计算机科学
COMPUTER SCIENCE

飞行模式转换的RSML^{-e}到Lustre模型转换与验证方法

王智艺, 胡军, 徐恒

引用本文

王智艺, 胡军, 徐恒. 飞行模式转换的RSML^{-e}到Lustre模型转换与验证方法[J]. 计算机科学, 2025, 52(12): 48-59.

WANG Zhiyi, HU Jun, XU Heng. Transition and Verification Method from RSML^{-e} to Lustre Model for Flight Mode Transition [J]. Computer Science, 2025, 52(12): 48-59.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.250600027

飞行模式转换的 RSML^{-e}到 Lustre 模型转换与验证方法

王智艺　胡　军　徐　恒
南京航空航天大学计算机科学与技术学院　南京 211106
(371608761@qq.com)

摘　要　自动飞行系统是现代大型飞机飞行控制的核心系统,飞行导引系统作为自动飞行系统的核心子系统,管理和控制自动飞行系统的模式转换。自动飞行系统的不同飞行阶段本质上是飞行模式的转换,决定了飞机的飞行安全。然而,飞行模式转换具有耦合兼容的多维度复杂静态结构,以及交互合作和过渡切换的多层次模式动态组合的本质特征,因此保证飞行模式转换的正确性至关重要。基于模型驱动的软件建模方法将飞行模式转换需求建模为半形式化模型或形式化模型,从而分析和验证模型满足的性质。现有方法面临两方面挑战:1)自然语言需求到半形式化需求模型的建模,大多为手工建模,且不同建模语言建立的需求模型之间存在差异;2)半形式化需求模型无法直接进行模型检验,需将其转换为模型检验工具的输入模型,且不同的验证工具的验证效率也存在差异。为此,基于自动飞行模式转换的 RSML^{-e}需求模型,提出一种将 RSML^{-e}模型转换为 Lustre 同步数据流语言的系统性方法。首先,从数据类型、变量、逻辑短语、AND-OR 表、宏等多个维度构建映射规则,将 RSML^{-e}模型中元素逐一转换为 Lustre 同步数据流语言所支持的形式,并在描述安全性质时对变量的数量进行缩减;其次,模型转换后,将转换所得的 Lustre 模型及安全性质输入 Jkind 模型检验工具进行验证,基于 Jkind 模型检验工具内置的多种优化技术,较好地缓解了模型验证过程中状态空间爆炸的问题,实现了对大规模模型的高效验证;最终,通过该流程成功验证了自动飞行系统模式转换需求相关的安全性质,确保系统在各类工况下运行的可靠性。

关键词:形式化验证;RSML-e;Lustre;模型转换;安全性;自动飞行系统
中图分类号　TP311

Transition and Verification Method from RSML^{-e} to Lustre Model for Flight Mode Transition

WANG Zhiyi, HU Jun and XU Heng
School of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China

Abstract　The automatic flight system is the core system of flight control for modern large aircraft. The flight guidance system, as the core subsystem of the automatic flight system, manages and controls the mode transition of the automatic flight system. The different flight stages of an automatic flight system are essentially flight mode transition, which determine the flight safety of the aircraft. However, flight mode transition has the essential characteristics of a multi-dimensional complex static structure that is coupled and compatible, as well as a multi-level dynamic combination of interactive cooperation and transitional switching. Therefore, ensuring the correctness of flight mode transition is of crucial importance. The model-driven software modeling method models the flight mode transition requirements as semi-formal or formal models, thereby analyzing and verifying the properties satisfied by the model. The existing methods face two challenges: 1) The modeling from natural language requirements to semi-formal requirement models is mostly done manually, and there are differences among the requirement models established by different modeling languages; 2) The semi-formal requirements model cannot be directly used for model checking. It needs to be transitioned into the input model of the model checking tool, and the verification efficiency of different verification tools also varies. Based on the RSML^{-e} requirement model of automatic flight mode transition, this paper proposes a systematic method for transitioning the RSML^{-e} model into the Lustre synchronous data flow language. Firstly, mapping rules are constructed from multiple dimensions such as data types, variables, logical phrases, AND-OR tables, and macros. It transitions the elements in the RSML^{-e} model one by one into the forms supported by the Lustre synchronous data stream language, and reduces the number of variables when describing the safety properties. Secondly, after the model transition, the Lustre model and safety properties obtained from the transition are input into the Jkind model checking tool for verification. Based on the various optimization techniques built into the Jkind model checking tool, the problem of state space explosion during the model verification process is better alleviated, and

到稿日期:2025-06-04　返修日期:2025-09-10
基金项目:国家自然科学基金联合基金项目(U22412044)
This work was supported by the Joint Founds of the National Natural Science Foundation of China(U22412044).
通信作者:胡军(hujun.nju@139.com)

efficient verification of large-scale models is achieved. Ultimately, the safety properties related to the mode transition requirements of the automatic flight system are successfully verified through this process, ensuring the operational reliability of the system under various working conditions.

Keywords Formal verification, RSML^{-e}, Lustre, Model transition, Safety, Automatic flight system

1 引言

自动飞行系统(Automatic Flight System, AFS)，是指引飞机按照预先设定的程序或者根据环境条件选择合适的飞行模式，从而使飞机自动飞行的系统。自动飞行系统是辅助飞行员控制飞机的关键辅助工具，其不同飞行阶段本质上是飞行模式转换。飞行模式承担着人机交互接口的重要功能，它由自动飞行系统的核心子系统，即飞行导引系统(Flight Guidance System, FGS)控制。飞行模式转换系统的静态结构复杂，须兼顾多维度耦合兼容性，它的多层次模式可以进行交互合作及过渡切换的动态组合。飞行模式理论设计的错误，或者模式转换逻辑的混淆，都会导致系统危害或者灾难。根据美国航空安全报告系统(Aviation Safety Reporting System, ASRS)的统计数据，在与飞行系统自动化相关的事件审查中，20%的事故被归类为“模式转换问题”。因此，模式转换已经成为影响现代飞机自动飞行系统安全性的关键因素。

基于模型的软件开发(Model-Based Software Engineering, MBSE)是一种以抽象模型为核心的软件开发方法，它用形式化或半形式化的模型代替需求文档，并通过自动化转换生成代码或可执行系统。MBSE 强调使用领域特定建模语言或通用建模语言(如 UML 和 SysML)来定义系统，并通过模型转换减少手动编码，提高开发效率和系统可靠性。形式化方法[1]是基于严格数学基础，对计算机软(硬)件系统进行形式规约、开发和验证的技术。其中，形式规约使用形式语言构建所开发系统的规约，刻画系统不同抽象层次的模型和性质，如需求模型、设计模型等。形式验证是证明不同形式规约之间的逻辑关系，如需求模型是否满足安全性质规约。使用形式化方法可以检验出系统行为的不一致和不完整，从源头上保障了系统需求的正确性和可靠性，保证系统在运行过程中的安全性，降低系统维护成本。

针对已建立的需求模型，需要将其转换为模型检验工具接受的输入。模型检验(Model Checking)是一种形式化验证方法，用于自动验证系统模型是否满足给定的性质规约(通常表达为时序逻辑公式，如 LTL 或 CTL)。其核心思想是穷举系统所有可能的状态，检查是否存在违反安全性质的行为。

现有的形式化建模语言分为两类。一类是具有严格形式化定义的数学语言，如：Event-B[2] 主要用于描述在不同抽象层次上模块的功能性设计；ASM[3] 和 Statecharts[4] 等基于自动机的建模语言，主要适用于描述离散算法、协议和离散控制逻辑；进程代数 CSP[5] 和 CCS[6] 等主要描述进程间的并发交互。这类语言使用难度高，且缺乏面向工程的工具支持，在工程中应用较少。另一类是以 UML, SysML 和 AADL[7] 为代表的半形式化建模语言，具有直观、易于理解等优点，且有工具支持，在工程中应用较广。

在机载软件领域，20 世纪 90 年代末期，美国联邦航空管理局(FAA)使用需求状态机建模语言(RSML)成功地对民机空中防撞系统(TECAS)需求进行了完整的形式化规约描述[8]。之后，美国罗克韦尔-柯林斯公司和美国 NASA 使用修改版的 RSML^{-e}半形式化语言，首次对 FGS 中的飞行模式转换建立了形式化需求模型[9]。

相较于国外，国内对机载软件领域的形式化建模方法的研究和应用较少，但在一些安全攸关领域也出现了相关工作。例如，南京大学 Bu 等[10] 提出参数混成自动机建模复杂的信息物理融合系统，并利用在线模型检测技术提出故障预测机制；北京航空航天大学 Li 等[11] 采用 Petri 网模型对机载系统中的除冰软件系统进行了形式化建模；南京航空航天大学 Chen 等[12] 使用 Promela 语言对飞机襟缝翼控制系统中的控制单元软件代码进行了建模。

为了保证飞机的飞行安全，对飞行模式转换的需求模型进行形式化验证是必要的，它可以较早地发现需求规范中存在的错误，避免在系统开发后期因需求缺陷而进行大规模的修改和调整，重做大量的工作，进而大大增加开发的代价，延长系统的开发时间。

对飞行模式转换的需求模型进行形式化验证存在两个挑战。一是虽然半形式化语言所建立的模型具有良好的可读性和可理解性，是非专业人员和专业人员之间的有效沟通媒介，但其依赖人工手动建模，没有工具支持，不具有可执行性，建立出来的模型不能进行模型检验，需要转换为可验证的形式。而由于不同语言之间存在显著差异，模型之间的转换规则需要逐一构建。二是由于飞行模式转换复杂程度较高，在形式化验证时可能会出现“状态空间爆炸”的现象，且不同的验证工具的验证效率也存在差异，因此选择合适的验证方法和工具缓解“状态空间爆炸”问题也显得尤为重要。

对于以上挑战，现有的工作大多将模型转换为 NUSMV (New Symbolic Model Verifier)模型进行验证。NUSMV 是一个强大的模型检查工具，主要用于验证离散、并发和时序系统的属性。文献[13]使用 RSML^{-e}语言对自动飞行控制系统(AFCS)需求进行建模，提出一种将 RSML^{-e}模型转换成 NUSMV 模型的方法，并用 NUSMV 对模型的属性进行验证。文献[14]提出了从 AltaRica3.0 模型到 NuSMV 模型的转换规则及算法，使得在 AltaRica 中不能验证的时序属性在 NuSMV 中能得以验证。文献[15]研究了基于 NuSMV 的 AADL(Formalized Architecture Analysis and Design Language)模型形式化验证方法，提出由 AADL 到 NuSMV 的模型转换方法。文献[16]设计了从 SCR(Formalized Requirement model)模型到 NuSMV 模型的自动转换框架。针对不同的验证工具，文献[17]对比了 SDV(MATLAB Simulink Design Verifier)和 NUSMV 在飞行模式转换需求模型验证中的性能，结果显示 NUSMV 的效率要明显优于 SDV。文献[18]对比了 SCADE 模型检验器与 JKind，结果

表明，JKind 的效率更高。

本文基于 RSML^{-e} 语言建立的飞行模式转换需求模型，提出一种将 RSML^{-e} 模型转换为 Lustre 同步数据流语言的系统性方法。将飞行模式转换的需求模型从数据类型、变量、逻辑短语、AND-OR 表、宏等多个维度转换为 Lustre 同步数据流语言形式，并用 Jkind 模型检验工具对飞行模式转换需求模型进行形式化验证。严格检查飞行模式转换是否具有潜在的风险，提高自动飞行系统的安全性。同时，利用 Jkind 模型检验工具内置的多种优化技术，较好地缓解了在进行形式化验证时所出现的状态爆炸问题，实现了对大规模模型的高效验证。

本文对飞行模式转换需求模型的形式化验证主要包含两个部分。第一部分，将飞行模式转换需求的 RSML^{-e} 模型转换成用 Lustre 同步数据流语言描述的模型，并根据需求将安全性质用 Lustre 同步数据流语言描述出来。第二部分，将 Lustre 模型与 Lustre 安全性质结合，使用 Jkind 模型检验工具对飞行模式转换需求模型进行验证，从而验证需求的正确性。其完整流程如图 1 所示。

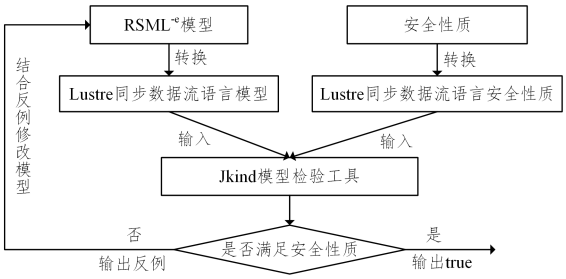

图 1 飞行模式转换需求模型的形式化验证流程

Fig. 1 Formal verification process of the flight mode transition requirement model

本文第 2 章对 RSML^{-e} 和 Jkind 模型检验工具以及 Lustre 同步数据流语言进行分析和介绍；第 3 章介绍飞行模式转换需求模型的转换与验证方法；第 4 章结合实例对所提出的方法进行验证；最后总结全文并展望未来。

2 背景知识

2.1 RSML^{-e}

需求状态机语言 RSML^{-e}[19] 是一种基于状态的规范语言，由美国明尼苏达大学关键系统组基于 RSML[20] 语言开发。该语言能够为过程控制系统的行为建模，同时解决了 RSML 语言中对显示事件的依赖性容易导致错误的问题。

一个 RSML^{-e} 模型包含输入接口（输入变量）、状态变量、输出接口（控制变量）、AND-OR 表、宏和常量。

2.2 RSML^{-e} 特征

1）数据类型

RSML^{-e} 语言支持的数据类型包括布尔型（Bool）、枚举型（Enum）、整型（Int）和实型（Real）。

2）默认初始值（Undefined）

RSML^{-e} 语言中的“默认初始值”机制允许在系统初始化阶段将特定变量显式声明为 Undefined 状态，既保留了系统状态空间的完整性，又能够以形式化方式表征传感器失效时环境信息的不可知性。

3）时序

RSML^{-e} 支持时序的概念，它以离散的形式表示系统，可以表示当前研究状态及其前驱状态。其中，前驱状态用 PREVIOUS STEP 关键字来表示，如 PREVIOUS STEP(变量 a)；当前研究状态则无需特殊修饰。

4）变量

（1）输入变量（输入接口）

输入变量为输入接口在模型中的具体表现，用于接收外部对系统的输入。输入变量分为两种：系统对外部环境的监测（MONITORED），如 FGS 中对控制面板上开关状态的监测；以及其他系统对本系统的输入（INPUT），其反映了其他系统的输出接口对本系统的影响，如 FGS 中另一侧 FGS 对本侧 FGS 系统的输入。输入变量的定义方式如图 2 所示。

变量名
Type：变量的数据类型
Initial Value：变量的初始值
Classified as：INPUT|MONITORED

图 2 输入变量的定义

Fig. 2 Definition of input variable

图 3 展示了导航（NAV）模式开关对 FGS 的输入。

NAV_Switch
Type：{OFF，ON}
Initial Value：Undefined
Classified as：MONITORED

图 3 输入变量 NAV_Switch

Fig. 3 Input variable NAV_Switch

（2）状态变量

状态变量是模型最关键的一环，描述了所建系统的具体行为，其定义方式如图 4 所示。

变量名
Parent：父变量 | None
Type：变量的数据类型
Initial Value：变量的初始值
Classified as：State
状态迁移

图 4 状态变量的定义

Fig. 4 Definition of state variable

图 5 所示的 NAV_Selected 变量描述了导航（NAV）模式在进入 Selected 状态后的状态变化。

父变量：RSML^{-e} 模型是一种层次化模型，其支持父变量的定义，当且仅当父变量满足点后的条件时（图 5 中，变量 NAV 值为 Selected），该变量才能进行状态迁移，否则该变量值保持其值为 Undefined 不变。▷符号前的变量表示父变量的父变量（图 5 中，NAV 的父变量为 Modes）。

状态迁移：状态迁移是状态变量的核心，它包含迁移和迁移条件两个部分。if 前的部分表示迁移，→左侧的值称为原状态，→右侧的值称为目标状态。if 后的部分表示迁移条件，若当前状态为原状态，并且满足迁移条件，则迁移到目标状态。当迁移中不含→，只包含变量的一个状态时，该状态为目

标状态，此时只要满足迁移条件，无论当前状态是什么，都迁移到目标状态。

NAV_Selected		
Parent：Modes. On ▷ NAV. Selected		
Type：{Armed，Active}		
Initial Value：Undefined		
Classified as：State		
：＝Offside_NAV_Selected if not Is_This_Side_Active		
Undefined→Armedif TRUE		
Armed→Activeif		
		OR
AND	Activate_NAV	T
	Is_This_Side_Active	T

图 5 状态变量 NAV_Selected

Fig. 5 State variable NAV_Selected

(3)控制变量(输出接口)

控制变量为输出接口在模型中的具体表现，用于系统对外部环境的输出，其定义方式如图 6 所示。

变量名
Parent：None
Type：变量的数据类型
Initial Value：变量的初始值
Classified as：CONTROLLED
控制条件

图 6 控制变量的定义

Fig. 6 Definition of controlled variable

图 7 所示的 Is_NAV_Selected 变量描述了导航(NAV)模式是否被选择(Selected)，并向其他系统传达这一信息。

Is_NAV_Selected
Parent：None
Type：BOOLEAN
Initial Value：FALSE
Classified as：CONTROLLED
：＝(NAV ＝Selected) if TRUE

图 7 控制变量 Is_NAV_Selected

Fig. 7 Controlled variable Is_NAV_Selected

5)逻辑短语

逻辑短语是 RSML^{-e} 中的重要组成部分，出现在变量的迁移条件和 AND-OR 表中。其为变量、常量、关系运算符(＞\＜\≥\≤\＝)、时序表达式(PREVIOUS STEP)和用于状态变量的 When 符号中的一种或多种的组合。其值为真(True)或假(False)，一般用 P 指代逻辑短语，not P 代表对其取反。

6)AND-OR 表

为了解决命题逻辑符号语言的复杂性问题，RSML^{-e} 使用 AND-OR 表来代替析取范式(DNF)。AND-OR 表格的最左一列列出了逻辑短语。其他的列是这些逻辑短语的连接，并且列出了逻辑短语的逻辑真值。规定只要有某一列的值为真，则整个表的值就为真。而每列的真值为真，当且仅当此列的真值与它们所关联的逻辑短语的真值都一致。例如，表 1 所列的 AND-OR 表等同于 DNF 范式(When_Nonbasic_Vertical_Mode_Activated＝true)∨(Modes ＝Off)。

表 1 AND-OR 表

Table 1 AND-OR table

		OR	
AND	When_Nonbasic_Vertical_Mode_Activated	T	.
	When(Modes ＝Off)	.	T

AND-OR 表的使用提高了模型的可读性，相较于代码而言，表格的形式使开发者或者测试者更易于发现规约中所犯的错误。

7)宏

为了对 AND-OR 表进行重用，以及方便对模型进行理解，RSML^{-e} 用宏来封装 AND-OR 表及布尔值，当需要使用 AND-OR 表及布尔值时，只需通过宏名进行引用即可。宏的具体形式如图 8 所示。

宏名
Condition：AND-OR 表\true\false

图 8 宏

Fig. 8 Macro

宏的使用提高了规范的可读性和可重用性。通过用宏封装 AND-OR 表，不仅可以提高规范的可读性，还有助于定位错误和跟踪变化。

2.3 Jkind

JKind 模型检验器(Jkind Model Checker)是一种针对 Lustre 同步数据流语言的模型检验工具[21-22]，是开源的工业级的模型检验器。JKind 使用多个并行的引擎对具有无限状态空间的模型进行属性验证，并在验证不通过时生成反例。其引擎结构图如图 9 所示。其中，有界模型检查(BMC)引擎执行转换关系的标准迭代展开以找到反例，并将其用作 k-归纳的基本情况[23-25]。BMC 引擎保证其找到的任何反例的长度都是最小的；k-归纳(k-Induction)引擎执行 k-归纳的归纳步骤，在过程中可能使用其他引擎生成的不变量；不变量生成(Inv Gen)引擎使用基于模板的不变量生成技术，使用它自己的 k 归纳循环[26]；属性定向可达性(PDR)引擎使用隐式抽象技术执行属性定向可达性[27-29]。与 BMC 和 k-归纳不同，每个属性都由不同的 PDR 子引擎单独处理。最后，建议(Advice)引擎根据 JKind 的先前运行生成不变量。

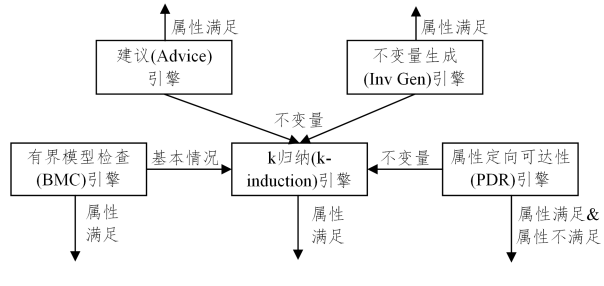

图 9 JKind 引擎结构图

Fig. 9 Structure diagram of JKind engine

JKind 在验证安全性质时采用增量式的方法逐步逼近正确结果，将有限步的归纳推广到无限行为，避免了一次性计算全部可达状态；并通过添加约束缩小搜索空间及对结果进行复用等方法提升效率。因此，在对规模较大的模型进行验证

时，JKind 可以有效缓解“状态空间爆炸”的问题。

2.4 Lustre 特征

1)数据类型

Lustre 语言支持的数据包括布尔型(Bool)、枚举型(Enum)、整型(Int)、实型(Real)、数组(Array)和结构体(Struct)。其中，结构体可以记录不同数据类型的元素，如 type r1=struct{a:bool;b:int}定义了一个可以存储一个布尔型和一个整型的结构体，r1.a 的取值范围为 true 或 false，r1.b 的取值范围为整数。

2)时序

Lustre 中也支持时序的概念，与 RSML^{-e}一样，同样包含当前研究状态及其前驱状态。在 Lustre 中，前驱状态用 pre 关键字来表示，如 pre(变量 a)；当前研究状态则无需特殊修饰。

3)流表达式

Lustre 中流表达式用箭头来表达，其计算结果在初始时刻取箭头左侧表达式的值，否则取箭头右侧表达式的值。如图 10 所示，描述了一个累加器。

```
node counter() returns(out1:int);
let
out1=(0→pre(out1))+1;tel;
```

图 10 累加器示例

Fig. 10 Accumulator example

图 10 中的 out1 在不同时间节点下的取值如表 2 所列。

表 2 out1 在不同时间节点下的取值

Table 2 Value of out1 at different time nodes

时间节点	out1
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10

流表达式常用来指示初始值。

4)布尔表达式

Lustre 支持用与(And)、或(Or)、非(Not)、异或(Xor)、蕴含(=>)逻辑运算符连接的布尔表达式。同时，Lustre 也支持整型和实型的关系运算符，即大于(>)、大于或等于(>=)、小于(<)、小于或等于(<=)。此外，它还支持所有类型的相等式(=)和不等式(<>)。

5)节点

节点是 Lustre 编程特定行为的核心机制。一个节点接收输入，并通过 let 后的执行主体为局部变量和输出变量赋值，如图 11 所示。

JKind 分析的切入点是主节点。如果有多个节点，JKind 将认为主节点是用--%MAIN 声明注释的节点，或者将使用文件中的最后一个节点作为主节点。主节点的输入变量可依据变量类型约束条件进行随机赋值。主节点中--%PREPERTY 用来指示需要验证的属性。如图 11 中，用--%MAIN 声明注释的主节点，其中--%PREPERTY p1 用来指示 JKind 验证 p1 始终为真。

```
node ccount(in:bool) returns(out:int);
var
p1:bool;
let
--%MAIN;
out=if in then((0→pre(out))+1) else 0;
p1=0<=outand out<=1;
--%PROPERTY p1;
tel;
```

图 11 节点

Fig. 11 Node

6)if-then-else 表达式

Lustre 支持使用 if-then-else 表达式，其可以用作变量赋值的条件表达式。

3 飞行模式转换需求模型的转换与验证

3.1 模型转换方法

由于 RSML^{-e}模型仅能够形式化地描述系统需求，无法实现系统的安全性验证，因此需要将其转换成能够使用模型检验工具进行安全性验证的语言，如 Lustre 同步数据流语言模型。转换算法如算法 1 所示，其接收 RSML^{-e}模型文件 R 作为输入，并输出 Lustre 同步数据流语言模型文件 L。假设 R 中数据类型集合为 DATATYPE，变量集合为 V，宏集合为 M。

算法 1 RSML^{-e}模型到 Lustre 同步数据流语言模型转换算法

输入：RSML^{-e}模型文件 R

输出：Lustre 同步数据流语言模型文件 L

```
1.  Initialize L=""; type_List=""; main_node_input="";
    main_node_output="true"; main_node_var=""; main_node_
    code=""
2.  L.append(type_List)
3.  L.append("node main(main_node_input) returns
    (main_node_output);\n var main_node_var\n let\n --%MAIN;
    \n
    main_node_code \n tel;\n")
4.  for each datatype∈DATATYPE do
5.    if datatype==bool or int or real
6.      type _List.append(type-trans(datatype))
7.    if datatype==enum{x1,x2,…,xn}
8.      type _List.append("type"+"datatype.title=enum{x1,
        x2,…,xn};"+"\n"+type-trans(datatype))
9.  for each v∈V do
10.   if v∈input variable
11.     main_node_input.append("v_in:type(v).title;\n")
12.     v_node,v_main←var-trans(v)
13.     L.append(v_node)
14.     main_node_var.append("v:type-trans(type(v)).title;\n")
15.     main_node_code.append(v_main)
16. for each m∈M do
```

```
17.     m_node,m_main←m-trans(m)
18.     L.append( m_node)
19.     main_node_var.append("m:bool_trans;\n")
20.     main_node_code.append(m_main)
21.   return L
22. end
```

算法 1 中，type_List 表示 L 中数据类型的列表；main_node_input，main_node_output，main_node_var，main_node_code 分别表示主节点的输入参数、输出参数、变量列表以及主节点执行主体。算法 1 第 2－3 行将数据类型列表 type_List 和主节点加入 L。第 4－8 行用 type-trans 将数据类型依次进行转换，并将原数据类型和转换后的数据类型加入 type_List。第 9－20 行用 m-trans 和 var-trans 将变量和宏转换为节点加入 L，并在主节点中实例化变量节点和宏节点。

3.1.1 类型的转换

由于 RSML^{-e}支持使用默认初始值 Undefined，而 Lustre 同步数据流语言不支持使用默认初始值 Undefined，因此需要对数据类型进行转换。具体转换如算法 2 所示，其接收数据类型 type_input 作为输入，并返回新的数据类型 type_output。

算法 2 type-trans(type_input)

```
1. Initialize type_output=""
2. if type_input==bool
3.     type_output.append(
       "type bool_trans=enum{True,False,Undefined_bool};\n")
4. if type_input==enum{x1,x2,…,xn}
5.     type_output.append("typetype_input.title_trans=enum{")
6.     for x in enum{x1,x2,…,xn} do
7.       type_output.append("x,")
8.       type_output.append("Undefined_type_input.title};\n")
9. if type_input==int
10.    type_output.append("type int_trans=
       struct{a:{Undefined_int,Defined_int};b:int};\n")
11. if type_input==real
12.    type_output.append("type real_trans=
       struct{a:{Undefined_real,Defined_real};b:real};\n")
13. return type_output
```

算法 2 中，type_input.title 指原数据类型的名称，如 enum{Disengaged，Engaged} 的名称为 ifengage；type-trans(type_input).title 表示转换后数据类型的名称，如 bool_trans，int_trans，ifengage_trans。

对于布尔型和枚举型，本文新建一个枚举型将 Undefined 作为一个枚举值并与 true 和 false 或原枚举值一同放入新建的枚举型，如算法 2 中第 2－8 行所示。对于整型或实型，本文新建一个包含两个元素的结构体类型，其中一个元素为枚举型，存储变量是否为默认初始值；另一个元素为整型或实型，存储原整型或实型，如算法 2 中第 9－12 行所示。

3.1.2 变量的转换

变量的转换如算法 3 所示，其接收变量 v 作为输入，返回节点 v_node 和在主节点中对节点的实例化语句 v_main。假设变量 v 依赖的其他变量或宏集合为 A={a_1，a_2，…，a_n}。

算法 3 var-trans(v)

```
1. Initialize v_node="";v_main=""
2. if v∈State Variable or v∈Controlled Variable
3.     v_node.append("nodev(")
4.     for a∈A do
5.       v_node.append("a:type-trans(type(a)).title;")
6.     v_node.append( ");\n let\n result=")
7.     if v has initial value v-initial
8.       v_node.append("v-initial->")
9.     if v has parent value v-parent.VALUE
10.      v_node.append(
         "if not(v-parent=VALUE) then Undefined_type(v) else")
11.    for each transition v.source→v.target if v.condution
12.      v_node.append(
         "if pre(result)=v.source and v.condution then v.target
         else")
13.      v_node.append("pre(result);\n tel;\n")
14.      v_main.append("v=v(")
15.    for a∈A do
16.      v_main.append("a,")
17.      v_main.append(");\n")
18.      if v∈Input Variable
19.        v_node.append("nodev(v_in:type(v).title)
           returns(result:type-trans(type(v)).title);\n let\n result
           =")
20.      if v has initial value v-initial
21.        v_node.append("v-initial->")
22.        v_node.append("v_in;\n tel;\n")
23.        v_main.append("v=v(v_in);\n")
24. return v_node,v_main
```

算法 3 中第 2－17 行为对普通变量(状态变量和控制变量)的转换，其取值依赖于集合 A 中其他变量或宏的值，所以第 4－6 行将 A 中的变量或宏加入其转换后节点的输入参数。第 9－10 行在节点输出参数 result 中用 if-then-else 语句对其父变量进行判断，只有父变量满足条件后才能进行后续对状态迁移的判断，否则将 Undefined 赋给 result。第 11－13 行用 if-then-else 语句对其状态迁移进行判断，并在满足状态迁移条件时对节点输出参数 result 赋予新值，在不满足状态迁移条件时保持 result 值不变。

算法 3 中第 18－23 行为对输入变量的转换，输入变量的取值不依赖其他变量或宏，可依据变量类型约束条件进行随机赋值，这也是主节点输入参数的性质，所以第 19 行将相应的主节点输入参数 v_in 加入其转换后节点的输入参数，并用 v_in 给其输出参数 result 赋值。

此外，算法 3 中第 7、第 8 行，第 20、第 21 行用 Lusre 中的流表达式为变量转换后节点的输出参数 result 赋予初始值，第 14－17 行、第 23 行为变量转换后的节点生成其在主节点中的实例化语句。

3.1.3 宏的转换

RSML^{-e}模型中的宏被用来封装复杂的 AND-OR 表或布尔值，所以宏的转换首先要对 AND-OR 表和 AND-OR 表中包含的逻辑短语进行转换。

1)逻辑短语的转换

RSML^{-e}中包含不同形式的逻辑短语,将它们转换为 Lustre 同步数据流语言中的布尔表达式,具体方法如算法 4 所示,其接收逻辑短语 logic_input 作为输入,并输出布尔表达式 logic_output。其中,运算符集合为 OP={>,<,≥,≤,=}。

算法 4 logic-trans(logic_input)

```
1. Initialize logic_output=""
2. if logic_input contains only true\false
3.     logic_output.append("true\false")
4. if logic_input contains only bool variable a
5.     logic_output.append("a=True")
6. if logic_input contains op∈OP and a before op and b after op
7.     if a contains PREVIOUS STEP(a')
8.         logic_output.append("pre(a') op b")
9.     if a contains WHEN(a')
10.        logic_output.append("a' op b")
11.        logic_output.append("a op b")
12. return logic_output
```

算法 4 中第 6—11 行包含了涉及时序表达式的比较和涉及状态变量的比较的逻辑短语的转换。其中,RSML^{-e}中 PREVIOUS STEP(a')代表前驱状态 a'的值,将它转换为 Lustre 同步数据流语言中代表 a'前驱状态的值 pre(a')。对于涉及状态变量的比较,When(a')将外层的 When 去掉,只留下 a'。

2)AND-OR 表的转换

AND-OR 表的转换如算法 5 所示,其接收 m 行 n 列(包括最左列逻辑短语)AND-OR 表 m * n-excel 作为输入,并输出布尔表达式 AO。假设 AND-OR 表第一列的逻辑短语集合为 LOGIC-PHRASE={l_1,l_2,…,l_m}。

算法 5 andor-trans(m * n-excel)

```
1. Initialize AO=""; expr1=""; expr2=""; …; exprn-1=""
2. for i=1->n-1 do
3.   for j=1->m do
4.     if m * n-excell[j][i]==T
5.       expri.append(logic-trans(lj)+"and")
6.     if m * n-excell[j][i]==F
7.       expri.append("not"+logic-trans(lj)+"and")
8.       expri.append("true")
9.       AO←expr1+"or"+expr2+"or"+…exprn
10. return AO
```

算法 5 中,第 2—8 行按列遍历 AND-OR 表,将每一列中每一行的 T 或 F 依据该行的逻辑短语用 and 连接成一个个子表达式;第 9 行将子表达式用 or 连接,形成最终的 AO 布尔表达式。

3)宏的转换

宏的转换如算法 6 所示,其接收宏 m 作为输入,返回节点 m_node 和在主节点中对节点的实例化语句 m_main。假设宏 m 中 AND-OR 表依赖的其他变量或宏的集合为 A={a_1,a_2,…,a_n}。

算法 6 m-trans(m)

```
1. Initialize m_node=""; m_main=""
2. if m is true\false
3.     m_node.append("nodem(a1:bool)returns(result:bool_trans);\
       n let\n result=True\False;\n tel;\n")
4.     m_main.append("m=m(true);\n")
5.  if m is AND-OR-Excel
6.     m_node.append("nodem(")
7.     for a∈A do
8.       m_node.append("a:type-trans(type(a)).title;")
9.       m_node.append(")returns(result:bool_trans);\n let\n
         result=ifandor-trans(AND-OR-Excel) then True else False;
         \n tel;\n")
10.      m_main.append("v=v(")
11.    for a∈A do
12.      m_main.append("a,")
13.      m_main.append(");\n")
14. return m_node,m_main
```

算法 6 中,第 2—4 行对用来代替布尔值的宏进行转换,将一个恒为真的布尔变量 a_1 作为转换后节点的输入参数,并直接将 True\False 赋予节点的输出参数 result。第 5—13 行对用来代替 AND-OR 表的宏进行转换,将 AND-OR 表依赖的其他变量或宏作为转换后节点的输入参数,并将 AND-OR 表通过 andor-trans 进行转换后得到的布尔表达式赋予输出参数 result,当该布尔表达式为真时,result=True,反之,result=False。第 4 行、第 10—13 行为宏转换后的节点生成其在主节点中的实例化语句。

3.1.4 算法复杂度分析

时间复杂度分析:由于从 RSML^{-e}到 Lustre 模型的转换由主算法(算法 1)和子算法(算法 2—算法 6)共同工作完成,因此需要同时分析主算法和子算法的时间复杂度。由于主算法 1 调用了算法 2、算法 3 和算法 6,算法 6 调用了算法 5,算法 5 又调用了算法 4,因此首先对最底层的算法 4 进行分析。

算法 4 是模型中逻辑短语的转换,算法操作不涉及循环操作,因此,算法 4 的时间复杂度为 O(1)。

算法 5 是对 RSML^{-e}中 AND-OR 表的转换,由于 RSML^{-e}模型 R 中的 AND-OR 表(事实上是一个 $m*n$ 的矩阵)大小不一,因此算法分析中取 AND-OR 表大小的上界,即每个 AND-OR 表都为 $m*n$ 矩阵,而算法 5 对此矩阵进行遍历,在遍历过程中调用时间复杂度为 O(1)的算法 4 对逻辑短语进行转换,因此算法 5 的时间复杂度 O(mn)。

算法 6 是对模型宏的转换,其转换过程调用时间复杂度为 O(mn)的算法 5,并两次对宏所依赖的其他变量或宏进行遍历,在算法分析中取宏所依赖的其他变量或宏数量的上界,即 RSML^{-e}模型 R 中所有变量和宏的总数为 $v+h$,因此算法 6 的时间复杂度为 O($2(v+h)+mn$)。

算法 3 是模型变量的转换,其中第 4、第 5 行和第 15、第 16 行对变量所依赖的其他变量或宏进行遍历,在算法分析中取变量所依赖的其他变量或宏数量的上界,即 RSML^{-e}模型 R 中所有变量和宏的总数为 $r+b$,第 11、第 12 行对变量中的状态迁移进行遍历,在算法分析中取变量中状态迁移数量的上界 r,因此算法 3 的复杂度为 O($2(v+h)+r$)。

算法 2 是模型数据类型的转换,算法操作不涉及循环操作,因此算法 2 的时间复杂度为 O(1)。

接下来分析主算法的时间复杂度。RSML^{-e}模型 R 中变量的数据类型为 t 种，变量的个数为 v，宏的个数为 h，变量中状态迁移个数的上限为 r，AND-OR 表大小的上界为 $m*n$。主算法中第 4－8 行对数据类型进行遍历并调用算法 2 对数据类型进行转换，第 9－15 行对变量进行遍历并调用算法 3 对变量进行转换，第 16－20 行对宏进行遍历并调用算法 6 对宏进行转换，所以主算法的时间复杂度为 $O(t+v*(2(v+h)+r)+h*(2(v+h)+mn))=O(2v^2+2h^2+4vh+2vr+2hmn+t)=O(2(v+h)^2+2vr+2hmn+t)$，由于 $2(v+h)^2$ 远远大于 $2vr+2hmn+t$，因此主算法的时间复杂度为 $O((v+h)^2)$。

综上所述，从 RSML^{-e}到 Lustre 模型转换的时间复杂度为 $O((v+h)^2)$。

空间复杂度分析：由于算法对 RSML^{-e}模型中的各个模块依次进行转换，并且各个模块在转换过程中只涉及模块内的元素，因此在对一个模块的转换完成后可以删除该模块转换前的数据，未额外占用存储空间，所以转换算法的空间复杂度为 O(1)。

3.2 安全性质的描述

在对模型进行转换后，需要将安全性质描述为 Lustre 同步数据流语言所支持的形式。下面通过一个例子来说明安全性质的描述。FGS 需要满足这样一条性质：如果此侧处于活动状态且模式信号关闭，则当对侧 FD 打开时，模式信号应打开，将它描述为 Lustre 同步数据流语言所支持的形式如下：prop1＝((not pre(Modes)＝On) and (not pre(Onside_FD)＝On) and Is_This_Side_Active＝True and Onside_FD＝On)＝＞Modes＝On。

飞行模式转换需求模型中存在一类变量，它的取值由且只由唯一其他变量所决定，对它的取值的判断等同于判断决定其取值的变量的取值，但在模型检验过程中却要分配相应的内存用于存储它的值，如果这类变量应用过多，势必会额外占用大量内存，加重状态空间爆炸问题，影响模型检验的效率。因此为了节省内存，缓解状态空间爆炸问题，本文在描述安全性质时避免使用 Mode_Annunciations_On 这类其值由且只由唯一其他变量所决定的变量，转而用决定其取值的变量的值代替，如 Mode_Annunciations_On＝True 用 Modes＝On 来代替，Mode_Annunciations_On＝False 用(not Modes＝On)来代替。

3.3 模型验证

在完成对模型的转换和安全性质的描述后，将所得到的 Lustre 模型与 Lustre 安全性质输入 Jkind 模型检验工具进行验证。如图 1 所示，如模型满足安全性质，则输出 true；如模型不满足安全性质，则输出反例，进而根据反例修改模型，直至模型满足安全性质。

4 实例验证

本文选用文献[9]中的 FGS 模型来进行实例的分析验证。实验环境为：CPU 为 lntel$^{(R)}$ Core$^{(TM)}$ i7-8750H CPU @ 2.20GHz 2.21GHz，GPU 为 GeForce GTX 950M 4GB，内存大小为 8GB，操作系统为 Windows 11。

FGS 接收来自姿态航向参考系统(AHRS)、空中数据系统(ADS)、飞行管理系统(FMS)、导航无线电和飞行控制面板(FCP)的输入。利用这些信息，它通过模式逻辑(Modes Logic)选择合适的飞行模式，并通过飞行控制律(Control Laws)计算飞行导引操纵指令提供给自动驾驶仪(AP)，之后由自动驾驶仪将这些命令转换为飞机控制面的运动，以最小化测量状态和期望状态之间的差异。FGS 的主要功能如表 3 所列。

表 3 典型 FGS 功能

Table 3 Typical FGS functions

FGS 功能
计算飞行导引操纵指令
选择并指示飞行模式
控制飞行指引仪(FD)
控制自动驾驶仪(AP)

其中选择并指示飞行模式是 FGS 最为重要的功能。飞行模式分为横向模式和纵向模式，横向模式和纵向模式又分为多个子模式，如表 4 所列。

表 4 飞行模式

Table 4 Flight modes

横向模式	纵向模式
滚转保持(ROLL)	俯仰保持(PITCH)
进近(APPR)	高度保持(ALT)
复飞(GA)	高度选择(ALTSEL)
航向选择(HDG)	进近(APPR)
导航(NAV)	飞行高度层改变(FLC)
	复飞(GA)
	垂直速度(VS)

飞行模式的转换需要满足相应的安全性质，截取部分安全性质进行展示，如表 5 所列。

表 5 部分安全性质

Table 5 Partial safety properties

编号	安全性质
1	HDG 处于 clear 状态，若此时按下 HDG 开关，那么 HDG 将处于接通状态
2	HDG 处于接通状态，若此时按下 HDG 开关(即关闭 HDG)，那么 HDG 将处于断开状态
3	NAV 处于关闭状态，若此时按下按钮，并按钮灯亮，那么 NAV 模式将被激活
4	NAV 处于被选择状态，若按下按钮，则 NAV 关闭
5	VS 处于关闭状态，若此时按下按钮，并按钮灯亮，那么 VS 模式将被激活
6	VS 处于激活状态，若按下按钮，则 VS 关闭
7	ALT 处于关闭状态，若此时按下按钮，并按钮灯亮，那么 ALT 模式将被激活
8	ALT 处于激活状态，若按下按钮，则 ALT 关闭
9	ALT 处于关闭状态，若此时按下按钮，并按钮灯亮，那么 ALT 模式将被激活，ALTSEL 模式处于待命状态
10	ALTSEL 处于激活状态，若捕获高度改变，则 ALTSEL 关闭

下面对 FGS 飞行模式转换的需求模型进行转换和验证。

4.1 模型转换

以输入变量 NAV_Switch，状态变量 NAV_Selected 和宏 When_NAV_Activated 为例进行说明。输入变量 NAV_Switch 在转换前的 RSML^{-e}形式如图 13 所示。转换后的节点如图 14 所示。

NAV_Switch
Type: {OFF, ON}
Initial Value: Undefined
Classified as: MONITORED

图 13 NAV_Switch RSML^{-e}模型

Fig. 13 RSML^{-e} model of NAV_Switch

```
typeswitch_lamp=enum{OFF,ON};
typeswitch_lamp_trans=enum{oFF,oN,Undefined_switch_lamp};

node
NAV_Switch(NAV_Switch_in:switch_lamp_trans) returns(result:switch_
lamp_trans);
let
result =Undefined_switch_lamp -> ifNAV_Switch_in=ON then oN else
oFF;
tel;
```

图 14 NAV_Switch Lustre 模型

Fig. 14 Lustre model of NAV_Switch

状态变量 NAV_Selected 在转换前的 RSML^{-e} 形式如图 15 所示。转换后的节点如图 16 所示。

NAV_Selected			
Parent: Modes. On ▷ NAV. Selected			
Type: {Armed, Active}			
Initial Value: Undefined			
Classified as: State			
:= Offside_NAV_Selected if not Is_This_Side_Active			
Undefined→Armedif TRUE			
Armed→Activeif			
			OR
	AND	Activate_NAV	T
		Is_This_Side_Active	T

图 15 NAV_Selected RSML^{-e}模型

Fig. 15 RSML^{-e} model of NAV_Selected

```
typebool_trans=enum{True,False,Undefined_bool};
typeselectedstate=enum{Armed,Active};
typeselectedstate_trans=enum{armed,active,Undefined_selectedstate};
type basestate=enum{Cleared,Selected};
typebasestate_trans= enum{cleared,selected,Undefined_basestate};

node
NAV_Selected(Is_This_Side_Active,Activate_NAV:bool_trans;NAV:bas-
estate_trans;Offside_NAV_Selected:selectedstate_trans) returns(result:se-
lectedstate_trans);
let
result=Undefined_selectedstate -> if not(NAV=selected) then Undefined_
selectedstate else if Is_This_Side_Active=False then Offside_NAV_Selected
else if pre(result)=Undefined_selectedstate then armed else if pre(result)=
armed and Activate_NAV=True and Is_This_Side_Active=True then active
else pre(result);
tel;
```

图 16 NAV_Selected Lustre 模型

Fig. 16 Lustre model of NAV_Selected

宏 When_NAV_Activated 在转换前的 RSML^{-e} 形式如图 17 所示。转换后的节点如图 18 所示。

When_NAV_Activated			
Condition:			
			OR
		Activate_NAV	T
	AND	PREVIOUS STEP(NAV_Selected)=Armed	T
		Is_This_Side_Active	T

图 17 When_NAV_Activated RSML^{-e}模型

Fig. 17 RSML^{-e} model of When_NAV_Activated

```
typebool_trans=enum{True,False,Undefined_bool};
typeselectedstate=enum{Armed,Active};
typeselectedstate_trans=enum{armed,active,Undefined_selectedstate};

node
When_NAV_Activated(Activate_NAV,Is_This_Side_Active:bool_trans;
NAV_Selected:selectedstate_trans) returns(result:bool_trans);
let
result =ifActivate_NAV= True and pre(NAV_Selected)=armed and Is_
This_Side_Active=True then True else False;
tel;
```

图 18 When_NAV_Activated Lustre 模型

Fig. 18 Lustre model of When_NAV_Activated

假设要验证的性质 NAV_Selected 始终为 Armed，主节点如图 19 所示。

```
typebool_trans=enum{True,False,Undefined_bool};
typeswitch_lamp=enum{OFF,ON};
typeswitch_lamp_trans=enum{oFF,oN,Undefined_switch_lamp};
typeselectedstate=enum{Armed,Active};
typeselectedstate_trans=enum{armed,active,Undefined_selectedstate};

node main(NAV_Switch_in:switch_lamp) returns(prop1:bool);
var
NAV_Switch:switch_lamp_trans;
When_NAV_Activated:boo_trans;
NAV_Selected:selectedstate_trans;
let
--%MAIN;
NAV_Switch=NAV_Switch(NAV_Switch_in);
When_NAV_Activated=When_NAV_Activated(Activate_NAV,Is_This_
Side_Active,NAV_Selected);
NAV_Selected=NAV_Selected(Is_This_Side_Active,Activate_NAV,NAV,
Offside_NAV_Selected);

prop1=NAV_Selected=armed;
--%PROPERTY prop1;
tel;
```

图 19 主节点模型

Fig. 19 Lustre model of main node

4.2 模型验证

利用前文所述方法得到 FGS 飞行模式转换需求模型的 Lustre 同步数据流语言模型及相应的安全性质，运行 JKind 工具，对安全性质进行验证。图 20 所示为安全性质。如果此侧处于活动状态且模式信号关闭，则当对侧 FD 打开时，模式信号应打开(prop1=((not pre(Modes)=On) and (not pre(Onside_FD)=On) and Is_This_Side_Active=True and Onside_FD=On)=>Modes=On)的验证结果。

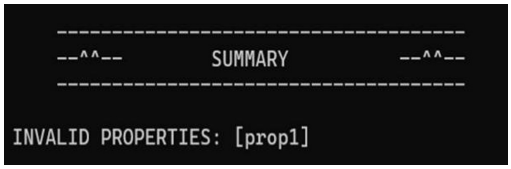

图 20 安全性质的验证结果

Fig. 20 Verification results of safety properties

该验证结果表明此性质不通过，通过对 Jkind 输出的反例(图 21 和图 22 截取了反例的部分内容)进行分析，发现是 Onside_FD 的初始状态未确定导致的，于是对 Onside_FD 赋予初始值 Off 后得到如图 23 所示的结果。

图 21 反例中的输入变量

Fig. 21 Input variable in the counterexample

图 22 反例中的普通变量

Fig. 22 Ordinary variables in counterexamples

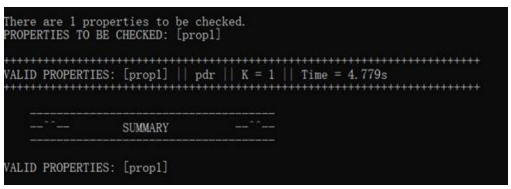

图 23 改正后的验证结果

Fig. 23 Corrected verification result

从结果可知，该安全性质在实例中是满足的，并且对于该性质的验证只用了 4.779 s，在时间上处于一个可接受的范围。

对于上文例举安全性质时所提到的 10 条涉及不同飞行模式的安全性质，本文用 NUSMV 和 Jkind 分别对其进行了验证，结果如图 24、图 25 及表 6、表 7 所示。

图 24 NUSMV 的验证结果

Fig. 24 Verification results of NUSMV

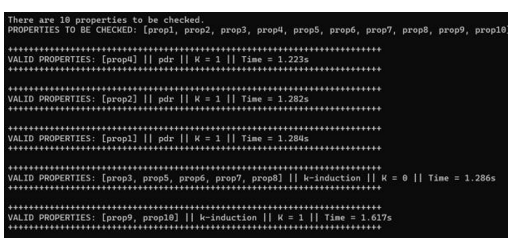

图 25 Jkind 的部分验证结果

Fig. 25 Partial verification results of Jkind

结果表明，10 个安全性质均通过验证，Jkind 花费的时间和占用的内存都要少于 NUSMV。

接着用 NUSMV 和 Jkind 分别对 FGS 飞行模式转换需求模型中涉及不同飞行模式的 50 个安全性质进行验证，结果如图 26、图 27 及表 8 所示。

表 6 安全性质是否满足

Table 6 Whether the safety properties are met

编号	安全性质	是否满足
1	HDG 处于 clear 状态，若此时按下 HDG 开关，那么 HDG 将处于接通状态	是
2	HDG 处于接通状态，若此时按下 HDG 开关(即关闭 HDG)，那么 HDG 将处于断开状态	是
3	NAV 处于关闭状态，若此时按下按钮，并按钮灯亮，那么 NAV 模式将被激活	是
4	NAV 处于被选择状态，若按下按钮，则 NAV 关闭	是
5	VS 处于关闭状态，若此时按下按钮，并按钮灯亮，那么 VS 模式将被激活	是
6	VS 处于激活状态，若按下按钮，则 VS 关闭	是
7	ALT 处于关闭状态，若此时按下按钮，并按钮灯亮，那么 ALT 模式将被激活	是
8	ALT 处于激活状态，若按下按钮，则 ALT 关闭	是
9	ALT 处于关闭状态，若此时按下按钮，并按钮灯亮，那么 ALT 模式将被激活，ALTSEL 模式处于待命状态	是
10	ALTSEL 处于激活状态，若捕获高度改变，则 ALTSEL 关闭	是

表 7 验证花费

Table 7 Cost of verification

	通过性质/个	花费时间/s	耗费内存/kB
NUSMV	10	1 219.72	1 615 980
JKind	10	6.692	242 288

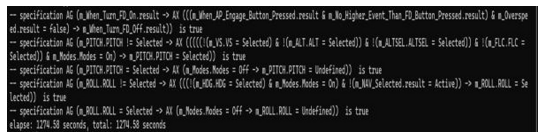

图 26 NUSMV 的部分验证结果

Fig. 26 Partial verification results of NUSMV

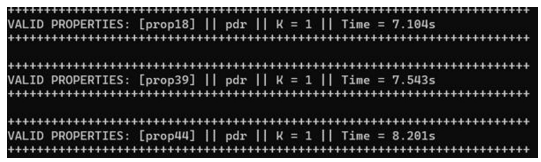

图 27 Jkind 的部分验证结果

Fig. 27 Partial verification results of Jkind

表 8 50 个安全性质验证结果

Table 8 Result of verification of 50 safety properties

	通过性质/个	花费时间/s	耗费内存/kB
NUSMV	50	1 274.58	1 623 596
JKind	50	80.69	943 696

结果表明，50 个安全性质均通过验证，对于 FGS 飞行模式转换需求模型的 50 个相同安全性质，Jkind 花费的时间和占用的内存都要少于 NUSMV；并且通过与上文对 10 个安全性质的验证结果对比可知，对于同一个飞行模式转换需求模型，所要验证的性质的数量对 JKind 验证时间和耗费内存有所影响，但对 NUSMV 的影响不大。由此可见，对于复杂系统来说，JKind 可以选择将性质分开进行验证，甚至一次只对

单条性质进行验证，以进一步缓解状态空间爆炸问题。综上，利用 Lustre 同步数据流语言和 JKind 工具对其进行验证可以缓解状态空间爆炸问题，大大加快系统开发进程。

结束语 为了提高软件系统的安全性，基于模型的安全性分析和验证已经成为软件开发周期的重要组成部分。本文首先对 RSML^{-e} 与 Lustre 同步数据流语言的特征进行分析，并提出了一种将 RSML^{-e} 模型转换为可进行属性验证的 Lustre 同步数据流语言模型的方法；然后将相关安全性质描述为 Lustre 同步数据流语言所支持的形式；最后对其进行验证。

本文在对模型的转换中将 RSML^{-e} 数据类型的布尔型、枚举型转换为 Lustre 同步数据流语言枚举型；将 RSML^{-e} 数据类型的整型、实型转换为 Lustre 同步数据流语言结构体类型；将 RSML^{-e} 中变量、宏转换为 Lustre 同步数据流语言节点；将 RSML^{-e} 中逻辑短语、AND-OR 表转换为 Lustre 同步数据流语言布尔表达式。

在对安全性质进行描述时避免使用文献[9]中 Mode_Annunciations_On 这类其值由且只由唯一其他变量所决定的变量，转而用决定其取值的变量的值代替，一定程度上减少了验证所需变量的数量，减轻了模型的复杂度。

对于实际的飞行模式转换需求模型，本文进行了模型转换和模型验证，并与 NUSMV 进行了对比。从结果来看，本文方法对实际系统的验证是有效的，并且在一定程度上缓解了状态空间爆炸的问题。

虽然本文方法在一定程度上缓解了状态空间爆炸的问题，但状态空间爆炸问题并未完全解决，未来还需在此基础上增加对其他缓解状态爆炸问题方法的研究与应用。

参 考 文 献

[1] BOWEN J, STAVRIDOU V. Safety-critical systems, formal methods and standards[J]. IEE Software Engineering Journal, 1993, 8(4): 189-209.

[2] PREDUT S N, IPATE F, GHEORGHE M, et al. Formal modelling of cruise control system using Event-B and Rodin platform [C]// 2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS). IEEE, 2018: 1541-1546.

[3] BONFANTI S, GARGANTINI A, MASHKOOR A. Design and validation of a C++ code generator from abstract state machines specifications[J]. Journal of Software: Evolution and Process, 2020, 32(2): e2205.

[4] SYRIANI E, SOUSA V, LÚCIO L. Structure and behavior preserving statecharts refinements[J]. Science of Computer Programming, 2019, 170: 45-79.

[5] HOARE CAR. Communicating sequential processes[M]// Theories of Programming: The Life and Works of Tony Hoare. 2021: 157-186.

[6] GORRIERI R, VERSARI C. CCS: a calculus of communicating systems[M]// Introduction to Concurrency Theory: Transition Systems and CCS. 2015: 81-161.

[7] STEWART D, LIU J J, COFER D, et al. AADL-Based safety analysis using formal methods applied to aircraft digital systems [J]. Reliability Engineering & System Safety, 2021, 213: 107649.

[8] HEIMDAHL M P E, LEVESON N G, REESE J D. Experiences from specifying the TCAS II requirements using RSML[C]// 17th DASC. AIAA/IEEE/SAE. Digital Avionics Systems Conference. Proceedings (Cat. No. 98CH36267). IEEE, 1998: C43/1-C43/8.

[9] MILLER S, TRIBBLE A, CARLSON T, et al. Flight guidance system requirements specification: NAS/CR-2003-212426[R]. Washington DC: National Aeronautics and Space Adminis-tration, Langley Research Center, 2003.

[10] BU L, WANG Q, REN X, et al. Scenario-based online reachability validation for CPS fault prediction[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2019, 39(10): 2081-2094.

[11] LI Z, LIU B, LU M, et al. Modeling and verification of de-icing software safety requirement based on expanded Petri net[J]. Journal of Beijing University of Aeronautics and Astronautics, 2012, 38(1): 64-69.

[12] CHEN Z, GU Y, HUANG Z, et al. Model checking aircraft controller software: A case study[J]. Software: Practice and Experience, 2015, 45(7): 989-1017.

[13] HU J, ZHANG W J, LI W Q. A requirement oriented formal modeling and verification method for safety critical systems[J]. Computer Engineering & Science, 2019, 41(8): 1426-1433.

[14] CHEN S, HU J, TANG H Y, et al. Transformation method for AltaRica3.0 model to NuSMV model[J]. Computer Science, 2020, 47(12): 73-86.

[15] LIU C, JIANG Y P, MA C Y, et al. Formal verification of AADL models by NuSMV[J]. Acta Aeronautica et Astronautica Sinica, 2022, 43(1): 451-466.

[16] ZHANG Y, HU J, WANG L S, et al. Formal verification method for SCR requirement model[J]. Journal of Chinese Computer Systems, 2022, 43(1): 193-202.

[17] LI J A, HU J, WANG L S, et al. Formal Modeling and Verification of Automatic Flight Mode Transition Logic[J]. Journal of Nanjing University of Aeronautics & Astronautics/Nanjing Hangkong Hangtian Daxue Xuebao, 2023, 55(5): 768-779.

[18] RAN D, CHEN Z, SUN Y, et al. SCADE Model Checking Based on Program Transformation [J]. Computer Science, 2021, 48(12): 125-130.

[19] TRIBBLE A C, LEMPIA D L, MILLER S P. Software safety analysis of a flight guidance system[C]// Proceedings. The 21st Digital Avionics Systems Conference. IEEE, 2002: 13C1-13C1.

[20] LUTZ R R. Analyzing software requirements errors in safety-critical, embedded systems[C]// Proceedings of the IEEE International Symposium on Requirements Engineering. IEEE, 1993: 126-133.

[21] GACEK A, BACKES J, WHALEN M, et al. TheJKind model checker[C]// Computer Aided Verification: 30th International

Conference. Springer，2018：20-27.

[22] HALBWACHS N，CASPI P，RAYMOND P，et al. The synchronous data flow programming language LUSTRE[C]// Proceedings of the IEEE. 1991：1305-1320.

[23] ANDREWS S，SOTOUDEH M，BARRETT C. Efficient sat-based bounded model checking of evolving systems[C]// 2025 Design，Automation & Test in Europe Conference（DATE）. IEEE，2025：1-7.

[24] CHATTERJEE P，ROY S，DIEP B P，et al. Distributed bounded model checking[J]. Formal Methods in System Design，2024，64(1)：50-72.

[25] HSU T H，SÁNCHEZ C，BONAKDARPOUR B. Bounded model checking for hyperproperties[C]// International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Cham：Springer，2021：94-112.

[26] KAHSAI T，GAROCHE P L，TINELLI C，et al. Incremental verification with mode variable invariants in state machines [C]// NASA Formal Methods：4th International Symposium. Berlin：Springer，2012：388-402.

[27] EÉN N，MISHCHENKO A，BRAYTON R. Efficient implementation of property directed reachability[C]// 2011 Formal Methods in Computer-Aided Design（FMCAD）. IEEE，2011：125-134.

[28] CIMATTI A，GRIGGIO A，MOVER S，et al. IC3 modulo theories via implicit predicate abstraction[C]// Tools and Algorithms for the Construction and Analysis of Systems：20th International Conference. Berlin：Springer，2014：46-61.

[29] DUREJA R，ROZIER K Y. Incremental design-space model checking via reusable reachable state approximations[J]. Formal Methods in System Design，2021，58(3)：375-398.

[30] GaimminLtd. Garmin G1000 pilot's guide for CessnaNav III [M]. USA：GaiminLtd，2005：429-477.

[31] KOLANO E P. Pilot's operating handbook and FAA approved airplane flight manual[M]. USA：FAA，2011

[32] WANG J，ZHAN N J，FENG X Y，et al. Overview of Formal Methods[J]. Journal of Software，2019，30(1)：33-61.

WANG Zhiyi，born in 2000，postgraduate. His main research interests include software analysis，model checking and formal verification.

HU Jun，born in 1973，Ph.D，associate professor，is a senior member of CCF. His main research interests include intelligent computing and analysis for advanced systems，model-based systems engineering，and system safety modeling and analysis.

（责任编辑：柯颖）