基于表格表达式的SCR需求模型转换

李思洁 魏 欧 战芸娇 王立松
(南京航空航天大学计算机科学与技术学院 南京 211100)

摘 要 基于形式化方法的需求规约过程以严格定义的语义和数学模型为基础,使得需求的表述更加清晰明了,易于理解。SCR方法是一种基于形式化符号-表格的表达式,以多维表格化结构表示系统需求的形式化需求规约方法。针对形式化需求的自动化测试和检验工具提高了需求分析的正确性和效率性,但目前工具缺少安全性质的自动验证,无法保证需求的安全性。因此,文中对基于SCR方法的T-VEC工具进行扩展,在语言解析器生成器antlr(ANother Tool for Language Recognition)的辅助下开发了模型转换工具T2N,设计了语言结构转换规则,将基于SCR的需求描述语言T-VEC转换为符号化模型检测语言XMV,以实现对提取的系统安全性质的自动化验证。最后,以需求工程中的典型案例——灯光控制系统为例进行实验分析,验证T2N工具的有效性和需求模型的安全性。

关键词 SCR方法,antlr工具,模型转换,T2N工具
中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2019.06.027

SCR Requirement Model Transformation Based on Table Expression

LI Si-jie WEI Ou ZHAN Yun-jiao WANG Li-song
(College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 211100, China)

Abstract The process of requirement specification based on formal methods is based on strictly defined semantics and mathematical models, making the presentation of requirements clearer and easier to understand. SCR method is a formalized requirement specification method based on formal symbol-tabular expression, which uses multi-dimensional tabular structure to represent system requirements. The automated testing and verification tools for formal requirements increase the accuracy of the requirements and the efficiency of analysis. However, some current tools lack of automatic verification of safety properties and can't guarantee the safety of the requirements. Therefore, this paper expanded T-VEC tool based on SCR method, developed model transform tool T2N with the help of language parser generator antlr, designed anguage structure transformation rules, and transformed requirement description language T-VEC based on SCR method into symbolic model checking language XMV, to achieve automatic verification of the extracted system safety properties. Finally, an example of a typical case lighting control system in requirement engineering was used for experimental analysis to verify the effectiveness of the T2N tool and the safety of the requirement model.

Keywords SCR method, Antlr tool, Model transform, T2N tool

1 引言

需求分析是软件生命周期中最重要的阶段[1]。Heitmeyer等[2]发现大部分的系统错误都是由需求中的错误引起的,而需求中的错误往往在需求分析和获取阶段引入[3]。表格表达式[4]是一种基于多维表格化结构表示计算机系统中变量之间的关系或函数的形式化符号。其作用在于通过对复杂的描述系统行为的数学表达式进行分解,来增强可读性、可理解性和可验证性,以便于检查需求[5]的一致性和完备性,及早在需求分析阶段发现缺陷和错误,减少后续的软件开发成本。

表格表达式最初结合SCR(Software Cost Reduction)方法[6]在美国海军的A-7E舰载机的软件系统开发中得到成功应用。SCR方法基于Parnas所提出的四变量模型(four variable model)[7],将软件系统及其交互环境中的变量分为监控变量、控制变量、输入变量、输出变量[8];为了准确描述需求,其增加了中间变量、模式、模式类;并且将软件系统的需求定义为变量之间的各种关系,这些关系所对应的数学表达式以表格形式定义[9]。其中的表格符号主要包括:模式转换表

到稿日期:2018-05-08 返修日期:2018-09-26 本文受国家重点基础研究发展计划(973计划)(2014CB744904),航空科学基金项目(20155552047),校研究生创新基地(实验室)开放基金资助项目(kfjj20181602)资助。

李思洁(1995-),女,硕士生,主要研究方向为系统安全性分析,E-mail:lisijie@nuaa.edu.cn;**魏 欧**(1974-),男,博士,副教授,CCF会员,主要研究方向为形式化方法、软件自动验证,E-mail:owei@nuaa.edu.cn(通信作者);**战芸娇**(1993-),女,硕士生,CCF会员,主要研究方向为形式化方法、需求分析;**王立松**(1969-),男,博士,副教授,CCF会员,主要研究方向为系统软件、软件工程。

(mode transition table)，用于描述不同模式下系统状态的转换关系；条件表(condition table)，用于描述在不同状态和条件下系统变量的取值；事件表(event table)，用于描述在不同状态以及事件触发下系统变量的取值。

T-VEC 是一款商用软件，是针对 SCR 方法，对系统的需求阶段和设计阶段进行测试验证和仿真的工具[10]，它同时提供了对 SCR 需求模型中表格的形式化文本描述，可实现对 SCR 方法的语法、语义检查，并且对模型中逻辑关系的一致性、完备性进行检验，使得系统开发工程师可以及早在需求阶段发现系统设计中的错误，降低系统开发成本。但 T-VEC 只能做静态检验，因此，本文对 T-VEC 工具进行进一步的完善，将其转换为模型检测语言，使其可以对系统的安全性质进行验证。

模型检测[11]是一种重要的自动化验证技术，由 Clarke 等以及 Quielle 等于 1981 年分别提出。其基本思想是：系统的行为使用状态迁移系统来表示，系统的性质采用模态逻辑公式来描述，将“系统是否满足某条性质的问题”转化为验证数学问题“状态迁移系统是否是逻辑公式的模型”，通过计算机程序在有限的时间内得到结果[12]。NuSMV 是常用的模型检测器，输入系统模型和系统性质即可完成对模型的验证。

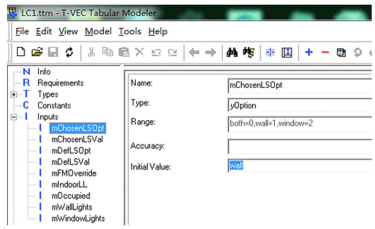
时序逻辑(temporal logic)是计算机科学专业中的重要领域，主要用于形式化验证，被用来描述表现和推理关于时间限定的命题的规则和符号化的任何系统。本文研究主要从系统由不同的起始状态出发的并行执行考虑，为了将安全性质加入到 T2N 转换后的 NuXMV 文件中，对需求模型的安全性进行验证，使用线性时序逻辑(Linear-time Temporal Logic，LTL)[13]将从模型中抽取的安全性质表示为时序逻辑公式的形式。

本文完成了 T2N 模型转换工具的开发，将基于 SCR 的 T-VEC 语言转换为符号化模型检测语言 XMV，为工具增加了安全性质自动验证的功能。该转换工具严格以 T-VEC 语言和 XMV 语言的语义为基础，借助 antlr 语法解析工具，来实现对系统安全行为性质的验证，提高系统的安全性。本文第 2 节对 T-VEC 和 NuXMV 语言进行介绍；第 3 节对 T2N 原型工具的设计框架进行说明；第 4 节描述工具的实现过程；第 5 节对工具的正确性进行实例验证；最后总结本文的工作，并对未来的工作进行展望。

2　T-VEC 与 NuXMV 语言

2.1　T-VEC 语言

T-VEC 语言定义了 SCR 需求模型中表格表达式的文本表示，由 T-VEC 工具生成，以.SS 为后缀的文件形式存储，文件中是一条条对系统定义和控制的语句，语句按顺序执行。图 1(a)为 T-VEC 及文本描述案例，其为使用 T-VEC 对 SCR 需求模型进行建模的界面，图 1(b)为 T-VEC 生成的文本化描述，不同于以往主流的编程语言，T-VEC 语言是对现实系统的抽象，支持非确定性的控制结构，没有返回值，亦没有复杂的数据结构等概念。但 T-VEC 与主流语言又有很多相似的特征和语法形式，如变量和类型的定义、变量的初始化与赋值、算术运算、注释、语法结构上的递归定义等。



(a)

```
VERSION "3.0.0"
SUBSYSTEM mcStatus(_mOccupied,mOccupied, _mcStatus)
{

 TYPE mcStatus__Type ISA ENUMERATION RANGE { unoccupied = 0 , occupied = 1 ,
temp_empty = 2 } ;
 VARIABLE _mOccupied ISA Boolean ;

 VARIABLE _mcStatus ISA mcStatus__Type ;

 VARIABLE mOccupied ISA Boolean ;

 VARIABLE mcStatus__OUT ISA mcStatus__Type ;


 LOGIC STRUCTURE
mcStatus_FROM_occupied_NoChange__LS(_mcStatus, mOccupied,mOccupied)
   CONSTRAINT
            (_mOccupied = TRUE)
               AND
            ((_mcStatus = occupied::mcStatus__Type)
               AND
            (_mOccupied = mOccupied));
```

(b)

图 1　T-VEC 及文本描述案例

Fig. 1　T-VEC and Text description case

逻辑上，T-VEC 语言由 3 种文件组成：1)T-VEC 语言中所有基本类型变量及其取值范围的定义，包括布尔型、浮点型、无符号数等；2)用户自定义类型和基本类型的传递定义，这部分主要是为了当基本类型的取值范围改变时，只需要改变基本类型文件中定义的内容，而不用修改传递定义文件中的内容，以保证变量取值范围的逻辑一致性；3)模型中状态迁移的描述文件，这类文件最为重要，包括对模型中变量的赋值、初始化，以及控制变量取值变化的约束条件，其本质是 SCR 需求模型中为表示变量取值变化的各种条件表、事件表、模式表，目的是为了体现模型的动态性。

实际上，T-VEC 语言通过 8 种基本结构来定义模型性质，分别是常量定义、变量类型定义、输入变量定义、中间变量定义、函数定义、断言定义、模式类定义、输出变量定义。其中中间变量、函数、断言、模式类的定义包含条件表和事件表，属于模型的约束条件定义；输出变量体现了模型的功能定义。这 8 种基本结构构成了 3 种描述文件。

2.2　NuXMV 语言

NuXMV[14]是一种对有限状态和无限状态系统进行同步分析的符号化模型检测器，可以以状态迁移的形式描述模型，并对模型中的规范进行验证。其前身是 NUSMV，由卡内基梅隆大学在 SMV 的基础上改进得到，提供了层次化建模和可复用组件的定义[15]。NuXMV 支持使用时序逻辑表达式 CTL 和 LTL 描述的性质的验证。

与 NUSMV 相比，NuXMV[16]主要在两个方面对 NUSMV 的功能进行了扩展。对于有限状态的情况，NuXMV 提供了最先进的基于 SAT 算法的强大的验证引擎；对于无限状态的情况，NuXMV 增加了整型和实数型两类变量，可实现无限状态的计算。由于 T-VEC 语言支持无限类型变量，因此本文选择 NuXMV 作为目标语言进行转换。

为了方便，下文中均将 NuXMV 模型建模语言称为 XMV。一个 XMV 程序由若干的模块定义所组成，模块记录了模型在模块内状态迁移时发生的行为，其中包括唯一的主模块(MODULE main)和若干子模块，模块之间可以互相嵌套，所有的模块嵌套在主模块中。XMV[17]的结构由关键字

MODULE, VAR, IVAR, DEFINE, ASSIGN, TRANS, INVAR, INIT, NEXT 和 SPEC 等表示。MODULE 代表可复用组件的单元;VAR 和 IVAR 用于变量声明;DEFINE 代表宏定义,可使用宏来符号化表示表达式;ASSIGN 和 TRANS 用来说明转换关系;INVAR 代表系统中的常量;INIT 表示对变量进行初始化,变量的取值是模块启动时的数值;NEXT 代表在相应的约束条件成立时,变量的下一时刻取值;SPEC 用于使用时序逻辑来说明系统性质。

3 T2N 原型工具的设计框架

本节对 T-VEC 模型到 NuXMV 模型的基本转换过程和实现方法进行了描述,并在 T2N 工具中得以实现。

NuXMV 的输入语言旨在描述在不同抽象级别的同步或异步有限状态机,与 SCR 方法的结构有很强的对应关系。在 NuXMV 中,系统被描述为状态迁移的模型;而对于任意一个由 T-VEC 建立的需求模型,所有变量的取值构成了模型的当前状态,状态的迁移由事件或条件通过改变相应变量的取值来驱动,因此无论是 NuXMV 模型,还是 T-VEC 创建的模型,都可以表示为有限状态机的形式,故只要 T-VEC 表达的状态迁移系统在 NuXMV 的表达范围内,就一定能将 T-VEC 模型转换为 NuXMV 模型,进而完成对 T-VEC 需求模型的验证。因此,可以通过设计一些辅助工具来实现两种模型语言之间的转换。T2N 实现框架的具体流程如图 2 所示。

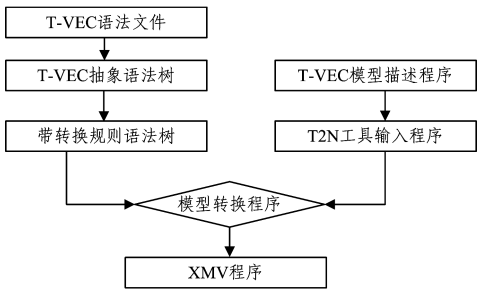


图 2 T2N 实现框架

Fig. 2 Implementation framework of T2N

将 T-VEC 程序(记为 T)描述的模型转换成 NuXMV 模型,并由 XMV 程序描述(记为 N),其主要包括以下 3 个过程。

1)对 T 进行预处理,使得输入到 T2N 的文件满足处理的约束条件。

2)利用 antlr 工具对 T 的语法文件进行解析,生成 T 的抽象语法分析树、语法树进出节点处理函数和语法树遍历程序。

3)基于 2)的处理结果,将 T 中的模型描述结构翻译成 N 中的模块类型,翻译过程主要完成下面 3 个步骤:

①针对 T 中的模型描述结构,定义每类结构的翻译规则;

②在语法树的进出节点处理程序中,以翻译规则为基础,对语法树的每类节点进行翻译;

③根据语法树的遍历顺序,将翻译规则在进节点或出节点函数中实现,以达到一次遍历即输出目标语言的目的。

3.1 预处理

为了使得 T2N 工具能够识别出 T-VEC 程序中的单词,从 T-VEC 文件到 T2N 工具输入文件之间需要经过预处理过程。预处理过程通过 python 脚本处理的方式完成,主要分为 3 个步骤:首先,将变量与类型定义、函数、条件表、事件表等表格所单独生成的描述文件合并为一个描述文件,使程序在一个完整的主模块中执行,以保证程序的一致性;然后,去掉注释行和文件路径等无用代码,精简待转换的文件;最后,分离连接在一起的词法描述符,使 T2N 工具能够准确地识别出单词符号的性质。

3.2 词法和语法处理

本文借助 antlr 工具完成对词法和语法的处理。antlr 是一个强大的语言解析器生成器,用于读取、处理、执行或翻译结构化文本文件或二进制文件,它被广泛用于构建语言、工具和框架。antlr 中分析对应语言的词法和语法的程序分别叫作 lexer 和 parser。默认情况下,antlr 会构造一棵对应解析语言的语法分析树,其中,语法树的叶子是输入的单词,叶子的上级结点是包含其孩子结点的词组名,线性的句子是语法树的序列化。antlr 生成的是递归下降解析器。本文应用的是 antlr 监听者的方法,采用全自动化的遍历方式,主导深度优先遍历过程。

为了使得 antlr 产生对 T-VEC 语言的词法和语法分析程序,首先需要得到 antlr 的输入文件,即对应的 T-VEC 语言的 g4 格式的语法文件。因此,从 T-VEC 语言的介绍和语言范例中提取 antlr 的语法,该语法由一个头部定义 grammer T-VEC 和一系列可以相互引用的语言规则组成。其中,语法规则采用扩展巴克斯-诺尔范式(EBNF)的形式表示,处理过程中需要去除间接左递归、安排优先级;词法规则采用正则表达式的形式表示。然后,将 T-VEC 语法文件输入 antlr 进行错误检查和解析。最终,得到正确的 T-VEC 语法的词法、语法分析,以及遍历器等 java 程序。

3.3 转换过程

antlr 能够完成对 T-VEC 语言的词法和语法的解析,生成语法分析树,并为语法树的每一个结点构建进出结点函数。为实现模型的翻译,首先需要定义源语言 T-VEC 中每类结构到目标语言 XMV 中相应结构的转换规则。一个 T-VEC 程序主要由 3 个部分构成:类型和变量的定义、控制变量状态改变的约束条件定义、模型功能定义,即变量取值变化与约束条件之间的对应关系。

T-VEC 和 NuXMV 存在许多共有的变量类型,对这些变量类型可直接进行翻译,扩大 T-VEC 中少数不一致的变量类型可等价转换为 NuXMV 中的类型。NuXMV 不支持用户自定义类型,T-VEC 中的自定义类型均以基本类型定义为基础,在翻译的过程中,将用户自定义类型拆分成基本的变量类型,并转换为 NuXMV 基本变量类型的定义。T-VEC 中的基本类型包括布尔型、字符型、不同精度的浮点型、不同范围的整型、不同范围的无符号数、字符串类型等。NuXMV 支持布尔型、字型、无符号和有符号字型等,在翻译中,将 T-VEC 和 NuXMV 共有的类型进行直接转换;将 T-VEC 中特有的浮点型等翻译为 NuXMV 中的 32 位带符号字型,将字符串型翻译为枚举型。另外,考虑到实际应用中极少情况会用到字符型,因此不对其进行翻译。数据类型的范围问题在具体转换过程

中比较困难，因此，转换后的数据均取最大范围。

在 T-VEC 中，控制变量取值变化的约束条件通过逻辑谓词的组合形式来定义，以唯一的标识符来表示，其涉及与或非关系、变量赋值、变量取值的比较等。在语义上，约束条件定义与 NuXMV 中的 DEFINE 定义等价，DEFINE 在 NuXMV 中的作用相当于宏定义，“:=”左边的标识符代表“:=”右边的表达式，表达式也采用逻辑谓词的形式表示。当定义的标识符出现时，标识符在语法上会被它关联的表达式替换。由于 T-VEC 表示的约束条件较 NuXMV 中的 DEFINE 定义有许多多余的表示，并且存在谓词符号不一致的情况，因此在转换时，通过删除多余表示和统一谓词符号的方式来解决上述问题，二者可以进行转换。翻译时，将其转换为 NuXMV 中的 DEFINE 定义。

模型功能定义体现了系统功能，语义上体现为：当相应约束条件的取值为真时，变量取相应的变化结果值，其中的约束关系由第二部分的标识符来表示，变量在第一部分中进行定义。模型功能定义和 NuXMV 中的 assign_constraint 都体现了变量取值的变化，assign_constraint 中的 init 和 next 都体现了逻辑关系约束。逻辑上二者等价，且可相互转换。翻译时，将模型功能定义转换为 NuXMV 中的 assign_constraint。

完成 T-VEC 语言结构的转换规则后，实现转换规则。首先，将 T-VEC 的 g4 语法文件作为输入文件输入到 antlr 中，antlr 会自动为语法文件的每个标识符生成其在语法树中对应的结点的进出函数。然后，将转换规则作为相应结点进出函数的功能来实现对每个结点对应结构的翻译，其中，antlr 自带许多对结点操作的辅助函数，包括计算结点的孩子结点个数的函数、判断当前结点父节点的函数、增加同类结点的函数等，这些函数辅助翻译的实现。最后，通过遍历语法树，实现目标语言文件的生成。

为了得到正确的目标语言程序，必须在语法树的遍历中对遍历路径上结点的进出函数的调用顺序进行控制。为此，本文提出的方法是：叶子结点总是在最后被遍历，如输入程序中的变量名等，因此，忽略其进出的先后问题；语法树上的非终端结点表示在翻译过程中会重复调用或递归调用的结点，如逻辑表达式、约束关系等，因为这些结点的详细信息要在遍历其孩子结点之后得知，且一次翻译中不仅被调用一次，因此，在出结点时进行翻译，并将每一次翻译的内容存储下来；程序中需要转换的关键字，在其最大程度上的非终端结点的进结点时翻译，因为此时如果在出结点时翻译，可能会打乱输出语言的顺序；最后，根节点的翻译在出结点时进行。

4　T2N 原型工具的实现

上一节主要介绍了 T2N 工具的理论框架，包括需要借助 antlr 进行的词法处理、语法处理，以及翻译的实现过程。本节将具体介绍实现的细节，对转换过程中针对源语言到目标语言的各类模型结构的转换规则以及设计的算法结构进行详细说明。

4.1　转换的硬件平台与语言

转换工具是在 Windows 平台上开发的，是以 Eclipse 为开发环境，用 Java 开发的一个命令行工具，借助词法和语法解析的 antlr 工具。首先，将 T-VEC 工具语言的语法描述成 g4 格式的语法文件，作为 antlr 语法解析器的输入，antlr 将对输入的语法文件进行解析，生成对应语法文件的抽象语法树，以 Java 文件的形式表示；然后，采用 Java 语言编写代码，以自上而下的形式遍历语法树，对于抽象语法树中的每一个节点，antlr 将自动生成进、出结点两个函数，转换工具继承这些函数，在函数体中完成相应语法结构的转换，遍历转换后的语法树，输出即为目标语言。源语言即 T-VEC 工具对应的 T-VEC 语言，目标语言是 NuXMV 的描述语言 XMV。

4.2　转换规则定义

在 T-VEC 中，对于除常量、自定义类型和输入变量以外的其他结构中的变量，都采用单独子系统对其定义。为了转换方便，将 T-VEC 翻译成 XMV 时，只定义主模块 MODULE main，在主模块中描述系统性质。针对 T-VEC 中的 8 种基本结构，下面将具体分析每种结构对应的转换规则。

4.2.1　类型的转换

T-VEC 语言支持基本类型的定义，包括布尔型、枚举型、整型、实数型和基于基本类型的用户自定义类型。由于 NuXMV 不支持用户自定义类型，因此将用户自定义类型转换为基本的布尔型、枚举型、实数型或整型的子集的定义。将由 VARIABLE 标注的变量定义转化为 VAR：identifier name：var_type 的形式，其中 identifier name 代表变量的名字，var_type 代表转换后的基本类型。另外，在 NuXMV 中要求为每一个声明的变量设置初始值，因此在完成变量转换后，需要为每一个变量设置初始数值。变量转换规则的对应关系如表 1 所列。

表 1　变量转换规则

Table 1　Variable conversion rules

T-VEC 类型	取值范围	XMV 类型	取值范围
BOOLEAN	TRUE/FALSE	boolean	true/false
INTEGER	$-2^8\cdots2^8-1$	signed word[32]	$-2^{31}\cdots2^{31}-1$
UNSIGNED	$0\cdots2^n-1$	Unsigned[32]	$0\cdots2^{31}-1$
FLOAT32	$-2^{16}\cdots2^{16}-1$	signed word[32]	$-2^{31}\cdots2^{31}-1$
FLOAT64	$-2^{32}\cdots2^{32}-1$	signed word[32]	$-2^{31}\cdots2^{31}-1$
STRING	—	枚举型	—
declared_type	—	基本类型	—

4.2.2　常量和变量的转换

T-VEC 中的常量定义了系统中变量的唯一取值，输入变量对应于 SCR 方法中的监控变量，输出变量对应于 SCR 方法中的受控变量，在 NuXMV 中未对变量进行分类，且 T-VEC 程序中未显式地为变量赋初始值，而在 NuXMV 中须为每一个变量设置初始状态取值。在转换时，将 T-VEC 中的常量和变量统一转换为 NuXMV 中的变量定义，并采用 init 为每个变量赋初始值，常量的初始值即为其数值定义，布尔型变量的初始值置为 false，数值型变量的初值设为 0，枚举型变量将第一个分量作为初始值。具体转换规则如下所示。

原：VARIABLE mOccupied ISA Boolean；

后：VAR mOccupied：boolean；ASSIGN init（mOccupied）:= FALSE；

4.2.3　中间变量的转换

T-VEC 中的中间变量（term）定义了输入变量、常量与其他中间变量之间的相互约束关系，除了基本的名称和类型等

定义外，还使用条件表或事件表的形式对约束关系进行描述。条件或事件通常表示成逻辑谓词的形式，转换时，翻译成 NuXMV 中的 DEFINE 约束关系，用标识符表示，相应约束条件下的中间变量取值翻译成 next(中间变量)的形式，当标识符代表的约束成立时，中间变量取对应的结果值。具体转换规则如下所示。

原：LOGIC STRUCTURE tCurrentLSOpt_1__LS(_mcStatus,_mOccupied,mcStatus__VAR,mOccupied)

CONSTRAINT mcStatus__VAR := mcStatus(_mOccupied,mOccupied,_mcStatus) AND NOT:(_mcStatus=occupied::mcStatus__Type) AND (mcStatus__VAR=occupied::mcStatus__Type);

RELATIONSHIP tCurrentLSOpt__OUT = _tCurrentLSOpt; RELEVANCE PREDICATE { DISJUNCTION {tCurrentLSOpt_NoEvent__LS};};

后：DEFINE tCurrentLSOpt_1__LS:= (mcStatus__VAR:=mcStatus)&(!_mcStatus=occupied)&(mcStatus__VAR=occupied);

ASSIGN next(tCurrent-LSOpt__OUT):=case

tCurrentLSOpt_1__LS:_tCurrentLSOpt TRUE:(tCurrentLSOpt__OUT);esac;

4.2.4 函数与断言的转换

在 T-VEC 中，函数(function)用于描述系统自定义的函数，包括对函数名、函数类型和基于函数取值的条件表的定义。断言(assertion)表示对输入变量取值的限制，如对自动提款机系统中存钱和取钱的数值限制在 100 及 100 的倍数之间。在转换到 NuXMV 时，将其转换为 XMV 中的宏定义，同样使用 DEFINE 和 next(函数名)的形式表示。与其他变量取值唯一不同的是，断言的取值结果只能为 TRUE。

4.2.5 模式类的转换

模式类刻画了系统的状态迁移，在 T-VEC 中使用模式转换表来表示系统不同模式之间的转化关系。模式转换前后时刻的模式分别用 source 和 destination 来表示，模式转换的条件用 guard 来表示。guard 转换到 XMV 中时，同样用 DEFINE 来表示，source 采用系统模式类的当前取值表示，destination 对应 guard 取值为真时模式类的 next 值。guard 定义的条件较之前复杂，转换为 XMV 时主要涉及以下 3 种表达式的转换：1)非逻辑的转换，将 T-VEC 中的 NOT 转换为 NuXMV 中的!；2)函数的转换，将约束条件中的函数转换为相应的谓词逻辑表达式；3)删去变量取值表达式中的自定义类型定义。具体转换规则如下所示。

原：LOGIC STRUCTURE mcStatus_FROM_occupied_NoChange__LS (_mcStatus,_mOccupied,mOccupied)

CONSTRAINT (_mOccupied=TRUE)AND((_mcStatus=occupied::mcStatus__Type)AND(_mOccupied=mOccupied));

RELATIONSHIP mcStatus__OUT=unoccupied;RELEVANCE PREDICATE {DISJUNCTION { mcStatus_FROM_unoccupied_NoChange__LS };};

后：DEFINE mcStatus_FROM_occupied_NoChange__LS:=(_mOccupied=TRUE)&((_mcStatus=occupied)&(_mOccupied=mOccupied));

ASSIGN next(mcStatus__VAR):= case mcStatus_FROM_unoccupied_NoChange__LS:

unoccupied;TRUE:(mcStatus__VAR);esac;

4.2.6 输出变量的转换

输出变量代表了 SCR 方法中的受控变量，是系统功能的体现，定义了输入变量、中间变量、模式类等 T-VEC 其他结构之间的函数关系。输出变量的定义和初始化在变量的统一定义中进行，变量取值结果的改变由事件表或条件表描述，转换为 NuXMV 时，仍然采用 DEFINE 和 next(输出变量)组合的形式。

4.3 转换实现算法

由于 antlr 是基于 java 编程的工具，因此，在转换规则的实现上，使用 java 来实现。在常量和变量定义、约束关系定义和模型功能定义的转换算法中，输入都是由 T-VEC 生成的需求模型描述文件，名称为系统名，以.SS 结尾，输出到由用户命名的同一个 NuXMV 文件中，以.xmv 结尾。

原：LOGIC STRUCTURE cWallLights_NoEvent__LS(_mcStatus,_mFMOverride,_mOccupied,_mWallLights,mcStatus__VAR,mFMOverride,mOccupied,mWallLights)

CONSTRAINT mcStatus__VAR := mcStatus(_mOccupied,mOccupied,_mcStatus) AND((_mWallLights=mWallLights) AND (_mFMOverride=mFMOverride)) AND(_mcStatus=mcStatus__VAR);

RELATIONSHIP cWallLights__OUT=_cWallLights;RELEVANCE PREDICATE {DISJUNCTION {cWallLights_NoEvent__LS};};

后：DEFINE mcStatus_FROM_occupied_NoChange__LS:=((mOccupied=TRUE)&(_mcStatus=occupied));

ASSIGN next(mcStatus__OUT):=case mcStatus_FROM_unoccupied_NoChange__LS:unoccupied;TRUE:mcStatus__OUT;

esac;

常量和变量转换算法的伪代码如算法 1 所示。第 1－4 行为变量赋值，用于赋值的数据在解析 T-VEC 语言文件时收集，其中，conVar(第 1 行)和 varVar(第 2 行)分别代表收集的 T-VEC 中的常量和变量结构定义，包括名称(name)、类型(type)、取值范围(range)等；由于 T-VEC 中的变量类型采用传递定义的形式，因此，firstDefine(第 3 行)用于记录基本变量类型的定义，包括名称(name)、类型(type)、取值范围(range)等；flag(第 4 行)为转换过程中需用到的标志位，赋初值为 0。

对于收集到的每个常量，按照之前定义好的转换规则进行转换(第 6 行)，并将结果写入到 NuXMV 文件中(第 7 行)。对于每个变量，第 10—14 行用于判断其名字是否包含在用于记录基本变量类型的结构中，如果包含，则证明此数据项是基本变量类型定义，而非变量定义，不进行转换；否则(第 15 行)，说明该数据项是变量定义，按照定义好的转换规则转换并写入 NuXMV 文件中(第 16－17 行)。

算法 1 常量和变量转换算法

Input：T-VECname.SS

Ooutput：Nuxmvname.xmv

1. ConVar←getConVar(T-VECname.SS)
2. varVar←GetvarVar(T-VECname.SS)
3. firstDefine←GetFirstDefine(T-VECname.SS)
4. flag←0
5. For each c ∈ conVar do

```
6. c′←TRANSFORM(c)
7. Writefile(Nuxmvname. xmv,c′)
8. End
9. For each v ∈ varVar do
10. flag←0
11. For each f ∈ firstDefine do
12. If f is v. name then
13. flag←1
14. End
15. End
16. If flag is 0 then
17. v′←TRANSFORM(v)
18. Writefile(Nuxmvname. xmv,v′)
19. End
20. End
21. Return Nuxmvname. xmv
```

约束条件转换的算法如算法 2 所示。enterCon(第 1 行)和 deleteCon(第 2 行)分别代表 T-VEC 输入文件中的约束关系集合和约束关系表示中只在 T-VEC 中而不在 NuXMV 中的待删除的部分。E(第 3 行)表示用于存储转换后约束关系的集合，赋初始值为空。转换时，首先针对每条约束关系(第 4 行)，判断其中是否含有需要删除的部分，如果存在(第 6 行)则删除(第 7 行)。在此算法中，删除操作采用替换来实现，即使用空字符串来代替需要删除的字符串。接下来，对约束关系中的谓词进行替换，将 T-VEC 表示非的 NOT 替换为 NuXMV 中的!(第 11 行)。最后，将转换后的约束关系存储到定义好的约束关系集合中(第 13 行)并写入文件(第 14 行)。

算法 2　约束条件转换算法

```
Imput:T-VECname. SS
Outpu:Nuxmvname. xmv and E
1. enterCon←GetEnterCon(T-VECname. SS)
2. deleteCon←getDeleteCon(T-VECname. SS)
3. E←Ø
4. For each e ∈enterCon do
5. For each d ∈deleterCon do
6. If isContraint(e,d) then
7. REPLACE(d,"")
8. End
9. End
10. If "NOT:" in e then
11. REPLACE("NOT:","!")
12. End
13. E←E∪e
14. Writefile(Nuxmvname. xmv,e)
15. End
16. Return Nuxmvname. xmv,E
```

模型功能定义转换算法如算法 3 所示。outVar(第 1 行)是输入文件中在各约束条件下的输出变量取值集合，constraint(第 2 行)是转换好的约束关系。constraint 集合中的每个约束关系(第 3 行)都对应了输出变量集合 outVar 中(第 4 行)的一个结果，按照定义好的模型功能定义转换规则，转换成 NuXMV 的形式，并将结果写入 XMV 文件中(第 5—6 行)。

算法 3　模型功能定义转换算法

```
Input:T-VECname. SS and E
Output:Nuxmvname. xmv
1. OUTVar←getOutVar
2. constraint←E
3. For each c∈constraint do
4. For each o∈outVar do
5. Writefile(Nuxmvname. xmv,c)
6. Writefile(Nuxmvname. xmv,o)
7. End
8. End
```

5　实例分析

本节对需求工程中的典型案例——灯光控制系统[1]进行需求建模和转换，同时给出了实验的验证结果与分析。

5.1　灯光控制系统的描述与建模

为了对模型转换过程进行详细说明，并对转换的正确性进行验证，下文以灯光控制系统为案例进行分析。

5.1.1　灯光控制系统介绍及选择依据

灯光控制系统主要控制办公室和走廊灯光的明暗程度在适当的范围内。每个办公室内，在墙壁和窗户上各设有一组照明系统。灯光控制系统可能有两种控制灯光的模式，一种是用户自主选择的灯光控制模式，另一种是默认的灯光控制模式。设备管理员可以通过操作强制操作按钮来控制空办公室内灯光的开与关。除了灯光控制之外，系统必须考虑自然条件下的阳光强度对系统整体灯光强度及控制的影响。

对模型转换过程的验证主要分为两部分。在第一部分的验证中，要求作为案例的系统能全面地体现变量间的约束关系。灯光控制系统中对灯光强度的控制随时受到环境中自然光线的影响。系统需要外部环境中的光照强度作为参考，反过来系统的控制又会影响外界环境的明暗，这体现了系统变量之间的约束关系。在第二部分安全性质的验证中，要求作为案例的系统能体现系统模式的变迁，且系统变量间有紧密的联系，使得在验证 LTL 公式时，可以体现一个变量的改变对系统中其他变量带来的不安全因素。在灯光控制系统中，具有表示空间状态的模式类，当房间状态不存在变化过程时，系统都将处于不安全状态。另外，灯光控制系统的规模较为合适，作为日常生活中的场景，可以很好地被读者所接受。综上所述，灯光控制系统是较为适合作为验证案例的系统。

5.1.2　系统建模过程

本节详细介绍使用 T-VEC 工具对灯光控制系统建立需求模型的过程，包括系统中的变量提取和变量间关系提取这两个过程。

(1)变量提取

变量提取主要选择那些与系统直接相关联的环境变量(即 SCR 方法中的监控变量与受控变量)以及间接改变系统行为的变量(如 SCR 方法的中间变量等)。

在确定系统中的各类变量前，首先对系统中的变量类型进行说明。在灯光控制系统中，除基本的数据类型外，许多自定义类型被用于更准确地定义系统变量，表 2 列出了这些自定义类型的名称和对应的基本类型。其中，yLightLevel 代表

系统中的灯光强度；yLight 表示设置在墙壁和窗户上的灯光的启动状态；yOption 用于控制照明设备的选择。

表 2 灯光控制系统自定义类型

Table 2 Custom types of light control system

名称	基本类型
yLightLevel	integer[0,10000]
yLight	enum{off,on}
yTimer	integer[0,30]
yOption	enum{both,wall,window}

从系统与外界环境交互的角度出发，灯光控制系统的主要功能是通过用户自定义和系统默认设置两种模式来控制办公室、走廊等外界环境的光照强度，因此提取出监控变量 mDefLSVal 和 mChosenLSVal，两者分别代表在默认设置模式下和用户自定义模式下灯光控制系统的灯光强度。变量 mWallLights 和 mWindowLights 分别代表墙壁和窗户灯光控制开关的状态。为了表明模式的选择，使用监控变量 mDefLSOpt 和 mChosenLSOpt 代表默认和自定义模式的选择情况。设置变量 mOccupied 代表房间的空闲状态。系统的设备管理员可以对灯光进行重置，因此，提取开关变量 mFMOverride 表示重置开关的状态。用 mIndoorLL 代表自然灯光强度。系统受到外界环境中变量影响的同时，也通过对灯光强度的控制影响着外界环境变量。cWallLL，cWindowLL，cWallLights，cWindowLights 4 个受控变量分别代表经灯光控制系统控制的墙壁灯光强度、窗户灯光强度、墙壁灯光的开启状态、窗户灯光的开启状态。具体监控变量和受控变量的信息如表 3、表 4 所列，包括变量的名称、类型、初始值和简要描述。

表 3 灯光控制系统监控变量

Table 3 Monitoring variables of light control system

名称	类型	初始值	简要描述
mOccupied	BOOLEAN	FALSE	True when office is occupied
mFMOverride	BOOLEAN	FALSE	True if Fac. Mgr Override
mWallLights	yLight	off	On/off status of wall lights
mWindowLights	yLight	off	On/off status of window lights
mDefLSVal	yLightLevel	100	Default ambient light level
mChosenLSVal	yLightLevel	200	Chosen ambient light level
mDefLSOpt	yOption	wall	Default ambient light option
mChosenLSOpt	yOption	wall	Chosen ambient light option
mIndoorLL	yLightLevel	0	Level of nat. light in office

表 4 灯光控制系统受控变量

Table 4 Controlled variables of light control system

名称	类型	初始值	简要描述
cWallLL	yLightLevel	0	Intensity level of wall lights
cWindowLL	yLightLevel	0	Intensity level of window lights
cWallLights	yLight	off	On/off status of wall lights
cWindowLights	yLight	off	On/off status of window lights

中间变量通常用于对受控变量进行分析和计算，使得对控制受控变量改变的条件和事件的描述更加简洁、清楚。表 5 列出了灯光控制系统建模中使用的 4 个中间变量。tCurrentLSVal 和 tCurrentLSOpt 分别代表了系统当前需要的灯光强度和当前负责系统照明的设施。中间变量 tRemLL 代表在自然光强度的基础上，为达到要求的照明强度，系统仍需要的灯光强度。tOverride 表示系统已经被按下重置键。

表 5 灯光控制系统的中间变量

Table 5 Intermediate variables of light control system

名称	类型	初始值	简要描述
tOverride	BOOLEAN	FALSE	True if Fac. Mgr. pushed override
tCurrentLSVal	yLightLevel	0	Current desired light level
tCurrentLSOpt	yOption	wall	Current desired light distribution
tRemLL	yLightLevel	0	Lighting produced by artificial light

(2)关系提取

关系提取是通过基于受控变量的表格描述来表达系统需求的过程。在 SCR 方法中，使用 REQ 关系来表达变量之间的关系，包括模式转换表、条件表、事件表等表格形式。

在灯光控制系统中，4 个受控变量的取值都与空间的空闲状态有关，可以看作是空间状态的函数，因此，定义系统模式类 mcStatus 来表明系统的空闲状态，包括占有(occupied)、暂时空闲(temp_empty)和空闲(unoccupied) 3 个模式取值。模式类 mcStatus 随着监控变量 mOccupied 和 mT3 取值的不同而有不同的模式值，表 6 是模式类 mcStatus 新老模式之间的模式迁移表，初始值为空闲模式。

表 6 灯光控制系统模式迁移表

Table 6 Migration table of light control system mode

Mode Class=mcStatus		
Old Mode	Event	New Mode
unoccupied	@T(mOccupied)	occupied
occupied	@F(mOccupied)	temp_empty
temp_empty	@T (NOTmOccupied)	unoccupied
	@T(mOccupied)	occupied

完成系统中的变量和模式类定义后，对系统中的 REQ 关系进行表示，即定义灯光控制系统中 4 个受控变量与监控变量、中间变量和模式类之间的函数关系。在灯光控制系统中，cWallLights 和 cWindowLights 表示墙壁和窗户两组灯光的开关状态，其取值与空间的空闲状态、mWallLights、mFMOverride、前一时刻的开关状态有关。表 7 的事件表描述了这一函数关系。其中第 2 行表示当空间模式由空闲变为占用时，系统墙壁灯光开启；第 5 行表明当空间模式为非占用状态时，管理员按下重置开关，系统墙壁灯光关闭。其他情况下，灯光的开关状态保持不变。

表 7 cWallLights 定义事件表

Table 7 Event table of cWallLights define

Event	$cWallLights'$
@T(mWallLights=on) WHEN cWallLights=off OR @T(mcStatus=occipied)	on
@T(mWallLights=off) WHEN cWallLights=on OR @T(mcStatus=unoccupied) OR @T(mFMOverride) WHEN mcStatus occupied	off

5.2 实例验证与分析

本节在使用 T-VEC 工具对灯光控制系统建立需求模型的基础上，介绍运用 T2N 工具将需求模型转换为 NuXMV 模型的转换步骤、系统安全性质的提取和验证，以及对验证结果的分析工作。该实验运行在 64 位操作系统上，硬件配置为 Intel(R)Core(TM)i7-6700HQ CPU@ 2.60 GHz 2.60 GHz，8.00 GB 内存。

5.2.1　模型转换

按照第 3 节中的转换步骤将 T-VEC 工具生成的灯光控制系统需求模型描述的 T-VEC 文件转换为 NuXMV 模型。其中有输入变量的转换，T-VEC 转换到 XMV 时增加了对变量赋初始值；模式类 mcStatus 的 T-VEC 描述语言转化为 XMV 形式的逻辑条件约束，二者等价；输出变量 cWallLights 事件表的转换，其中转换到 XMV 中的约束条件在逻辑约束关系部分定义。

下面将对生成的 XMV 程序进行 LTL 安全性质的添加，进而将其输入到 NuXMV 模型检测器进行验证。

5.2.2　安全性质提取

安全性质是系统运行过程中必须满足的规范，其保证了系统的正确性与安全性。本文从灯光控制系统中提取了系统正确运行时所要满足的安全性质，并添加到 T2N 工具生成的 XMV 文件中，同时输入到模型检测器 NuXMV 中对安全性质的满足性进行证明，进而验证 T2N 工具对模型的正确转换，以进一步对系统的安全性进行验证。

安全性质的提取主要从两个方向出发：1）对期待结果为真的性质的验证，主要从系统功能出发，系统模式变迁是系统功能中最重要的部分。在灯光控制系统中，最主要的模式类是房间内的空闲状态。为此，选择验证性质：房间内有人时，在灯光控制系统的运行情况下，未来某一时刻，系统空间状态会变为占有状态。以此检验 T2N 工具对模式迁移表转换的正确性以及对系统需求模型的安全性。2）选择期待结果为假的性质进行验证。模型检测不仅可以验证模型的正确性，而且在模型不正确或验证性质出错时给出了系统反例。通过光照强度的大小来控制灯光开关的开启与关闭是灯光控制系统的重要功能。因此，选取性质：空间内控制墙壁灯光的开关一直关闭。以此检测事件表墙壁开关（cWallLights）的状态变化，若性质验证通过，则表明模型转换错误，墙壁开关的状态未发生变化；若性质验证失败，并给出反例，则通过进一步分析反例来确定模型转换结果的正确性，进一步确定需求模型的安全性。

5.2.3　安全性质验证与结果分析

根据 5.2.2 节提取的安全性质，本节主要将安全性质表示成 LTL 公式的形式，并将 LTL 表示的性质添加到模型转换的 XMV 输出文件中，以进一步在 NuXMV 中进行验证，通过验证结果，分析模型转换的正确性和需求模型的安全性。

将从模式迁移表中提取的期待结果为真的安全性质（房间内有人时，灯光控制系统运行的未来某一时刻，空间将变为占有状态）表示成 LTL 公式为：LTLSPEC G((mOccupied＝TRUE)→F(mcStatus__OUT＝occupied))。从事件表墙壁开关（CWallLights）中提取的期待结果为假的安全性质（空间内控制墙壁灯光的开关一直关）体现的是系统所有将来的状态，因此，使用时态连接词 G，用 LTL 公式表达为：LTLSPEC G(cWallLights__OUT＝off)。

将上述两条安全性质添加到模型转换后的灯光控制系统 XMV 文件中，并输入 NuXMV 中进行验证。其中，对期待结果为真的性质的验证结果如图 3 所示。由最后一行可以看出，在系统将来运行的某一时刻，空间状态将变为占有状态。

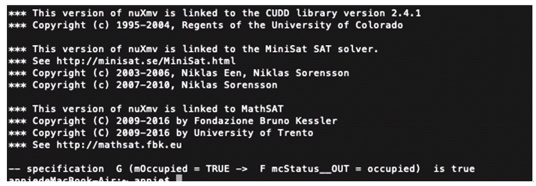


图 3　正例验证结果

Fig. 3　Positive example validation results

（1）输入变量转换

原：VARIABLE mOccupied ISA Boolean；

后：VAR mOccupied：boolean；ASSIGN init（mOccupied）：＝FALSE；

（2）模式类约束关系 T-VEC 程序

LOGIC STRUCTURE mcStatus_FROM_occupied_NoChange__LS（_mcStatus，_mOccupied，mOccupied）

CONSTRAINT（_mOccupied＝TRUE）AND（（_mcStatus＝occupied::mcStatus__Type）AND（_mOccupied＝mOccupied））；

RELATIONSHIP mcStatus__OUT＝unoccupied；RELEVANCE PREDICATE｛DISJUNCTION｛ mcStatus _ FROM _ unoccupied _ NoChange__LS｝；｝；

（3）模式类约束关系 XMV 程序

DEFINE mcStatus _ FROM _ occupied _ NoChange _ _ LS：＝（_mOccupied＝TRUE）&（（_mcStatus＝occupied）&（_mOccupied＝mOccupied））；

ASSIGN next(mcStatus__VAR)：＝ case mcStatus_FROM_unoccupied_NoChange__LS：

unoccupied；TRUE：(mcStatus__VAR)；esac；

（4）cWallLights 事件表转换

原：LOGIC STRUCTURE cWallLights_NoEvent__LS(_mcStatus，_mFMOverride，_mOccupied，_mWallLights，mcStatus__VAR，mFMOverride，mOccupied，mWallLights)；

CONSTRAINT mcStatus _ _ VAR：＝ mcStatus（_mOccupied，mOccupied，_mcStatus）AND((_mWallLights＝mWallLights)AND（_mFMOverride＝ mFMOverride）) AND（_ mcStatus ＝ mcStatus _ _ VAR)；

RELATIONSHIP cWallLights__OUT＝_cWallLights；RELEVANCE PREDICATE｛DISJUNCTION｛cWallLights _ NoEvent _ _ LS｝；｝；

后：DEFINE mcStatus _ FROM _ occupied _ NoChange _ _ LS：＝((mOccupied＝TRUE) &（_mcStatus＝occupied）)；

ASSIGN next(mcStatus__OUT)：＝case mcStatus_FROM_unoccupied_NoChange__LS：unoccupied；TRUE：mcStatus__OUT；

esac；

对期待结果为假的性质的验证结果如图 4 所示。从图 4 的第一行中可以看出，期待结果为假的性质的验证结果为假。在第 3 个状态中，当系统此刻的空间模式（mcStatus_OUT）为占有（occupied）且系统管理员未按下重置开关（mFMOverride＝FALSE）时，墙壁开关（cWallLights__OUT）的状态变为开启（on），即墙壁开关（cWallLights__OUT）的取值不可能一直处于关闭状态（off）。通过以下状态变迁，可以看出墙壁开关变量的取值随空间模式和系统重置开关的状态正确变化，从而证明了墙壁开关事件表模型转换的正确性。期待结果为假的性质为假，表明控制墙壁灯光的开关不会一直处于关闭状态。

同时通过该验证结果也发现了回路等问题，体现了需求模型的不安全性。

```
-- specification  G cWallLights__OUT = off  is false
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
  -- Loop starts here
  -> State: 1.1 <-
    _mOccupied = FALSE
    _mcStatus = unoccupied
    mOccupied = FALSE
    mcStatus__OUT = unoccupied
    _cWallLights = off
    _mFMOverride = FALSE
    _mWallLights = off
    cWallLights__OUT = off
    mFMOverride = FALSE
    mWallLights = off
    mcStatus_FROM_occupied_NoChange__LS = FALSE
    mcStatus_FROM_occupied_TO_temp_empty__LS = FALSE
    mcStatus_FROM_temp_empty_NoChange__LS = FALSE
    mcStatus_FROM_temp_empty_TO_occupied__LS = FALSE
    mcStatus_FROM_temp_empty_TO_unoccupied__LS = FALSE
    mcStatus_FROM_unoccupied_NoChange__LS = TRUE
    mcStatus_FROM_unoccupied_TO_occupied__LS = FALSE
    cWallLights_1__LS = FALSE
    cWallLights_2__LS = FALSE
    cWallLights_NoEvent__LS = TRUE
  -> State: 1.2 <-
    mOccupied = TRUE
    _cWallLights = on
    _mFMOverride = TRUE
    _mWallLights = on
    mFMOverride = TRUE
    _mFMOverride = TRUE
    _mWallLights = on
    mFMOverride = TRUE
    mWallLights = on
    mcStatus_FROM_unoccupied_NoChange__LS = FALSE
    mcStatus_FROM_unoccupied_TO_occupied__LS = TRUE
  -> State: 1.3 <-
    _mcStatus = occupied
    mOccupied = FALSE
    mcStatus__OUT = occupied
    _mFMOverride = FALSE
    _mWallLights = off
    cWallLights__OUT = on
    mFMOverride = FALSE
    mWallLights = off
    mcStatus_FROM_unoccupied_TO_occupied__LS = FALSE
  -> State: 1.4 <-
    _mOccupied = TRUE
    _mcStatus = temp_empty
    _cWallLights = off
    _mWallLights = on
    mFMOverride = TRUE
    mcStatus_FROM_temp_empty_TO_unoccupied__LS = TRUE
    cWallLights_2__LS = TRUE
    cWallLights_NoEvent__LS = FALSE
  -> State: 1.5 <-
    _mOccupied = FALSE
    _mcStatus = unoccupied
    mcStatus__OUT = unoccupied
    _mWallLights = off
    cWallLights__OUT = off
    mFMOverride = FALSE
    mcStatus_FROM_temp_empty_TO_unoccupied__LS = FALSE
    mcStatus_FROM_unoccupied_NoChange__LS = TRUE
    cWallLights_2__LS = FALSE
    cWallLights_NoEvent__LS = TRUE
```

图 4 反例验证结果

Fig. 4 Counterexample verification results

结束语 针对基于形式化需求分析方法的工具没有模型检测的功能，无法完成系统安全性质验证的问题，本文在 SCR 方法的检验工具 T-VEC 的基础上，开发了将需求描述语言 T-VEC 转换为模型检测语言 XMV 的模型转换工具 T2N。在具体实现中，借助语言解析器生成器工具 antlr，对 T-VEC 语言的语法文件进行解析，生成语法分析树、词法和语法解析程序、语法树遍历程序。随后，定义具体的语言转换规则，并将规则添加到语法树的结点处理程序中，实现了模型转换的功能，然后，将预处理后的 T-VEC 模型描述程序输入到 T2N 工具中，输出模型检测程序 XMV，添加 LTL 表示的具体系统安全性质约束。最终将其输入到 NuXMV 中完成对需求模型的安全性验证。本文结合需求工程中的经典案例灯光控制系统进行分析。尽管 T2N 模型转换工具可以自动完成基本安全性的验证，但目前仍存在局限，如变量的取值范围问题、逻辑约束语法上的递归问题等，我们将会在接下来的工作中对这些问题进行研究，以提高现有工具的适用范围。在本文中，对系统中的安全性质提取需要人为分析完成，将来可以开发系统模型分析的工具，使其自动化地生成描述系统安全性质的 LTL 公式。

参 考 文 献

[1] HEITMEYER C, BHARADWAJ R. Applying the SCR Requirements Method to the Light Control Case Study[J]. Journal of Universal Computer Science, 2000, 6(7): 88-92.

[2] THAYER R, DORFMAN M. Software Requirements Analysis and Specifications [M]. Wiley-IEEE Press, 2000.

[3] GOODRUM M, CLELAND-HUANG J, et al. What Requirements Knowledge Do Developers Need to Manage Change in Safety-Critical Systems? [C]// Requirements Engineering Conference. IEEE, 2017: 90-99.

[4] JIN Y, PARNAS D L. Defining the meaning of tabular mathematical expressions [J]. Science of Computer Programming, 2010, 75(11): 980-1000.

[5] HEITMEYER C L, LEONARD E I. Obtaining trust in autonomous systems: tools for formal model synthesis and validation [C]// 2015 IEEE/ACM 3rd FME Workshop on Formal Methods in Software Engineering (FormaliSE). IEEE, 2015: 54-60.

[6] MARCINIAK, JOHN J. Encyclopedia of Software Engineering [M]// Encyclopedia of Software Engineering, 1994.

[7] HEITMEYER C L, JEFFORDS R D. Applying a Formal Requirements Method to Three NASA Systems: Lessons Learned [C]// Aerospace Conference. 2007: 1-10.

[8] HU J, SHI J J, CHENG H, et al. A four-variable model based system security modeling and analysis method [J]. Computer Science, 2016, 43(11): 193-199. (in Chinese)
胡军，石娇洁，程桢，等. 一种基于四变量模型的系统安全性建模与分析方法[J]. 计算机科学，2016，43(11)：193-199.

[9] PARNAS D L, MADEY J. Functional Documentation for Computer Systems Engineering (Version 2) [J]. CRL Report, 1991, 7(237).

[10] BLACKBURN M R, BUSSER R D. T-VEC: a tool for developing critical systems [C]// Proceedings of 11th Annual Conference on Computer Assurance. 1996: 237-249.

[11] CHANDLER R E. Model Checking[M]// Encyclopedia of Biostatistics 2005.

[12] BULTAN T, HEITMEYER C. Applying infinite state model checking and other analysis techniques to tabular requirements specifications of safety-critical systems[J]. Design Automation for Embedded Systems, 2008, 12(1-2): 97-137.

[13] KRÖGER F. Temporal Logic of Programs[C]// Etacs Monographs on Theoretical Computer Science. 1987.

[14] CAVADA R, CIMATTI A, DORIGATTI M. The nuXmv Symbolic Model Checker[C]// International Conference on Computer Aided Verification. 2014, pp: 334-342.

[15] CIMATTI A, CLARKE E, GIUNCHIGLIA F. NUSMV: a new symbolic model checker[J]. International Journal on Software Tools for Technology Transfer, 2000, 2(4): 410-425.

[16] BIERNACKA A, BIERNACKI J, SZPYRKA M. State-based verification of RTCP-nets with nuXmv[C]// AIP Conference Proceedings. 2015: 375-390.

[17] CHOI Y, HEIMDAHL M P E. Model Checking RSML-e Requirements[C]// IEEE International Symposium on High Assurance Systems Engineering. 2002: 109.