面向模型检测的 LTL 语句自动生成方法

段喜龙[1,2]，陆智伟[1,2+]，郑 巍[1,2]，陈晋升[1,2]，樊 鑫[1,2]，肖 鹏[1,2]

（1. 南昌航空大学 软件学院，江西 南昌 330063；2. 南昌航空大学 软件测评中心，江西 南昌 330063）

摘 要：为优化线性时态逻辑语句的生成过程，减少模型检测的时间，提出一种面向模型检测的基于自然语言处理生成线性时态逻辑验证语句的方法。对需求文档提取关键词，将文档中的数据和可以代表模型中状态的名词进行提取，注释 UML 模型，对 UML 模型中的状态进行归类，将模型中的状态分为数据属性类和调用操作类，利用配对的线性时态逻辑格式生成线性时态逻辑，用于软件模型一致性验证。实验结果表明，该方法与 ST 模型相比可以提高模型检测的效率。

关键词：自然语言处理；模型一致性；线性时态逻辑；UML 模型；形式化验证工具；模型验证；模型注释

中图法分类号：TP311 **文献标识号**：A **文章编号**：1000-7024（2023）08-2337-08

doi：10.16208/j.issn1000-7024.2023.08.013

Automatic generation of LTL statements for model checking

DUAN Xi-long[1,2], LU Zhi-wei[1,2+], ZHENG Wei[1,2], CHEN Jin-sheng[1,2], FAN Xin[1,2], XIAO Peng[1,2]

(1. School of Software, Nanchang Hangkong University, Nanchang 330063, China;
2. Software Testing and Evaluation Center, Nanchang Hangkong University, Nanchang 330063, China)

Abstract: To optimize the generation process of linear temporal logic statements, reduce the time of model detection, a model checking oriented method for generating linear temporal logic verification statements based on natural language processing was presented. Keywords were extracted from the requirements document, the data in the document and nouns that represented the state in the model were extracted, the UML model was annotated, the states in the UML model were classified, the states in the model were divided into data attribute classes and call operation classes, and the paired linear temporal logic format was used to generate linear temporal logic for software model consistency verification. Experimental results show that the proposed method can improve the efficiency of model detection compared with ST model.

Key words: natural language processing; model consistency; linear temporal logic; UML model; formal verification tools; model validation; model annotation

0 引 言

模型检测作为软件系统验证中的一个常用手段，已经在多个领域得到了应用。在基于模型开发的软件系统验证中，线性时态逻辑 LTL（linear-time temporal logic）用于描述软件的性质，这些性质又被称为线性时间属性。这是迈向构建模型检测理论重要的一步，现已得到了广泛的应用[1-4]。

LTL 能得到广泛应用有赖于人们对它的表达能力的研究。文献［5］中给出，LTL 的表达能力与一阶谓词逻辑等价。尽管它们的表达能力是一样的，但两者在可满足性问题，可判定问题还有推理问题上的解决难易程度却大不相同。这些问题在一阶谓词逻辑上的解决难度是非初等的，也就是说它的解决方案的复杂度上界是可以无限增长的[6,7]。而在 LTL 上，这些问题的解决难度都只是 PSPACE-完全的[8,9]。这也是能够得到广泛应用的一个重要的理论支撑。而 LTL 语句生成需要测试人员对被测模型要有充分的了解，而这一过程需要花费很多时间。

收稿日期：2022-09-13；**修订日期**：2023-08-01

基金项目：国家自然科学基金项目（61867004）

作者简介：段喜龙（1978-），男，黑龙江哈尔滨人，硕士，讲师，CCF 专业会员，研究方向为软件可靠性；+通讯作者：陆智伟（1998-），男，江西上饶人，硕士研究生，CCF 学生会员，研究方向为软件可靠性；郑巍（1982-），男，江西萍乡人，博士，教授，硕士生导师，CCF 高级会员，研究方向为软件可靠性；陈晋升（1998-），男，山西运城人，硕士研究生，CCF 学生会员，研究方向为软件测试；樊鑫（1981-），男，湖北荆州人，硕士，副教授，研究方向为软件测试；肖鹏（1988-），男，江西吉安人，博士，讲师，研究方向为软件测试。E-mail：luzhiwei_nchu@126.com

在模型检测方面，文献［12］从时间逻辑角度对系统行为进行了刻画，文献［13］采用时间抽象互模拟方法来验证模型；文献［14］分别采用动态层次化 UML 状态机模型和符号模型进行验证；文献［15］将时间自动机（timed automata，TA）模型转换为有限状态迁移图，并将有限状态迁移图转换为非确定有限状态机（finite state machine，FSM），从而采用基于 FSM 的方法进行测试；文献［16］是一种通过在运行时验证软件源代码中的断言来检测不一致的方法。但是这些工作都不是完全自动化的，验证人员需要手动的生成 LTL 语句，验证效率低。因此，研究和实现 LTL 自动生成的方法是必要的。

本文提出了一个基于自然语言处理的线性时态逻辑自动生成的方法，支持基于模型的开发软件设计的分析。在本文的方法中，通过自动生成线性时态逻辑声明来建立来对模型与需求的进行一致性分析。本文研究的目标是减少在软件开发后期检测需求模型一致性所需的工作。

1 本文工作

本文提出的 LTL 语句自动生成方法如图 1 所示。本文采用的输入是需求说明书和 UML 模型，然后采用关键词提取，基于注释 UML 模型的 LTL 生成方法，从而生成 LTL 语句。

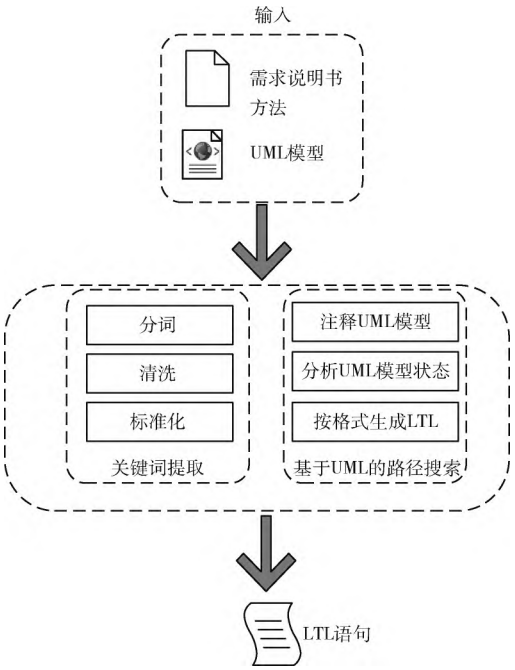

图 1 LTL 语句自动生成方法

1.1 关键词提取

关键词提取是本文根据需求文档生成 LTL 验证语句的关键步骤。关键词提取流程如图 2 所示，首先对需求文档进行分词，采用语言技术平台（language technology platform，LTP）[17]，LTP 提供了一系列中文自然语言处理工具，用户可以使用这些工具对于中文文本进行分词、词性标注、句法分析等等工作。利用 LTP 进行分词，接着对词语进行清洗，清洗过程包括单词翻译、分析停用词；最后通过标准化（词干提取和词形还原）得到关键词集合。

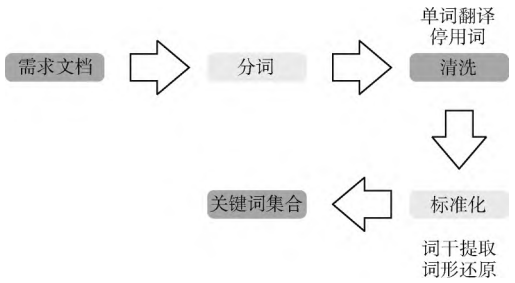

图 2 关键词提取流程

本文采用的关键词提取算法为 TextRank 算法[18]。

TextRank 模型表示为一个有向有权 $G=(V,E)$ 由点集合 V 和边集合 E 组成，E 是 $V\times V$ 的子集

$$WS(V_i)=(1-d)+d*\sum_{V_i}^{n}\frac{w_{ij}}{\sum V_k w_{jk}}WS \tag{1}$$

式（1）表示的是 TextRank 中一个单词长度 i 的权重取决于与在长度 i 前面的各个点长度 j 组成的长度 (j,i) 这条边的权重，以及长度 j 这个点到其它边的权重之和。

$WS(V_i)$ 表示的是句子的权重，右侧的求和表示每个相邻句子对本句子的贡献程度，在单文档中本文可以大致认为所有的句子都是相邻的，不需要像多文档一样进行多个窗口的生成和抽取。W_{ij} 表示两个句子的相似度，WS 表示上次迭代出句子的权重，d 为阻尼系数，一般为 0.85。

例如，文本中有句子“通过贷款人数据信息，并进行风险分析”，“风险”和“分析”均属于候选关键词，则组合成“风险分析”加入关键词序列。最后得到 DataSet$\{d_1,d_2,\cdots,d_i\}$ 集合。DataSet 集合是所有数据属性和调用操作的集合，是通过 TextRank 算法从需求文档中得到的。

1.2 基于注释 UML 模型的 LTL 生成算法

基于注释 UML 模型的 LTL 生成算法流程如图 3 所示。

LTL 生成算法采用 UML 模型 m 和关键词集合作为输入，输出一组 LTL 语句。首先对 UML 进行注释，通过分析注释后的 UML 模型得到 GuardSet 集合 $\{g_1,g_2,\cdots,g_i\}$（GuardSet 集合是关于 DataSet 集合的一组使用条件）。接着对 GuardSet 集合进行遍历得到状态 d，判断 d 是否是调用操作，若是调用操作则生成一条相应的调用操作 LTL 语句。若不是，则判断 d 是否为数据属性，若是则生成一个相应的数据属性 LTL 语句。否则将 d 归类为非法数据，开始判断下一个状态 d。

1.2.1 注释 UML 模型

在处理需求文档时，将 UML 模型用数据信息进行注释，产生的文件称为 UML 模型的 UML 注释文件（UML-

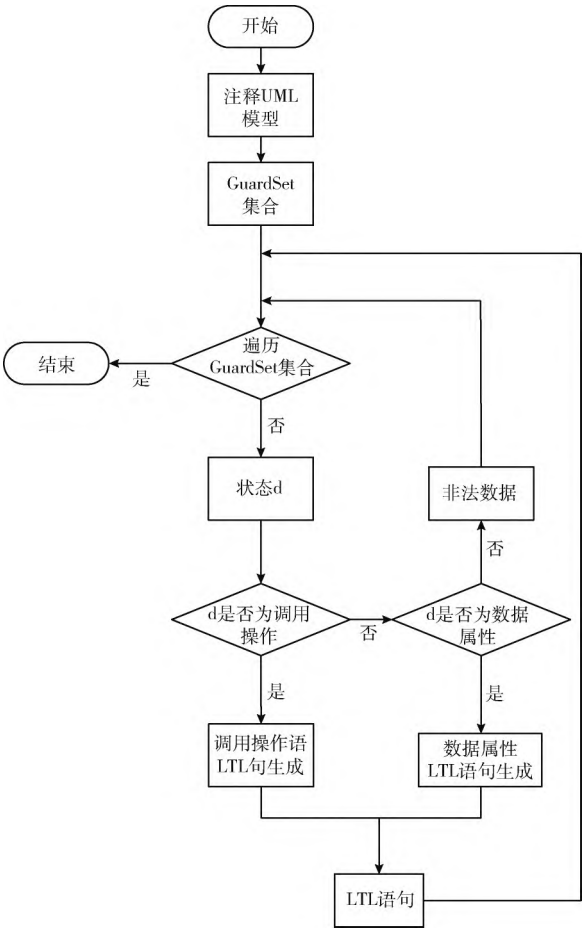

图 3　基于注释 UML 模型的 LTL 生成算法流程

notes)。UMLnotes 内容是 UML 的扩展并允许相关的数据在 UML 模型中指定数据信息。还引入了 UML 安全概要 UMLsec 的子概要。UMLnotes 的一些信息在注释 UML 模型简介见表 1。

表 1　注释 UML 模型简介

类型	标签
《critical》	{protectedData = {(string[1 … *])}} //必要的

《interpretation domain》	{senDecisions={(string[1… *])}} //必要的
	{expData = {((string[1], string[1 … *]) *)}}//可选{metric={string[0]}} //可选
	{threshold={double[0]}} //可选

在生成 LTL 语句过程中，在 UMLnotes 配置文件中有一个称为“《critical》”的类型，这是从 UMLsec 配置文件中重用的一个类型。在 UMLsec 中此构造型注释可能包含数据的类。在本文的工作中，使用了这个扩展类型来注释可能包含受需要验证的字符类。如表 1 所示，扩展了这个带有 {protectedData} 标签的构造型允许定义受保护的字符与注释为《critical》的类有关的特征。

《interpretation domain》注释了一个模型状态机，它会去记录类的会发生的行为，将其注释为一些标签注释 UML 所包含的标签如图 4 所示。本文的模型包含以下几个标签。

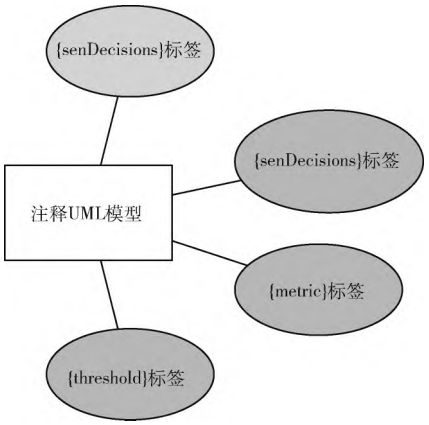

图 4　注释 UML 所包含的标签

(1) {senDecisions} 标签，它允许定义特定数据属性的设置值或调用操作事件是不应该基于特定的一些属性。

(2) {expData} 标签，它允许定义使用一些数据属性来区分是可以接受的状态。

(3) {metric} 和 {threshold} 标签。这些标签可用于根据模型的历史数据库识别属性的数据以防数据信息不可用。{metric} 和 {threshold} 标签允许指定要使用的相关度量和数字成为验证标准。对于灵活性问题，不限制规格 {metric} 标记到一组特定的指标，只要它具有到实现中的函数的映射即可，此标签就可以将任何字符串作为输入。

实现过程见 LTL 自动生成算法如算法 1 所示的第 (1) 行～第 (6) 行中。

1.2.2　对 UML 模型状态分类

接下来对于每个被定义为数据属性的，将执行以下操作：所有可以分配给 d_i 的可能值将从 UML 模型 m 中检索并存储在 rangeSet 中。rangeSet 会将验证模型用到的数据分配给 d_i。

当 $d_i \in$ DataSet 是数据属性时如式 (2) 所示

$$LTL_{g_i d_i} = \{(g_i \rightarrow \Diamond d_i == t_i), !(g_i \rightarrow \Diamond d_i == t_i)\} \tag{2}$$

在算法的假设中，可以分配给数据属性的可能值的数量是有限的。按照规则，属于 $LTL_{g_i d_i}$ 的每一对由两个声明组成：

(1) 如果 g_i 条件为真，数据属性 d_i 的值最终将等于 t_i。

(2) 如果 g_i 条件为假，数据属性 d_i 的值最终将等于 t_i。其中，t_i 是可分配给 s_i 的一个可能值。

当 d_i 是调用操作事件时如式 (3) 所示

$$LTL_{g_i d_i} = \left\{ \begin{array}{l} (g_i \rightarrow \Diamond event_{queue}?[call_d]_i == t_i), \\ !(g_i \rightarrow \Diamond event_{queue}?[call_d]_i == t_i) \end{array} \right\} \tag{3}$$

在这种情况下，LTL 由一对两个声明组成：

(1) 如果 g_i 条件为真，调用操作事件 $call\_d_i$ 最终将被添加到事件队列中。

(2) 如果 g_i 条件为假，调用操作事件 $call\_d_i$ 最终将添加到事件队列中。

实现过程见算法 1 的第 (4) 行～第 (16) 行中。

1.2.3 生成 LTL 语句

在算法中还声明了一个空集 LTL。该集合将用于算法来存储可以生成的所有 LTL。

对于图 3 中每个使用数据的条件 g_i，相对于 d_i，一条 LTL 语句将被定义如下：对于每个 value 属于 rangeSet，将定义一对声明和添加到 LTL 中。每对声明都应具有其对应的线性时间逻辑的格式。遍历 rangeSet 中的所有值后，该 LTL 将被添加到 LTL 总的集合中。对于图 3 中每个定义为调用操作的事件，将执行以下操作：对于每个使用的条件 g_i，一条 LTL 语句将被生成。每个生成的 LTL 将被添加到 LTL 总的集合中。

实现过程见算法 1 的第 (17) 行～第 (38) 行中。

算法 1：LTL 自动生成算法

(1)generateLTL (m, Requirements);
(2)Inputs: a UML model m and a Requirements
(3)Output: a set of batches of LTL
(4)P← ∅
(5)P←Participle(Requirements);
(6)sm←getIndividualFairnessStateMachine(m);
(7)DataSet←∅;
(8)guardSet←∅;
(9)DataSet←P. data;
(10)guardSet←getGaurds(m);
(11)LTL←∅;
(12)foreach state ∈ model do
(13) expSet←∅;
(14) expSet←getExplanatory(sm,state);
(15) usedConditionsSet←∅;
(16) usedConditionsSet ← getUsedConditions (m, P, DataSet,sm, guardSet);
(17) if state is a data attribute then
(18) rangeSet←∅;
(19) rangeSet←getRange(state, m);
(20) foreach g ∈ usedConditionsSet do
(21) batch←∅;
(22) foreach v ∈ rangeSet do
(23) batch. add(
(24) {g→<>state == v}, {!g→<>state==v});
(25) end
(26) LTL. add(batch);
(27) end
(28)end
(29)else
(30) foreach g∈ usedConditionSet do
(31) batch←∅;
(32) batch. add(
(33) {g→<>(event_queues?[call_s])},{!g→<>(event_queues?[call_s])});
(34) LTL. add(batch);
(35) end
(36) end
(37)end
(38)return LTL

2 实验和结果

本文采用模型验证的流程来验证生成 LTL 语言的准确性。模型检验流程如图 5 所示，展示的是模型验证的一般流程。首先利用转换转换规则将 UML 模型转换为 Promale 验证语言。根据本文生成的 LTL 语句，通过使用 SPIN model checker 来验证模型，以此来研判本文生成 LTL 语句的准确性。

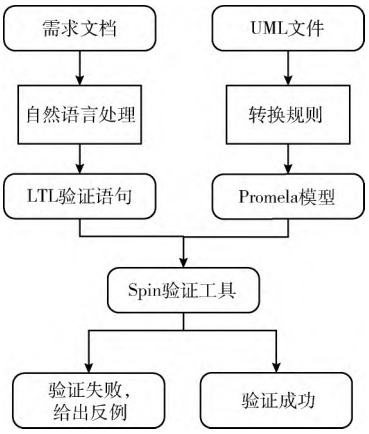

图 5 模型检验流程

2.1 LTL 自动生成案例

贷款管理系统基于真实业务流程模型，该模型由荷兰金融机构的事件日志生成。贷款管理系统包括两个主要流程，即贷款申请管理和风险分析管理。前者验证是否会接受贷款请求。后者为每个接受的贷款请求创建一个贷款申请，并进行风险分析，以决定是否批准该申请。本文从 StatlogCredit 数据集中提取数据，该数据集存储了 1000 条数据。实验的目标是检查如果两个贷款申请人的数据在特定的地方存在差异，是否会调用风险分析方法。

贷款管理系统设计模型如图 6 所示显示了贷款管理系

统设计的模型。实验检测 creditHistoryStatus 和 save 相关的部分。UML 类图，类图通过显示软件的类、它们的属性、操作以及类之间的关系来描述软件的结构。此类图中的一个类是“LoanManagement”。

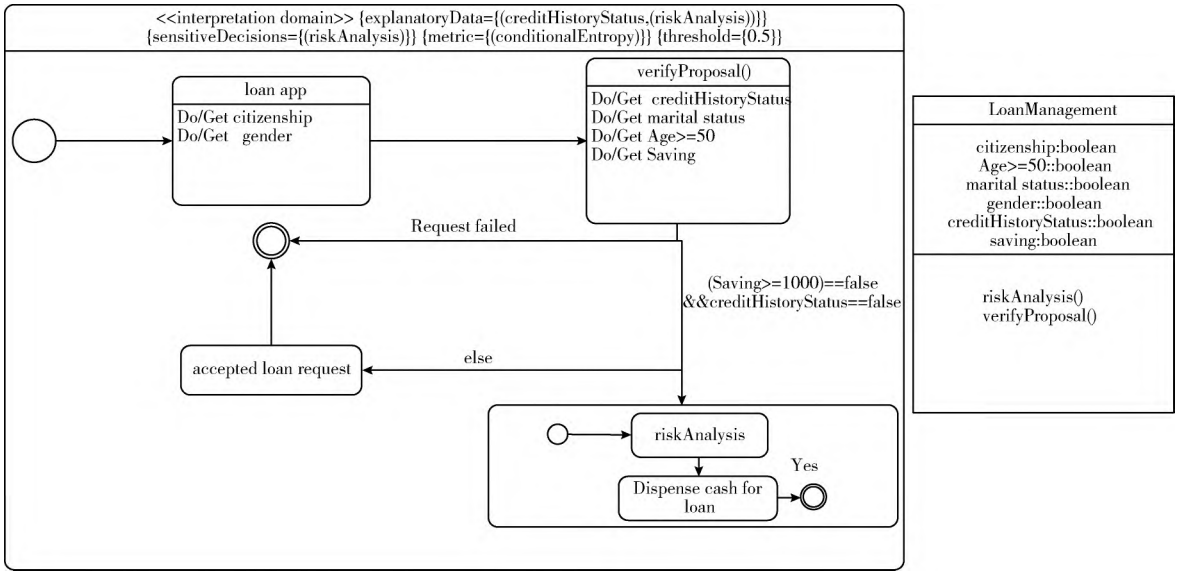

图 6 贷款管理系统设计模型

“citizenship”是一个字符串数据属性。代表贷款申请人的国籍。“gender”是一个布尔类型属性。表明一个申请人的性别，“男”为 true，“女”则为 false。“Age＞＝50”是一个布尔类型属性，表示一个贷款申请人的年龄是否大于 50 岁，“marital status”是一个布尔类型属性，表示贷款申请人是否已婚，“creditHistoryStatus”是一个布尔类型属性，如果贷款申请人有很好的信用记录则为“true”，否则为“false”。“saving”是一个整数类型属性，表示贷款申请人的存款数目。

操作的一个示例是“verifyProposal()”。在模型注释完成后可以表示“LoanManagement”类的对象可以接收到的信号，并在模型的生命周期内接收。物体对接收到的信号作出反应并找到到其类的指定行为。

图 6 是一个 UML 状态机它描述“LoanManagement”类行为。UML 状态机描述实体（例如对象）的状态序列，例如调用操作，连同它的响应动作。状态机包含状态和转换。状态表示的是执行状态机行为的一种情况，在此期间某些不变条件成立，状态用方框表示。

图 6 中的状态是“空闲”和“riskAnalysis”。该状态可以是调用操作或接收到的信号，它的动作可以是一个属性的数据分配，一个调用操作或发送信号。在图 6 中，如果一个对象在“verifyProposal()”中状态和条件“[creditHistoryStatus == false&&saving >= 1000 == false]”是 true，并进入“riskAnalysis”状态。若分析结果为“yes”则接受用户贷款。

贷款管理系统的需求如下：

系统应提供贷款申请管理和风险分析管理的功能，前者验证是否会接受贷款请求。后者为每个接受的贷款请求创建一个贷款申请，通过贷款人数据信息，并进行风险分析，以决定是否批准该申请。具体见贷款人数据信息说明见表 2。

表 2 贷款人数据信息说明

序号	名称	类型	长度	约束
1	citizenship	string	—	
2	Age＞＝50	boolean	—	必填项
3	creditHistoryStatus	boolean	—	是或否
4	gender	boolean	—	是或否
5	saving	int	—	必填项
6	marital status	boolean	—	必填项

2.2 生成线性时态逻辑

在贷款管理系统示例中，通过 1.2 中的基于注释 UML 模型的 LTL 生成算法，首先，以成对 LTL 语句的要求表达“信用状态”对“存款”的所有组合，其中每对 LTL 要求单独考虑关于一个可能属性的声明要求。其次验证是否恰好一对的声明得到满足（即，最终为真），而另一对的声明被违反（即，始终为假）。也就是说，“信用状态”与“存款”没有关联。这些产生的效果表示为成对的 LTL 声明，即 p1 和 p2。p1 的 LTL 对“信用状态”属性检查“调用风险分析”是否为真，p2 的 LTL 对“存款”属性检查其是否为假。

(ltl claim1 {(LoanRequest __creditHistoryStatus == true) -> <> (event__queues[1]?[call riskAnalysis])},

ltl claim2{(LoanRequest__creditHistoryStatus== true) -> <> (event. queues[1]?[call_riskAnalysis])}p1

(ltl claim3{(LoanRequest__saving>= 1000) -> <> (event__queues[1]?[call_riskAnalysis])},

ltl claim4{!(LoanRequest__saving>=1000)-><> (event_queues[1]?[call_riskAnalysis])})p2

2.3 验证生成线性时态逻辑的准确性

在生成一批 LTL 验证语句之后，实验根据 LTL 语句验证 UML 模型。利用 p1 和 p2 去验证图 6 的 UML 模型的正确性。验证结果见贷款管理系统验证结果见表 3。

表 3 贷款管理系统验证结果

部分	LTL 语句	结果
p1	ltlclaim1{(LoanRequest_saving>=1000) -> <> (event__queues[1]?[call_riskAnalysis])},	Satisfied
	ltlclaim2{!(LoanRequest_saving>=1000) -> <>(event_queues[1]?[call_riskAnalysis])	Violated
p2	ltlclaim3{(LoanRequest__creditHistoryStatus == true) -> <> (event__queues[1]?[call riskAnalysis])	Satisfied
	ltlclaim4{(LoanRequest__creditHistoryStatus == true) -> <> (event__queues[1]?[call riskAnalysis])	Violated

为了解释结果，本文用模型检查器为违反的 LTL 语句生成的反例。从生成的事件跟踪部分如算法 2 所示的是一个节选的痕迹 Spin 事件，作为违反示模型中 p1，p2 两个声明的反例。考虑图 6 的两条说明："LoanRequest_creditHistoryStatus = 0"，"LoanRequest_saving >=1000 审批与存款和信用状态之间存在联系。

算法 2：生成的事件跟踪部分代码

(1)claim claim2

(2)LoanPrposal 5 [9] 10 loanSystem. pml:14 ((!((!((!((LoanRe-quest_saving>=1000)))) && !(event__queues[1]?[4])))

(3)LoanPrposal 5 [1] 5 loanSystem. pml:14 (1)

(4)LoanPrposal 10 [10] 10 loanSystem. pml:19 (!(event_queues[1]?[4]))

(5)claim claim4

(6)LoanPrposal 5 [3] 10 loanSystem. pml:36 ((!((!((!((LoanRe-quest_creditHistoryStatus==1)))) && !(event_queues[1]?[4])))

(7)LoanPrposal 5 [1] 5 loanSystem. pml:36 (1)

(8)LoanPrposal 10 [4] 10 loanSystem. pml:41 (!(event_queues[1]?[4]))

2.4 实验结果

实验还利用上述方法对另外 4 个系统进行研究。

(1) 学校奖学金管理系统，该系统描述学生申请学校奖学金的情况。在系统的活动中，结果是奖学金申请成功与否，但是需求中要求它不应根据申请人的个人特征（如性别和身体健康状况）来影响奖学金申请成功。本文创建了一个数据集以检查其模型与需求的一致性。该数据集包含 20 个人的 6 个数据属性。

(2) 快递管理系统，以亚马逊配送管理系统为基础，展示了一个真实的事件。基于事件描述设计了交付系统的 UML 模型。亚马逊的软件为那些订单超过 35 美元，并且住在亚马逊商店附近的邮政区里的主要客户提供免费送货服务。本文创建了一个包含 30 个人 5 个数据属性的数据集来验证模型的一致性。

(3) 精简电梯模型，电梯的功能包括上行、下行、报警、显示、开/关门等。在验证过程中可以增加一条和某一行为需求描述相似的变迁，对其进行取反操作，观察模型检验能否检测出与需求不一致的行为。本文创建了一个包含 20 个人 4 个数据属性的数据集来验证模型的一致性。

(4) 前主桨舵机系统，前主桨舵机是飞行控制系统的执行机构，接受来自电传控制计算机的指令，进行相应的动作，拉动倾斜器前倾或后倾，以实现对飞机的俯仰控制。对其旋转直接驱动阀（rotary direct drive valve，RDDV）模块进行检验。本文创建了一个包含 15 条 3 个数据属性的数据集来验证模型的一致性。

UML 模型的概述见表 4。第二列显示了模型中 UML 元素的数量。例如，贷款系统模型由 27 个要素组成。元素的数量包括类、属性、操作、状态机、状态和转换的数量。第三列和第四列分别提供了模型生成的 LTL 语句数量和验证所需的时间。例如，贷款系统模型产生了 4 项 LTL 语句。Spin 模型检查器花了 36 s 来验证这 4 项声明，成功验证并发现了模型中存在的错误。这些测试工作是在一台配有 Intel i5 处理器和 16 GB 内存的计算机上进行的。

表 4 提供了分析模型中检测到的一致性违规数量。对于每个检测到的不一致行为，该表提供了：①违规行为发生的受保护数据；②违规行为的来源（即，导致违规行为的数据段）；③违规行为是由于数据流还是直接使用违规行为源而发生的。例如，在触发"riskAnalysis()"调用操作时，该表显示了贷款系统模型违反了需求模型一致性，其中两项错误违反由于"信用状态"和"存款"而发生的。

2.5 实验对比

由于现有针对软件需求文档 UML 模型等编程语言模型的形式化验证的研究比较缺乏，历史有关研究并不多，因此本文利用已有的 ST 语言模型形式化验证方法与本文提出的基于注释 UML 模型的 LTL 生成方法验证模型的时间效率进行分析和对比。文献 [19] 提出一种 ST 语言模型形式化验证方法。首先针对 ST 语言模型进行分解，通过数据

表 4 UML 模型的概述

模型	元素数目	LTL 语句数目	时间	验证部分	违规的数目	错误发生位置	违规行为的来源	违规行为发生的路径
贷款管理系统	27	4	36	riskAnalysis()	2	age	creditHistory-Statu&&saving	Data Flow
						citizenship	creditHistory-Status&&saving	
奖学金管理系统	65	8	48	scholarshipStatus	2	gender	height	Data Flow
快递管理系统	34	4	30	freeDeliveryStatus	1	ethnicity	zipCode	Direct Usage
精简电梯模型	35	1	20	passenger	1	position	state	Direct Usage
前主桨舵机系统	14	1	25	status	1	ACT	RDDV	Direct Usage

流分析得到模型程序依赖图，最后根据程序依赖图生成 NuSMV 的输入模型。ST 语言模型形式化验证方法的研究对象为逻辑控制器程序，与本文的研究对象模型与需求的一致性比较类似，其中实例的模型都利用程序语言编写。两种方法都利用了模型形式化检测工具，所以可以进行时间效率的对比，两种方法的对比见 ST 模型与 UML 模型形式化验证方法对比见表 5。

表 5 ST 模型与 UML 模型形式化验证方法对比

指标	UML 模型	ST 模型
描述对象	同等程序模型	逻辑控制器程序
中间状态	LTL 语句	程序依赖图
检测工具	Spin	NuSMV

实验将贷款管理系统、奖学金管理系统、快递管理系统、精简电梯模型、前主桨舵机系统共 5 个 UML 模型，将本文验证方法和 ST 程序模型验证方法进行对比实验，每种系统模型测试 3 次，然后计算模型验证花费时间的平均值。本文提出的基于注释的 UML 模型的 LTL 生成方法测试结果在 UML 模型验证性能分析图如图 7 所示；ST 语言模型形式化验证方法测试结果在 ST 模型验证性能分析如图 8 所示。在图 7 两条折线分别代表基于注释 UML 模型到验证语句的生成时间（UML_LTL 时间）以及总时间。图 8 所示两条折线分别表示 ST 模型到程序依赖图生成时间（PDG 时间），以及总时间。

由图 7 所示可得，UML 模型到 LTL 语句转换耗时较少，在 UML 模型代码行数较少时，生成中间模型耗时较少，并且随着 UML 模型规模的增加，总时间增长比较缓慢，说明本文提出的基于注释 UML 模型的 LTL 生成方法更适合规模较大模型的转换与验证。

由图 8 所示可得，由 ST 模型到中间状态生成消耗时间较多，NuSMV 模型验证花费时间较少。在 ST 模型规模不大时，程序依赖图生成过程花费的时间相对较多，随着 ST 模型规模的增加，总时间增长缓慢。说明针对模型规模较小的 ST 模型在生成中间状态过程中效率相对较低，不适合规模较大的模型进行转换。

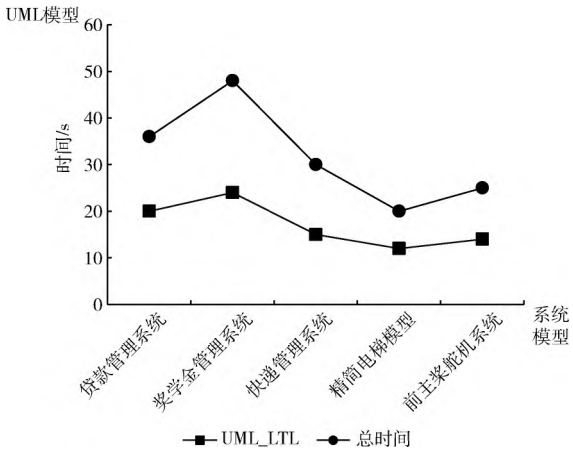

图 7 UML 模型验证性能分析

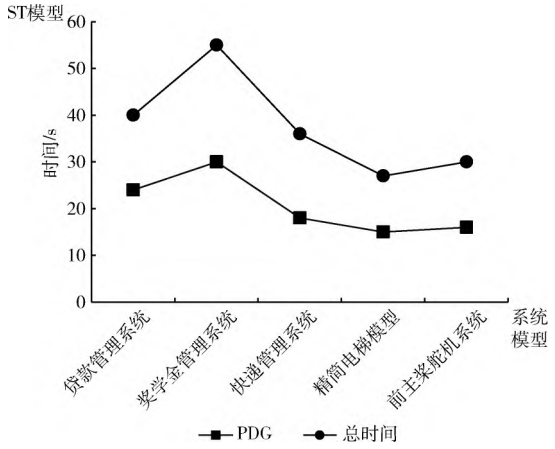

图 8 ST 模型验证性能分析

由对比结果可以了解到，本文的基于注释 UML 模型的 LTL 生成方法的时间效率优于已有的 ST 语言模型转换方法生成 NuSMV 模型的时间效率，并且适合各种规模模型的验证。

3 结束语

基于注释 UML 模型的 LTL 验证语句自动生成方法，经过了一个完整的模型验证过程，发现可以提高了模型检测的效率并能准确发现模型中存在的错误，减少了测试人员模型检测的时间。目前存在的问题是：本文的方法搜索单个属性 LTL 验证语句，而有时一个 LTL 验证语句需要用到多个属性。由于 UML 模型具有很大的可变性，因此无法保证生成完整的 LTL 语句。为了解决这些问题需要考虑多属性 LTL 语句，确定开发人员在建模过程中必须遵循的约束，以便验证生成的 LTL 验证语句的完整性。优化自然语言处理算法对需求文档关键词的提取过程，用以丰富生成的 LTL 验证语句中的属性。

参考文献：

[1] Gutierrez J，Harrenstein P，Wooldridge M. From model checking to equilibrium checking：Reactive modules for rational verification [J]. Artificial Intelligence，2017，248 (6)：123-157.

[2] Cai Mingyu，Xiao Shaoping，Kan Zhen. Reinforcement learning based temporal logic control with soft constraints using limit-deterministic generalized buchi automata [P]. China：10.48550/arXiv.2101.10284，2021.

[3] Guan Y，Guo J，Li Q. Formal verification of a hybrid iot operating system model [J]. IEEE Access，2021，PP (99)：1.

[4] WANG J，ZHAN NJ，FENG XY，et al. Over-view offormal methods [J]. Journal of Software，2019，30 (1)：33-61.

[5] Anevlavis T，Philippe M，Neider D，et al. Being correct is not enough：Efficient verification using robust linear temporal logic [J]. ACM Transactions on Computational Logic，2022，23 (2)：1-39.

[6] Thomas W. Ehrenfeucht，vaught，and the decidability of the weak monadic theory of successor [J]. ACM SIGLOG News，2018，5 (1)：14-18.

[7] Farias M，Martins AT，Ferreira F. The descriptive complexity of decision problems through logics with relational fixed-point and capturing results [J]. Electronic Notes in Theoretical Computer Science，2017，332 (5)：113-130.

[8] Roveri M，Ciccio CD，Francescomarino CD，et al. Computing unsatisfiable cores for LTLf specifications [J]. Journal of Systems Architecture，2022，114 (4)：10-37.

[9] Hirsch R，Mclean B. Disjoint-union partial algebras [J]. Logical Methods in Computer Ence，2017，13 (2)：2-10.

[10] Nazarpour H，Falcone Y，Bensalem S，et al. Concurrency-preserving and sound monitoring of multi-threaded component-based systems：Theory，algorithms，implementation，and evaluation [J]. Formal Aspects of Computing，2017，29 (5)：951-986.

[11] Al-Bataineh O，Rosenblum D. Efficient decentralized LTL monitoring framework using tableau approach [P]. Singapore：10.48550/arXiv.1803.02051，2018.

[12] Hansson H，Jonsson B. Formal aspects of computing a logic for reasoning about time and reliability [J]. Formal Aspects of Computing，2019，6 (5)：512-535.

[13] Min Z，Ying Y. Towards SMT-based LTL model checking of clock constraint specification language for real-time and embedded systems [C] //18th ACM SIGPLAN/SIGBED Conference. ACM，2017：61-70.

[14] Chen Z，Fang H，Luo X. Optimized step semantics encoding for bounded model checking of timed automata [C] //International Symposium on Theoretical Aspects of Software Engineering，2019：29-31.

[15] Mavridou A，Katis A，Giannakopoulou D，et al. From partial to global assume-guarantee contracts：Compositional realizability analysis in FRET [J]. Formal Methods，2021，20 (26)：503-523.

[16] Abbas M，Rioboo R，Ben-Yelles CB，et al. Formal modeling and verification of UML activity diagrams (UAD) with FoCaLiZe [J]. Journal of Systems Architecture，2020，114 (4)：1-14.

[17] Che W，Feng Y，Qin L，et al. N-LTP：An open-source neural language technology platform for Chinese [C] //Empirical Methods in Natural Language Processing，2021：42-49.

[18] YU Lamei，YANG Liangbin. TextRank keyword extraction method based on information entropy [J]. Computer and Digital Engineering，2022，50 (3)：516-519 (in Chinese). [于腊梅，杨良斌. 融合信息熵的 TextRank 关键词抽取方法 [J]. 计算机与数字工程，2022，50 (3)：516-519.]

[19] CHANG Tianyou，WEI Qiang，GENG Yangyang. Method of building PLC program model based on state transition [J]. Computer Application，2017，37 (12)：3574-3580 (in Chinese) [常天佑，魏强，耿洋洋. 基于状态转换的 PLC 程序模型构建方法 [J]. 计算机应用，2017，37 (12)：3574-3580.]