\{int_{12},\ldots,int_{15}\}$为 Block 发送给 M_1 的 Block 逻辑分布,$\{int_{16},\ldots,int_{18}\}$为 Branch 发送给 M_1 的 Branch 逻辑分布,$\{int_{19},\ldots,int_{22}\}$为 Block Layout 发送给 M_1 的 Block 实体分布,$\{int_{23},\ldots,int_{25}\}$为 Branch Layout 发送给 M_1 的 Branch 实体分布,$\{int_{10},int_{11}\}$为 M_1 经过计算后发送的 EOA 报文,int_{48} 为 Train 的报文接收状态.其他子问题详见表 4.

Table 4 Decomposition results of sub-problem diagrams

表 4 子问题图的分解结果

子问题 P_i	子机器 M_i	子环境 E_i	子交互 IS_i	子需求 R_i
P_1	M_1	E_1={TR,BL,BR,LO,RO}	$\{int_1,\ldots,int_{25},int_{48}\}$	R_1
P_2	M_2	E_2={TR,BL,BR,OT}	$\{int_1,\ldots,int_{18},int_{26},\ldots,int_{34},int_{48}\}$	R_2
P_3	M_3	E_3={TR,BL,BR,LO}	$\{int_1,\ldots,int_{18},int_{19},\ldots,int_{22},int_{48}\}$	R_3
P_4	M_4	E_4={TR,BL,BR,ZB}	$\{int_1,\ldots,int_{18},int_{38},int_{39},int_{48}\}$	R_4
P_5	M_5	E_5={TR,BL,BR,TD}	$\{int_1,\ldots,int_{18},int_{40},\ldots,int_{42},int_{48}\}$	R_5
P_6	M_6	E_6={TR,BL,BR,SI,BZ}	$\{int_1,\ldots,int_{18},int_{35},\ldots,int_{37},int_{43},\ldots,int_{45},int_{48}\}$	R_6
P_7	M_7	E_7={TR,BL,BR,RO}	$\{int_1,\ldots,int_{18},int_{23},\ldots,int_{25},int_{48}\}$	R_7
P_8	M_8	E_8={TR,BL,BR,SI,OL}	$\{int_1,\ldots,int_{18},int_{35},\ldots,int_{37},int_{46},\ldots,int_{48}\}$	R_8

4.2 需求验证模型的生成

根据表 1 中的转换规则,可以根据每个子问题的问题图自动生成其对应的需求验证模型.以子问题 P_1 为例,说明其具体的生成过程.首先,根据子机器 M_1 构建的验证子机器可以构造 VM_1.VR_1 为验证报告.E_1,M_1 和 VR_1 共同构成验证子问题的环境 E_1',接下来生成 VM_1 和 E_1' 之间的交互.

对于 M_1 和 E_1 之间的交互 a,b,c,d,e,f,m 而言,由于其交互发送方及交互内容不变,交互的接收方变为验证子机器 VM_1,因此用 a',b',c',d',e',f',m'来表示 VP_1 中 VM_1 与 E_1 和 M_1 之间的交互,并且构成新的验证子交互 IS_1',其中,$\{int_{49},\ldots,int_{57}\}$为 Train 发送给 VM_1 的列车定位信息、$\{int_{58},int_{59}\}$为 CAL_EOA 发送给 VM_1 的 EOA 报文、$\{int_{60},\ldots,int_{63}\}$为 Block 发送给 VM_1 的 Block 逻辑分布、$\{int_{64},\ldots,int_{66}\}$为 Branch 发送给 VM_1 的 Branch 逻辑分布、$\{int_{67},\ldots,int_{70}\}$为 Block Layout 发送给 VM_1 的 Block 实体分布,$\{int_{71},\ldots,int_{73}\}$为 Branch Layout 发送给 VM_1 的 Branch 实体分布,int_{96} 为 Train 的报文接收状态.n 表示 VM_1 与验证报告 VR_1 之间的交互,其中 int_{97} 表示 VM_1 发送给 VR_1 的验证结果 $bVlid$,int_{98} 为 VM_1 发送给 VR_1 的验证错误时反例,int_{99} 为 VR_1 发送给 VM_1 的报告效果 reportEfforts.因此, $IS_1' = IS_1' \cup \{int_{97},int_{98},int_{99}\}$.

根据图 8 中的交互情况,我们可以得到需求验证模型的动态视图.首先,VM 观察到 Train、Block、Branch、Block Layout 和 Branch Layout 分别发送列车定位信息($int_{49},\ldots,int_{57}$)、Block 逻辑分布($int_{60},\ldots,int_{63}$)、Branch 逻辑分布($int_{64},\ldots,int_{66}$)、Block 实体分布($int_{67},\ldots,int_{70}$)以及 Branch 实体分布($int_{71},\ldots,int_{73}$);然后,VM 接收到 CAL_EOA 发送的移动授权终点的计算结果(int_{58},int_{59});接下来,VM 观察到列车发送的报文接收状态(int_{96});最后,根据获得的信息,VM 将 CAL_EOA 是否满足需求 R_1 的验证结果(int_{97})和验证错误时的反例(int_{98})发送给验证报告 VR.限于篇幅,具体的视图参见 https://github.com/lxsbamboo/ZC-Verification.git.

4.3 需求验证性质的生成

同样以子问题 P_1 为例,可以根据子问题图中的需求 R 自动生成验证性质 R'.R'引用机器 M_1 和环境 E_1 发送给 VM 的交互 a',b',c',d',e',f',m',并约束 VM 和 VR 间的交互 o.由上述描述,子问题 P_1 生成的验证系统的问题图 VP_1 可以对其进行静态描述,如图 9 所示.机器 M_1 和环境 E_1 发送交互信息给验证机器 VM,然后,VM 对机器 M_1 是否满足需求 R 进行验证并向 VR 发送验证结果和错误时的反例.通过问题领域 M_1 和 E_1 中的交互和期望生成的 VM 和 VR 间的交互来判断是否满足验证性质 R'.其中,交互的具体定义及变量的含义见表 3.

基于图 9 验证子问题 VP_1 的问题图,根据图 8 中需求引用和约束的先后顺序关系,我们可以对子问题 R_1 的验证系统进行动态的描述:首先,*VM* 观察到 Train、Block、Branch、Block Layout 和 Branch Layout 分别发送的列车定位信息($int_{49},\ldots,int_{57}$)、Block 逻辑分布($int_{60},\ldots,int_{63}$)、Branch 逻辑分布($int_{64},\ldots,int_{66}$)、Block 实体分布($int_{67},\ldots,int_{70}$)以及 Branch 实体分布($int_{71},\ldots,int_{73}$);然后,CAL_EOA 将移动授权终点的计算结果发送给 *VM*(int_{58},int_{59});接下来,*VM* 观察到列车发送的报文接收状态(int_{96});最后,期望 *VR* 的报告效果 reportEfforts 发生变化(int_{99}).限于篇幅,具体过程参见 https://github.com/lxsbamboo/ZC-Verification.git.

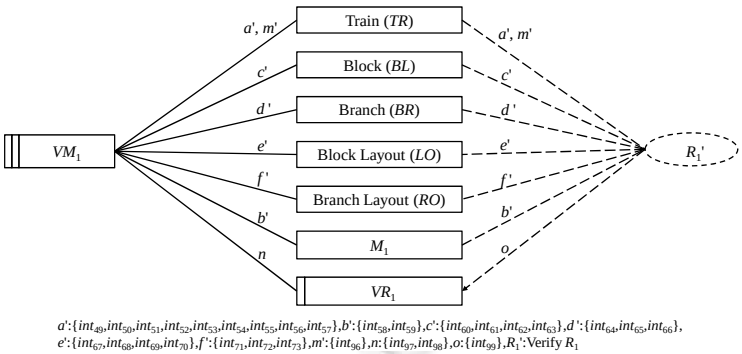


Fig.9 A problem diagram for verifying sub-problem VP_1

图 9 验证子问题 VP_1 的问题图

4.4 需求验证问题Scade模型的生成

基于 8 个验证子问题图以及子问题的情景图,根据算法 1,可以自动生成 8 个 Scade 模型,对其进行组合即可对安全需求进行验证.还是以 VP_1 为例,说明其 Scade 模型的生成.操作符 *M* 用来描述行为交互,它的输入变量为行为交互中所有 M_1 接收的交互变量的集合,也就是 E_1 发送给 M_1 的设备位置信息,即 *ThaPo*,*THiPo* 等;其输出为行为交互中所有由 M_1 发出的交互变量集合,也就是 M_1 发送给的移动授权分配结果,即 *EOAPos* 和 *EOATyp*.操作符 *E* 用来描述交互使能,在图 8 中,{$int_1,\ldots,int_9,int_{12},\ldots,int_{25}$}之间存在同步序关系,$int_{11}$ 和 int_{48} 之间存在行为使能关系,因此,*E* 的输入{$int_1,\ldots,int_9,int_{11},\ldots,int_{25},int_{48}$}中的交互变量.在操作符 *E* 中,$E_1$ 发送的设备位置信息用局部变量来表示,EOA 报文和报文接收状态 *receives* 用传感器表示,在 *E* 中创建断言来表示这些交互之间的使能关系.操作符 *O* 用期望交互描述,其输入为 *M* 和 *E* 的所有输入和输出变量集合的并集,输出为验证结果 *Result*.图 10 是该验证子问题的 Scade 模型示意图,其中,我们把局部输入变量复制了 3 份来提高模型的易读性.

O 是对期望交互的描述,用来表示外部设备在发送相应位置信息后得到的状态变化.因此,我们根据情景图中的期望交互来构建具体的操作符 *O*.在操作符 *O* 中,用局部变量来表示外部设备发送的设备位置信息,用 sensor 来表示报文接收状态 *received*.当列车位置信息(*THaPo*,*THiPi*,*TTaPo*,*TTiPo*,*VHaPo*,*VTaPo*,*VTiPo*,*TRlen*)、Block 逻辑布局(*BLpID*,*BLnID*,*BLLen*,*BLiBR*)、Branch 逻辑布局(*BRpID*,*BRnID*,*BRLen*)、Block 实体布局(*LOpID*,*LOnID*,*LOLenLOiBR*)和 Branch 实体布局(*ROpID*,*ROnID*,*ROLen*)全部发送成功,即以上变量皆不为空时,我们希望 Train 的报文接收状态变为 *received*,即 *received* 变量不为空.对以上结果用 *imply* 操作符连接,并将判断结果输出.根据以上描述,我们可以得到操作符 *O* 的具体内容,如图 11 所示.

类似地,我们得到所有的 8 个验证子问题的 SCADE 模型.这 8 个子问题类似.在现实规模上,以 VP_1 为例,它有 19 个设备,包括 3 辆车(train)、6 个区块(block)、6 个分支(branch),其他设备各 1 个.它与这些设备的交互大约有 265 个变量现象.在进行验证时,使用如下机器配置:WIN7 操作系统,处理器为 E5 2620 v3,RAM 为 64GB,可以得到 ZC 的 8 个安全子需求皆能验证成功,其中,子问题 7 的验证时间最短,131s;子问题 4 的验证时间最长,为 147s.每个子问题验证的平均时间为 138s,所有子问题的总验证时长为 1 103s.所有子问题的验证结果详见网址 https://github.com/lxsbamboo/ZC-Verification.git.实际上,之前由于 CAL_EOA 问题过大,验证过程中出现了状态爆炸问题,SCADE 中的内置验证器无法对该系统进行验证.对比之下,充分说明必须对模型进行分解,最好能

够在需求阶段进行分解.

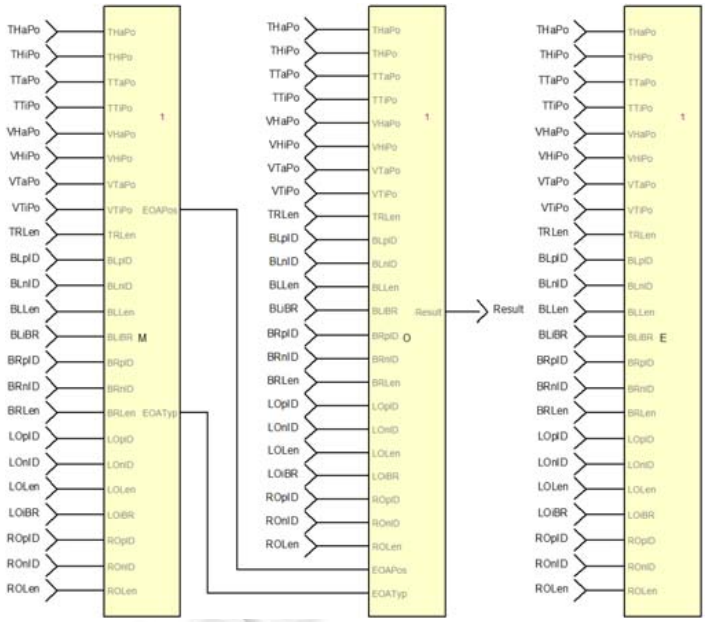


Fig.10 Scade model for verifying sub-problem VP_1

图 10 验证子问题 VP_1 的 Scade 模型

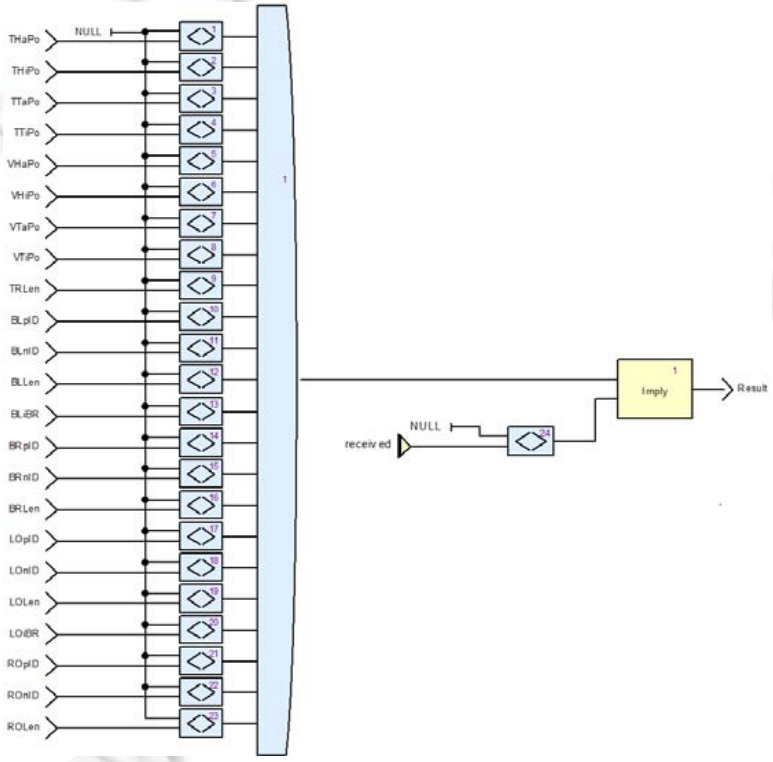


Fig.11 Operator *O* in the Scade model

图 11 Scade 模型中的操作符 *O*

5 相关工作

本文研究轨道交通系统的安全需求的自动验证问题.已经有很多安全需求的验证工作.根据安全需求的描述方式不同,可以分为半形式化需求的验证和形式化需求的验证.

半形式化需求的验证,实际上是指需求是由基于统一建模语言(UML)[21]等半形式化语言表示,可以由领域专家绘制,而验证则要将半形式化语言转换为形式化语言进行.这方面的工作也很多,例如,黄友能等人[12]采用扩展后的UML对ZC子系统的系统功能进行半形式化建模,然后根据转换规则将UML模型转换为线性混成自动机的形式化模型,使用BACH软件[22]进行验证,验证的是需求的一致性;Fotso等人[13]使用SysML/KAOS[23]对系统需求、领域特性以及与混合ERTMS/ETCS 3级标准[24,25]相关的安全不变量进行半形式化建模,然后自动转换为B系统[26]规范,以获得形式化规范的体系结构,最后用Rodin工具[27]对其进行验证,也是验证的需求的一致性;杨璐等人[14]采用半形式化方法消息顺序图(message sequence chart,简称MSC)[28]对ZC切换场景功能和受限活性,然后将MSC模型转换为形式化的时间自动机,通过UPPAAL[29]对其进行验证.我们的方法本质上也是属于这种类方法,但本文的优点在于:(1) 问题框架方法在建模需求时其独特的环境视角与其他的建模语言有本质的区别,它更适合建模ZC这类系统;(2) 自动的需求分解,可以极大地降低系统复杂度;(3) 自动的验证模型生成,非常方便领域专家使用;(4) 与其他需求一致性验证不同,本文验证的是规约是否满足需求,属于需求可满足性验证的一种;(5) 基于需求分解的组合验证,可以提高验证效率,部分缓解因复杂带来的状态空间爆炸.

在形式化需求验证中,都是形式化专家根据领域专家的自然语言描述,直接建立形式化模型,并采用相应的形式化验证工具进行性质验证.例如,Hartig 等人[9]在对用自然语言描述的铁道车辆的安全需求进行分析之后,使用ANSI/ISO-C规范语言(ACSL)[30]将其形式化,然后通过FRAMA-C[31]对形式化模型进行演绎验证;李蓉[10]使用SCADE来对用自然语言描述的ZC系统功能进行形式化建模与验证;Aiello等人[32]形式化需求建模语言(Form-L)[33]对使用自然语言描述的飞机子系统部件及其设计变体的需求进行建模,通过Modelica需求库[34,35]将需求的Form-L模型映射到Modelica[36]上,通过仿真实现对需求的验证;Chhabra等人[11]构建了一个基于Petri网和域的上下文推理的工具,将自然语言规范转换为形式化的表示决策表(expressive decision table,简称EDT)[37],然后使用转换过程中的本体来验证给定的其他需求,以检查它们与现有需求的一致性.上述验证方法,对于不具有形式化知识的领域专家来讲很难使用.而本文提出的方法是使用PF进行需求的建模,使用的原语为现实中的实体和实体与待构建的软件之间的交互,领域专家对此十分熟悉,相比来说会方便使用很多.另外,本文方法可以自动地去验证,不仅如此,我们还是一种组合验证方法,可以验证复杂的系统,特别适合ZC这类系统.

6 结束语

ZC是一个复杂的安全攸关系统,其安全需求的可满足性需要形式化的验证.本文提出了ZC系统安全需求建模、分解与自动验证的方法,采用PF方法对ZC的安全需求问题进行建模,继而通过基于情景的自动分解方法将需求问题分解成较小的子问题,根据每个子问题的需求模型自动生成需求的验证子问题模型,然后自动生成验证子问题的Scade形式化模型,最后通过模型检验器Design Verifier对Scade模型进行了验证.本文的主要贡献为设计了一个基于问题框架模型的ZC系统的安全需求自动验证方法,根据需求模型自动生成安全需求验证问题模型和验证性质,并生成由Scade语言描述的验证模型,进行组合验证.通过该方法可以由半形式化需求模型自动生成形式化模型进行验证,降低了ZC验证的难度,使得领域专家也可以做形式化验证.

我们将该方法应用到实际的轨道交通区域控制器的安全需求的建模、分解与验证中,并做了一些实验.实验结果证明,使用该方法可以有效地缓解状态空间爆炸问题,并能实现对安全需求自动验证,且验证时间成本大为降低,这使得ZC的安全需求验证问题有了明显的进步.本方法不仅为轨道交通系统,也为其他类似复杂系统,如信息物理系统的需求验证提供了一个很好的借鉴思路.

References:

[1] Gao CH. Research on the key techniques of the independent and innovative CBTC. Urban Rapid Rail Transit, 2011,24(4):1−4 (in Chinese with English abstract).

[2] Lutz RR. Analyzing software errors in safety-critical embedded systems. In: Proc. of the IEEE Int'l Symp. on Requirements Engineering. San Diego: Computer Society, 1993. 126−133.

[3] McDermid JA. Software safety: Where's the evidence? In: Proc. of the 6th Australian Workshop on Safety Critical Systems and Software, Vol.3. St Lucia: Australian Computer Society, 2001. 1−6.

[4] Barnat J, Bauch P, Benes N, Brim L, Beran J, Kratochvila T. Analysing sanity of requirements for avionics systems. Formal Aspects of Computing, 2016,28(1):45−63.

[5] BS EN 50128. Railway Applications—Communication, Signaling and Processing Systems—Software for Railway Control and Protection Systems. British Standards Institute, 2011.

[6] BS EN 50129. Railway Application—Communications, Signaling and Processing Systems—Safety Related Electronic Systems for Signaling. British Standards Institute, 2018.

[7] Yang XW. Modeling and safety verification of zone controller in CBTC with UML [MS. Thesis]. Beijing: Beijing Jiaotong University, 2008 (in Chinese with English abstract).

[8] Yuan Z, Chen X, Liu J, Yu Y, Sun H, Zhou T, Jin Z. IEEE Std 1474.1-2004 IEEE Standard for Communications-based Train Control (CBTC) Performance and Functional Requirements. The Rail Transit Vehicle Interface Standards Committee, 2005.

[9] Hartig K, Gerlach J, Soto J, Busse, J. Formal specification and automated verification of safety-critical requirements of a railway vehicle with Frama-C/Jessie. In: Proc. of the FORMS/FORMAT 2010. Berlin, Heidelberg: Springer-Verlag, 2011. 145−153.

[10] Li R. Modeling and verification of zone controller in CBTC with SCADE [MS. Thesis]. Chengdu: Southwest Jiaotong University, 2015 (in Chinese with English abstract).

[11] Chhabra A, Sangroya A, Anantaram C. Formalizing and verifying natural language system requirements using Petri nets and context based reasoning. In: Proc. of the MRC@ IJCAI. Stockholm: CEUR-WS.org, 2018. 64−71.

[12] Huang YN, Zhang PJ, Hou XP, Tang T. Modeling and verification method of ZC subsystem in urban rail transit based on hybrid automata. China Railway Science, 2016, 37(2):114−121 (in Chinese with English abstract).

[13] Fotso SJT, Frappier M, Laleau R, Mammar A. Modeling the hybrid ERTMS/ETCS level 3 standard using a formal requirements engineering approach. In: Butler MJ, ed. Proc. of the Int'l Conf. on Abstract State Machines, Alloy, B, TLA, VDM, and Z. Southampton: Springer-Verlag, 2018. 262−276.

[14] Yang L, Chen YG. Modeling and verification of switch scene of zone controller based on MSC and UPPAAL. Railway Standard Design, 2018,62(5):171−174,179 (in Chinese with English abstract).

[15] Jackson M. Software Requirements and Specifications: A Lexicon of Practice, Principles and Prejudices. Addison-Wesley, 1995.

[16] Jackson M. Problem Frames: Analyzing and Structuring Software Development Problems. Addison-Wesley, 2001.

[17] Le Sergent T. SCADE: A comprehensive framework for critical system and software engineering. In: Proc. of the Int'l SDL Forum. Berlin, Heidelberg: Springer-Verlag, 2011. 2−3.

[18] Yuan Z, Chen X, Liu J, Yu Y, Sun H, Zhou T, Zhi J. Simplifying the formal verification of safety requirements in zone controllers through problem frames and constraint-based projection. IEEE Trans. on Intelligent Transportation Systems, 2018,19(11): 3517−3528.

[19] Jin Z, Chen X, Didar Z. Performing projection in problem frames using scenarios. In: Sulaiman S, ed. Proc. of the 2009 16th Asia-Pacific Software Engineering Conf. Batu Ferringhi: IEEE Computer Society, 2009. 249−256.

[20] IEEE Recommended Practice for Communications-based Train Control (CBTC) System Design and Functional Allocations. IEEE Standard 1474.3-2008, 2008. 1−117.

[21] Fowler M, Kobryn C. UML Distilled: A Brief Guide to the Standard Object Modeling Language. Addison-Wesley Professional, 2004.

[22] Bu L, Li Y, Wang L, Li X. BACH: Bounded reachability checker for linear hybrid automata. In: Cimatti A, ed. Proc. of the 2008 Formal Methods in Computer-aided Design. Portland: IEEE, 2008. 1−4.

[23] Gnaho C, Semmak F. Une extension SysML pour l'ingénierie des exigences dirigée par les buts. In: Proc. of the INFORSID. 2010. 277−292.

[24] EEIG ERTMS Users Group. Hybrid ERTMS/ETCS Level 3. Principles Ref: 16E042, Version 1A, 2017.

[25] Furness N, van Houten H, Arenas L, Bartholomeus M. ERTMS Level 3: The game-changer. IRSE News, 2017,232:2−9.

[26] Abrial JR. Modeling in Event-B: System and Software Engineering. Cambridge University Press, 2010.

[27] Abrial JR, Butler M, Hallerstede S, Hoang TS, Mehta F, Voisin L. Rodin: An open toolset for modelling and reasoning in Event-B. Int'l Journal on Software Tools for Technology Transfer, 2010,12(6):447−466.

[28] Rudolph E, Graubmann P, Grabowski J. Tutorial on message sequence charts. Computer Networks and ISDN Systems, 1996,28(12): 1629−1641.

[29] Bengtsson J, Larsen K, Larsson F, Pettersson P, Yi W. UPPAAL—A tool suite for automatic verification of real-time systems. In: Proc. of the Int'l Hybrid Systems Workshop. Berlin, Heidelberg: Springer-Verlag, 1995. 232−243.

[30] Baudin P, Filliâtre JC, Marché C, Monate B, Moy Y, Prevosto V. ACSL: ANSI/ISO C specification language. Version 1.4. 2009. http://framac.cea.fr/download/acsl_1.4.pdf

[31] Correnson L, Cuoq P, Puccetti A, Signoles J. Frama-C user manual, boron release. 2010. http://frama-c.com/download/user-manual-Boron-20100401.pdf

[32] Aiello F, Garro A, Lemmens Y, Dutré S. Formal modeling of system properties for simulation-based verification of requirements: Lessons learned. In: Fierro D, ed. Proc. of the 3rd INCOSE Italia Conf. on Systems Engineering. Naples: CEUR-WS.org, 2017. 54−61.

[33] Nguyen T. FORM-L: A modelica extension for properties modelling illustrated on a practical example. In: Proc. of the 10th Int'l Modelica Conf., Vol.96. Lund: Linköping University Electronic Press, 2014. 1227−1236.

[34] Garro A, Tundis A, Bouskela D, Jardin A, Thuy N, Otter M, Buffoni L, Fritzson P, Sjölund M, Schamai W, Olsson H. On formal cyber physical system properties modeling: A new temporal logic language and a modelica-based solution. In: Proc. of the 2016 IEEE Int'l Symp. on Systems Engineering (ISSE). IEEE, 2016. 1−8.

[35] Otter M, Thuy N, Bouskela D, Buffoni L, Elmqvist H, Fritzson P, Garro A, Jardin A, Olsson H, Payelleville M, Schamai W, Thomas E, Tundis A. Formal requirements modeling for simulation-based verification. In: Proc. of the 11th Int'l Modelica Conf., Vol.118. Versailles: Linköping University Electronic Press, 2015. 625−635.

[36] Fritzson P, Engelson V. Modelica—A unified object-oriented language for system modeling and simulation. In: Proc. of the European Conf. on Object-oriented Programming. Berlin, Heidelberg: Springer-Verlag, 1998. 67−90.

[37] Venkatesh R, Shrotri U, Krishna GM, Agrawal S. EDT: A specification notation for reactive systems. In: Fettweis GP, ed. Proc. of the 2014 Design, Automation & Test in Europe Conf. & Exhibition (DATE). Dresden: European Design and Automation Association, 2014. 1−6.

附中文参考文献:

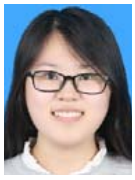
[1] 郜春海.自主创新 CBTC 系统的核心技术研究.都市快轨交通,2011,24(4):1−4.

[7] 杨旭文.基于 UML 的 CBTC 系统区域控制器的建模与安全性验证[硕士学位论文].北京:北京交通大学,2008.

[10] 李容.基于 SCADE 的 CBTC 区域控制器建模与验证[硕士学位论文].成都:西南交通大学,2015.

[12] 黄友能,张鹏基,侯晓鹏,唐涛.基于混成自动机的城市轨道交通 ZC 子系统建模与验证方法.中国铁道科学,2016,37(2):114−121.

[14] 杨璐,陈永刚.基于 MSC 与 UPPAAL 的区域控制器切换场景建模与验证.铁道标准设计,2018,62(5):171−174,179.



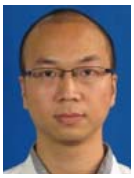
刘筱珊(1996－),女,山东平度人,硕士生,CCF 学生会员,主要研究领域为需求工程,形式化方法.

陈铭松(1982－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为信息物理融合系统设计自动化,计算机体系结构,物联网技术,形式化方法.

袁正恒(1990－),男,博士,主要研究领域为需求工程,形式化方法,安全攸关系统.

刘静(1964－),女,博士,教授,博士生导师,CCF 专业会员,主要研究领域为可信软件,模型驱动式软件开发方法,面向服务的软件架构.

陈小红(1982－),女,博士,副教授,CCF 专业会员,主要研究领域为需求工程,形式化方法,安全攸关系统.



周庭梁(1980－),男,博士,高级工程师,主要研究领域为安全苛求系统,可信测评,形式化方法.