计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.200700184

一种 AltaRica 3.0 模型中类的平展化方法

祁 健[1,2] 胡 军[1,2] 谷青范[3] 荣 灏[3] 展万里[1,2] 董彦宏[1,2]
1 南京航空航天大学计算机科学与技术学院 南京 211106
2 软件新技术与产业化协同创新中心 南京 210007
3 中国航空无线电电子研究所 上海 200233
(flagship@nuaa.edu.cn)

摘 要 AltaRica是一类面向复杂安全关键系统的建模语言,卫士转换系统(Guarded Transition System,GTS)是最新的AltaRica 3.0的执行语义模型。AltaRica 3.0层次结构语法模型中类的平展化是将AltaRica 3.0语法模型转换为等价的平展化GTS语义模型过程中的一个重要步骤。文中提出了一种AltaRica 3.0模型中类的平展化优化方法。首先,设计专用的数据结构来存储AltaRica 3.0模型中类的语义结构,并对原有的ANTLR(Another Tool for Language Recognition)元语言描述的AltaRica 3.0模型颗粒度进行重新精化和定义;其次基于ANTLR生成相应的词法和语法分析器,并自动构造输入模型的语法树,通过对语法树的遍历,取得细粒度的类的关键信息并进行存储;然后设计了专用的算法,高效地实现了类的平展化过程;最后通过实例系统的分析,验证了所提方法的正确性和有效性。

关键词:ANTLR,AltaRica 3.0,GTS,模型转换,类的平展化
中图法分类号 TP311

Class Flattening Method for AltaRica 3.0 Model

QI Jian[1,2],HU Jun[1,2],GU Qing-fan[3],RONG Hao[3],ZHAN Wan-li[1,2] and DONG Yan-hong[1,2]
1 College of Computer Science and Technology,Nanjing University of Aeronautics and Astronautics,Nanjing 211106,China
2 Collaborative Innovation Center of Novel Software Technology and Industrialization,Nanjing 210007,China
3 China National Aeronautic Radio Electronics Research Institute,Shanghai 200233,China

Abstract AltaRica is a modeling language for complex safety-critical systems. Guarded Transition System(GTS) is the latest execution semantic model of AltaRica 3.0. The flattening of classes in the AltaRica 3.0 hierarchical syntax model is an important step in the process of transforming the AltaRica 3.0 syntax model into an equivalent flattened GTS semantic model. In this paper,a flattening optimization method for classes in AltaRica 3.0 models is proposed. Firstly,this paper designs a dedicated data structure to store the semantic structure of the class in the AltaRica 3.0 models,refines and defines the granularity of the AltaRica 3.0 model described by the original ANTLR(Another Tool for Language Recognition) meta language. Secondly,this paper generates the corresponding lexical and syntax analyzer based on ANTLR to automatically construct the syntax tree of the input model. Through traversing the syntax tree,the key information of fine-grained class is obtained and stored. Then,a dedicated algorithm is designed to realize the flattening process of the class efficiently. Finally,the correctness and effectiveness of this method are verified through the analysis of several example systems.

Keywords ANTLR,AltaRica 3.0,GTS,Model Transformation,Class flattening

1 引言

在复杂安全关键性系统[1]中,系统故障的发生往往会带来巨大的人员伤亡和财产损失,因此系统的可信性、可靠性及安全评估极其重要。基于模型的安全评估[2](Model-based Safety Assessment,MBSA)自1990年问世以来便受到了学术界和工业界的广泛关注,其核心思想是用高级描述形式编写模型,使其更接近于所研究系统的功能和物理架构[3],并在此基础上对系统进行安全评估和可靠性分析。

AltaRica 3.0[4-6]是基于MBSA的、致力于安全分析的高级建模语言,它由系统结构建模语言(System Structure Modeling Language,S2ML)和卫士转换系统GTS[7]组成,其中S2ML用于描述AltaRica 3.0的层次结构,GTS用于对AltaRica 3.0进行安全分析。为了对系统进行有效的安全评估

到稿日期:2020-07-29 返修日期:2020-11-19 本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。
基金项目:国家重点基础研究发展计划(973计划)项目(2014CB744900)
This work was supported by the National Basic Research Program of China(973 Program)(2014CB744900).
通信作者:胡军(hujun.nju@139.com)

与验证，需要将具有层次结构的 S2ML 模型转换为语义等价的 GTS 模型。

模型转换[8-9]是一个基于原模型生成目标模型的过程。模型转换可分为 4 类：在一种语言内部转换其组织形式，称为代码重构；在一种语言内部，通过补充新的内容由粗精确度转换到细精确度，称为模型精化；相同抽象程度的不同语言间的转换，称为语言迁移；由高抽象程度的语言向低抽象程度语言的转换，称为代码生成。本文的研究基于 AltaRica 3.0 语言，将具有层次结构的 S2ML 模型转换为形式化语义模型 GTS，属于代码重构的范畴。

S2ML 向 GTS 的转换过程被称为平展化，平展化的过程依次涉及层次结构的平展化、同步的平展化、隐藏 3 个步骤。其中，层次结构的平展化包括类的平展化和块的平展化。在 AltaRica 3.0 中，类是十分重要的组成部分，用于表示可重用组件，因此针对类设计一个有效的平展化方法显得尤为重要。文献[10]提出的类的平展化方法采取递归方法对类进行平展化，每次遍历到当前类中的继承类或类实例时，需要重新遍历所有继承类和类实例对应的类的语法树以获取需要的信息，时空复杂度较高。本文旨在设计一个专有的数据结构来保存类的信息，通过操作数据结构来实现类的平展化，以避免对语法树的重复遍历，进而降低方法的时空复杂度。

本文第 2 节讨论了本文研究所涉及的技术和语言基础；第 3 节建立了 S2ML 到 GTS 的映射规则，详细介绍了类的平展化方法的总体架构、算法思想及伪代码，并对算法的复杂度进行了分析；第 4 节通过实验对算法进行了评估，并通过一个具体实例对实验结果进行详细分析；第 5 节介绍国内外研究现状和本文的相关工作；最后总结本文，对未来工作进行了展望。

2 相关技术背景

2.1 ANTLR 概述

ANTLR 是一款语法分析器生成工具，可用于读取、处理、执行和翻译结构化的文本或二进制文件，它被广泛应用于学术领域和工业生产实践[11]。类的平展化方法涉及两种模型语言之间的识别和转换，因此采用 ANTLR 来实现原语言的识别和向目标语言的转换。

ANTLR 的主要解析过程依次为：词法分析、语法分析、抽象语法树(Abstract Syntax Tree，AST)分析。基于自定义的语法文件，通过上述步骤，程序能够识别和处理符合语法规范的文本，并依据转换规则，将原语言文本转换为目标语言的文本。

2.2 AltaRica 3.0 模型语言

AltaRica 3.0 是一种用于复杂系统的、基于事件的建模语言，同时具有面向对象和面向原型的特性。AltaRica 3.0 模型中的 class 用于表示可重用组件，可以被实例化(面向对象)，AltaRica 3.0 语言支持多重继承，即一个类可以继承多个父类；block 用于表示模型中具有唯一出现次数的组件(面向原型)，整个系统及子系统通常用 block 来表示。

AltaRica 3.0 由系统结构建模语言(S2ML)和卫士转换系统(GTS)组成，其中 S2ML 是描述 AltaRica 3.0 层次结构的模型语言，GTS 是用于对 AltaRica 3.0 进行安全分析的形式化语义模型。AltaRica 3.0 由两者组合而成，即 AltaRica 3.0＝S2ML＋GTS。

S2ML 由基本组件 class 和 block 组成。每个基本组件由变量(variable，包括状态变量和流变量)、事件(event，代表组件的故障和修复等)、转换(transition，表示组件的状态变化)、断言(assertion，描述变量之间的关系)、参数(parameter，表示组件的故障或修复率等)和观察者(observer，用户所要观察的量)组成。为了详细说明 S2ML 的组成信息，图 1 给出了一个冷却系统的示例，其详细建模过程见第 4 节中的实例研究部分。

该冷却系统由以下部分组成：1 个蓄水罐 Tank(储存所需的冷却剂)、4 个阀门 Valve(2 个控制输入的阀门和 2 个控制输出的阀门)、2 个水泵 Pump(将冷却剂从一端输送到另一端)、1 个反应堆 Reactor(该系统的冷却目标)。

该冷却系统的功能是将蓄水罐中的冷却剂通过水泵输送到反应堆，对反应堆进行降温处理。冷却剂可以通过两条线路 Line1 和 Line2 到达反应堆，当其中一条线路中的设备发生故障时，另一条线路仍然可以正常工作。

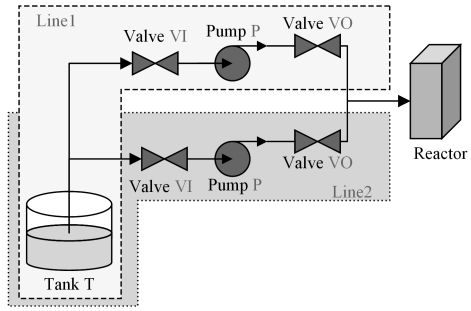

图 1 冷却系统

Fig. 1 Cooling system

冷却系统主块对应的 AltaRica 3.0 模型如图 2 所示。

```
block CoolingSystem
  Tank TK (evGetEmpty.delay = Dirac(0.0));
  block Line1
    embeds main.TK as t1;
    Valve VI, VO;
    Pump P;
    assertion
      VI.vfLeftFlow := t1.vfOutFlow;
      P.vfInFlow := VI.vfRightFlow;
      VO.vfLeftFlow := P.vfOutFlow;
  end
  block Line2
    embeds main.TK as t2;
    Valve VI, VO;
    Pump P;
    assertion
      VI.vfLeftFlow := t2.vfOutFlow;
      P.vfInFlow := VI.vfRightFlow;
      VO.vfLeftFlow := P.vfOutFlow;
  end
  block Reactor
    Boolean vfInFlow (reset = false);
  end
  observer Boolean oCooledReactor = Reactor.vfInFlow;
  parameter Real pPumpsCCF = 1.0e-6;
  event evPumpsCCF (delay = exponential(pPumpsCCF));
  transition
    evPumpsCCF: !Line1.P.evFailure & !Line2.P.evFailure;
  assertion
    Reactor.vfInFlow := Line1.VO.vfRightFlow or Line2.VO.vfRightFlow;
end
```

图 2 冷却系统主块的 AltaRica 3.0 模型

Fig. 2 AltaRica 3.0 model of cooling system's main block

GTS 是安全分析的一个重要形式，AltaRica 3.0 模型可

以被编译成 GTS 模型，进而利用相关的评估工具进行安全分析[12]。GTS 模型主要由变量、事件、转换、断言、参数、观察者以及变量的初始赋值组成。该冷却系统对应的 GTS 模型如图 3 所示。

```
block CoolingSystem
    Boolean Reactor.vfInFlow(reset = false);
    Boolean TK.vsIsEmpty (init = false);
    Boolean TK.vfOutFlow (reset = true);
    Boolean Line1.VI.vfLeftFlow, Line1.VI.vfRightFlow (reset = false);
    Boolean Line1.VI.vsWorking (init = true);
    Boolean Line1.VO.vfLeftFlow, Line1.VO.vfRightFlow (reset = false);
    Boolean Line1.VO.vsWorking (init = true);
    Boolean Line2.VI.vfLeftFlow, Line2.VI.vfRightFlow (reset = false);
    Boolean Line2.VI.vsWorking (init = true);
    Boolean Line2.VO.vfLeftFlow, Line2.VO.vfRightFlow (reset = false);
    Boolean Line2.VO.vsWorking (init = true);
    Boolean Line1.P.vfInFlow, Line1.P.vfOutFlow (reset = false);
    Boolean Line1.P.vsWorking (init = true);
    Boolean Line2.P.vfInFlow, Line1.P.vfOutFlow (reset = false);
    Boolean Line2.P.vsWorking (init = true);
    observer Boolean oCooledReactor = Reactor.vfInFlow;
    parameter Real pPumpsCCF = 1.0e-6;
    parameter Real pLambda = 1.0e-5;
    parameter Real pMu = 1.0e-2;
    event evPumpsCCF (delay = exponential(pPumpsCCF));
    event TK.evGetEmpty;
    event Line1.VI.evFailure (delay = exponential(pLambda));
    event Line1.VI.evRepair (delay = exponential(pMu));
    event Line1.VO.evFailure (delay = exponential(pLambda));
    event Line1.VO.evRepair (delay = exponential(pMu));
    event Line2.VI.evFailure (delay = exponential(pLambda));
    event Line2.VI.evRepair (delay = exponential(pMu));
    event Line2.VO.evFailure (delay = exponential(pLambda));
    event Line2.VO.evRepair (delay = exponential(pMu));
    event Line1.P.evFailure (delay = exponential(pLambda));
    event Line1.P.evRepair (delay = exponential(pMu));
    event Line2.P.evFailure (delay = exponential(pLambda));
    event Line2.P.evRepair (delay = exponential(pMu));

  transition
    evPumpsCCF: Line1.P.Working and Line2.P.Working ->
      {Line1.P.vsWorking:= false; Line2.P.vsWorking:=false;}
    TK.evGetEmpty: not TK.vsIsEmpty -> TK.vsIsEmpty := true;
    Line1.VI.evFailure: Line1.VI.vsWorking -> Line1.VI.vsWorking := false;
    Line1.VI.evRepair: not Line1.VI.vsWorking -> Line1.VI.vsWorking := true;
    Line1.VO.evFailure: Line1.VO.vsWorking -> Line1.VO.vsWorking := false;
    Line1.VO.evRepair: not Line1.VO.vsWorking -> Line1.VO.vsWorking := true;
    Line2.VI.evFailure: Line2.VI.vsWorking -> Line2.VI.vsWorking := false;
    Line2.VI.evRepair: not Line2.VI.vsWorking -> Line2.VI.vsWorking := true;
    Line2.VO.evFailure: Line2.VO.vsWorking -> Line2.VO.vsWorking := false;
    Line2.VO.evRepair: not Line2.VO.vsWorking -> Line2.VO.vsWorking := true;
    Line1.P.evFailure: Line1.P.vsWorking -> Line1.P.vsWorking := false;
    Line1.P.evRepair: not Line1.P.vsWorking -> Line1.P.vsWorking := true;
    Line2.P.evFailure: Line2.P.vsWorking -> Line2.P.vsWorking := false;
    Line2.P.evRepair: not Line2.P.vsWorking -> Line2.P.vsWorking := true;
  assertion
    Reactor.vfInFlow := Line1.VO.vfRightFlow or Line2.VO.vfRightFlow;
    Line1.VI.vfLeftFlow := T.vfOutFlow;
    Line1.P.vfInFlow := Line1.VI.vfRightFlow;
    Line1.VO.vfLeftFlow := Line1.P.vfOutFlow;
    Line2.VI.vfLeftFlow := T.vfOutFlow;
    Line2.P.vfInFlow := Line2.VI.vfRightFlow;
    Line2.VO.vfLeftFlow := Line2.P.vfOutFlow;
    TK.vfOutFlow := not TK.vsIsEmpty;
    if Line1.VI.vsWorking then Line1.VI.vfLeftFlow := Line1.VI.vfRightFlow;
    if Line1.VO.vsWorking then Line1.VO.vfLeftFlow := Line1.VO.vfRightFlow;
    if Line2.VI.vsWorking then Line2.VI.vfLeftFlow := Line2.VI.vfRightFlow;
    if Line2.VO.vsWorking then Line2.VO.vfLeftFlow := Line2.VO.vfRightFlow;
    Line1.P.vfOutFlow := if Line1.P.vsWorking then Line1.P.vfInFlow else false;
    Line2.P.vfOutFlow := if Line2.P.vsWorking then Line2.P.vfInFlow else false;
end
```

图 3　冷却系统的 GTS 模型

Fig. 3　GTS model of cooling system

3　AltaRica 3.0 中类的平展化方法框架

本节将对类的平展化方法进行详细介绍，首先列举 AltaRica 3.0 模型中 S2ML 到 GTS 的映射规则；其次描述类的平展化方法的总体架构和实现步骤；然后介绍类的平展化算法的设计思想；最后实现平展化算法，给出了算法的伪代码，并对算法的时空复杂度进行分析。

3.1　S2ML 到 GTS 的映射规则

模型转换必须保持两种模型语义上的一致性，因此，为了从形式上证明类的平展化方法的正确性和有效性，本节详细说明了 S2ML 和 GTS 的组成结构，对两种模型的语义进行了分析和比较，并在此基础上建立从 S2ML 转换到 GTS 的映射规则。

定义 1　系统结构建模语言(S2ML)是一个二元组：

$S=\langle C,B\rangle$

(1)C(Classes)是一组组件类，用于表示可重用组件模型。每个 class 是一个四元组，即 $C=\langle C_E,D,T,A\rangle$。

1)C_E 是一组扩展类，即类 C 的父类列表；

2)D(declarations)是一组声明元素，包括类的实例和原子元素($variables$，$events$，$parameters$，$observers$)。

3)行为子句，包括一组转换 T(transitions)和一组断言 A(assertions)。

(2)B(Block)是一个系统块，是整个系统的模型。一个 block 是一个五元组，即 $B=\langle B_A,B_E,D,T,A\rangle$。

1)B_A 是一组声明块，即 B 的所有子块；

2)B_E 是一组嵌入对象(块或类的实例)；

3)D(declarations)是一组声明元素，包括类的实例和原子元素($variables$，$events$，$parameters$，$observer$)；

4)行为子句，包括一组转换 T(transitions)和一组断言 A(assertions)。

定义 2　卫士转换系统(GTS)是一个七元组：

$G=\langle V,O,P,E,T,A,\xi\rangle$

(1)V(variables)是一组变量，包括状态变量(state variable)和流变量(flow variable)。

(2)O(observers)是一组观察者，观察者是一种不能用于描述系统行为的流变量，它是需要被观察的量。

(3)P(parameters)是一组参数，参数是一个常量值，它完全用于确定延迟常数。

(4)E(events)是一组代表事件的符号。

(5)T(transitions)是一组转换，通常表示为 $e:G\rightarrow P$，其中 e 是 E 的事件，G 是守卫(构建于集合 V 上的布尔表达式)，P 是对状态变量执行的动作，用于计算系统的新状态。$e:G\rightarrow P$ 的含义为：当 G 为 true 时，执行指令 P。

(6)A(assertions)是一组断言，是建立在集合 V 中变量上的指令，用于根据状态变量的值计算流变量的值。

(7)ξ 是 V 中变量的赋值，规定变量的初始值。

定义 3　由 S2ML 转换为 GTS 的过程称为平展化，其映射规则如表 1 所列。

假设一个真实的 S2ML 模型是 $S=\langle C,B\rangle$，其中 $C=\langle C_E,D_C,T_C,A_C\rangle$，$B=\langle B_A,B_E,D_B,T_B,A_B\rangle$；假设转换后的 GTS 模型是 $G=\langle V,O,P,E,T,A,\xi\rangle$。对于 S2ML 模型中类 C 中的 C_E(一组扩展类)以及块 B 中的 B_A(一组子块)和 B_E(一组嵌入对象)，其基本组成元素(变量、事件、参数、观察者、转换、断言和属性)分别表示为元素名加下标的形式，如变量

对应的名称分别为 V_{CE}，V_{BA} 和 V_{BE}。

表 1 S2ML 模型和 GTS 模型的映射规则

Table 1 Mapping rules of S2ML model and GTS model

S2ML 模型	GTS 模型
class 和 block 中的 state variable	state variable
class 和 block 中的 flow variable	flow variable
class 和 block 中的 observer	observer
class 和 block 中的 parameter	parameter
class 和 block 中的 event	event
class 和 block 中的 transition	transition
class 和 block 中的 assertion	assertion
class 和 block 中声明元素的属性 attribute	ξ

S 中的声明元素 $D(D_C \cup D_B)$ 由变量、事件、参数和观察者组成，即 $D_C=\langle V_C, E_C, P_C, O_C\rangle$，$D_B=\langle V_B, E_B, P_B, O_B\rangle$。$S$ 中的变量、事件、参数和观察者分别对应 G 中的 V，E，P 和 O，其中 $V=V_C \cup V_B \cup V_{CE} \cup V_{BA} \cup V_{BE}$，$E=E_C \cup E_B \cup E_{CE} \cup E_{BA} \cup E_{BE}$，$P=P_C \cup P_B \cup P_{CE} \cup P_{BA} \cup P_{BE}$，$O=O_C \cup O_B \cup O_{CE} \cup O_{BA} \cup O_{BE}$；$S$ 中的转换对应 G 中的 T，即 $T=T_C \cup T_B \cup T_{CE} \cup T_{BA} \cup T_{BE}$；$S$ 中的断言对应 G 中的 A，即 $A=A_C \cup A_B \cup A_{CE} \cup A_{BA} \cup A_{BE}$；$S$ 中声明元素中变量的属性信息 Att 是一组变量的赋值，对应 G 中的 ξ，即 $\xi=Att_C \cup Att_B \cup Att_{CE} \cup Att_{BA} \cup Att_{BE}$。

从 S2ML 到 GTS 的转换实际上是在 AltaRica 3.0 语言内部对其组织形式的转换。通过上述分析发现：对于 S2ML 模型中的每类元素，GTS 模型中都有唯一与之对应的元素。这说明两种模型在语义上具有一致性，能够进行正确的转换。

3.2 类的平展化方法的总体架构

文献[10]提出的类的平展化算法，在处理继承类和声明元素中的类实例时，需要反复地遍历语法分析树以获取相应类的信息，算法的时间复杂度较高。因此，为了减少对语法分析树的遍历次数，本文为 Class 类设计了一个专用的数据结构，以实现遍历两次语法分析树获取所有类的信息，并将其保存到数据结构 Map〈String, ClassInfo〉*allClassMap* 中，进而通过操作已获取的数据结构对所有类进行平展化。类的平展化方法的实现主要包括 5 个模块，分别为语法解析模块、读取信息模块、平展化模块（核心模块）、信息完整化模块、信息输出模块。其具体步骤如图 4 所示。

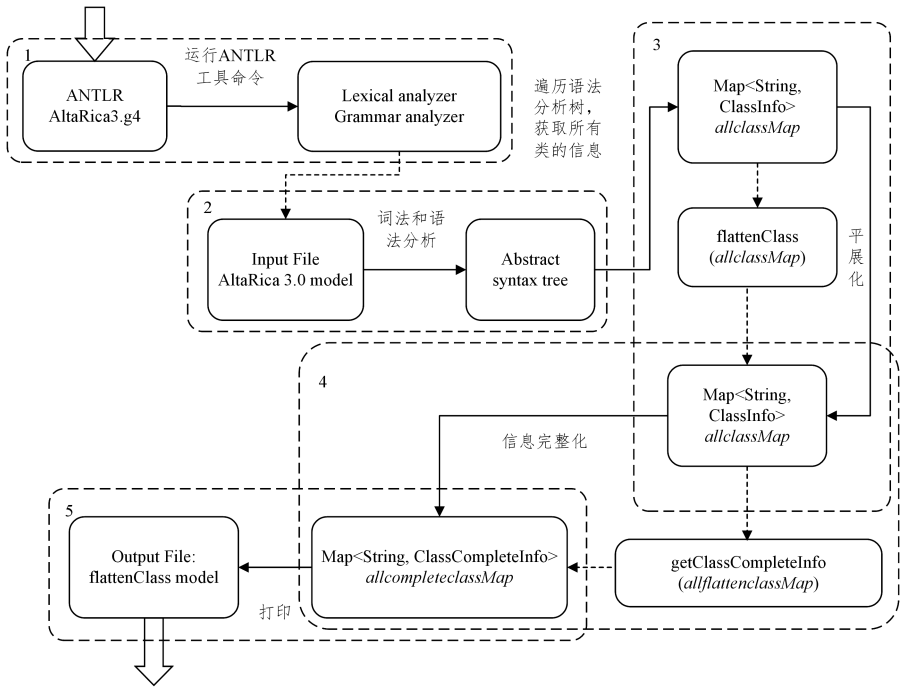

图 4 AltaRica 3.0 类的平展化方法框架

Fig. 4 Framework of class flattening algorithm of AltaRica 3.0

步骤 1 基于自定义的语法文件 AltaRica3.g4，运行 ANTLR 工具命令，生成一个由 AltaRica3Lexer.java 和 AltaRica3Parser.java 组成的、可以运行的语法识别程序，以及一组由 AltaRica3BaseListener.java 和 AltaRica3Listener.java 组成的监听器程序，如图 5 所示。

图 5 ANTLR 自动生成的程序

Fig. 5 Programs generated automatically by ANTLR

步骤 2 读取存有某一实例的 AltaRica 3.0 模型的输入文件，并对其进行如下处理：

(1)新建词法分析器处理输入的字符流；

(2)新建一个词法符号的缓冲区，用于存储词法分析器生成的词法符号；

(3)新建一个语法分析器，处理词法符号缓冲区中的内容；

(4)新建并获取语法分析树；

(5)新建一个标准的遍历器；

(6)遍历语法分析树，并调用自定义的方法获取相应的信息。

为了在遍历类中的声明元素时，准确地判断其是对象（类的实例）或是普通变量，需要对所有类的语法分析树进行一次预遍历，从而获取所有类的类名，并把所有类名存放到

ArrayList 中。当遍历到声明元素时，判断该声明元素的类型是否存在于 ArrayList 中，若存在，则该声明元素为对象；否则该声明元素为普通变量。

第二次遍历语法分析树获取每个类中的所有关键结点信息，并将其保存到数据结构 Map〈String，ClassInfo〉*allClassMap* 中，其中 String 存放类名，ClassInfo 存放该类名对应的类的信息。

步骤 3　对数据结构中的所有类进行平展化处理，包括对所有类的扩展类、声明元素中的对象（类的实例）和原子元素，以及转换和断言进行相应的处理，将平展化后的信息仍然保存到数据结构 Map〈String，ClassInfo〉*allClassMap* 中。其方法设计和实现将在第 3.3 节和第 3.4 节讨论。

步骤 4　原有保存类信息的数据结构 ClassInfo，仅保存了语法分析树中关键的结点信息，其他符号信息如“if”“：”“→”“(”等均未保存，因此在输出平展化信息之前，需要对 ClassInfo 的信息进行完整化。此外，为了方便输出操作，所有原子声明元素、转换和断言的完整化信息被分别保存到相应的 ArrayList 中，最终生成保存了完整信息的 ClassCompleteInfo。

步骤 5　为了检验算法的正确性，本文将所有类的 ClassCompleteInfo 中的信息打印到指定文件 Class_CompleteInfo.alt 中，与手工进行的类的平展化结果进行对比。

3.3　Class 平展化算法的设计

文献[10]给出的类的定义是一个五元组，即 $C=\langle C_E, B_A, D, T, A\rangle$，其中 B_A 是一组声明块。通过对多个中小规模系统的分析后发现并不存在类中嵌套块的情形，为避免类与块的嵌套定义，此处对类的组成进行合理简化，即去掉 B_A。考虑一个 $C=\langle C_E, D, T, A\rangle$ 组成的类：C_E 为一组扩展类，即类 C 的父类集合；D 为一组声明元素，包括对象（类的实例）和原子元素（变量、事件、参数和观察者）；T 为转换；A 为断言。

C_{flat} 是类 C 的平展化形式，从 C 到 C_{flat} 可通过以下步骤获得。

步骤 1　深度复制类 C，将其信息保存到 C_{flat} 中。

该步骤实现了将类 C 中的简单变量声明（类型为 Boolean，Integer，Real，Symbol 或用户自定义的域）、事件声明、参数声明和观察者声明复制到 C_{flat} 中。该步骤实现了对行为语句的平展化：类 C 的转换 T 直接复制到 C_{flat} 中；类 C 的断言 A 直接复制到 C_{flat} 中。

步骤 2　执行对一组扩展类 C_E 的扩展：C_E 即类 C 的父类集合，对于 C_E 中的每个类 K，需要将类 K 的副本 K_{flat} 添加到 C_{flat} 中。

步骤 3　处理 D 中的对象（嵌套的其他类 Q 的实例）：对于每个类 Q 的每个实例 q，依次将 Q 的副本 Q_{flat} 添加到 C_{flat} 中。在复制过程中，Q 中所有已命名的对象（变量、事件、参数、观察者及其在表达式和同步中的引用）前都要添加上带有点后缀的实例名，如 $q.inFlow$。

为了清楚地展示步骤 2 对一组扩展类 C_E 的扩展，以及步骤 3 对声明元素 D 中的对象的处理，下文将分别以两个简单的示例来进行说明。

首先，图 6 给出了一个继承类的示例。类 A 具有 2 个原子声明元素，分别为布尔变量 *working* 和事件 *failure*，事件 *failure* 对应一个转换 *failure*：*working*→*working*：=false；类 B 继承了类 A，具有 2 个原子声明元素 *inFlow* 和 *outFlow*，此外还有一个断言 *outFlow*：=if *working* then *inFlow* else false。

```
class A
    Boolean working(init=true);
    event failure;
    transition
        failure : working->working:=false;
end
class B
    extends A;
    Boolean inFlow,outFlow(reset=false);
    assertion
        outFlow:=if working then inFlow else false;
end
```

图 6　类 A 和类 B

Fig. 6　Class A and Class B

由于类 A 不是继承类且不存在其他类实例，说明类 A 本身就是一个平展化的类，因此仅需将类 A 中的所有属性直接复制到类 B 中，其平展化结果如图 7 所示，类 B 中新增的属性加粗标注。

```
class B
    Boolean working(init=true);
    Boolean inFlow,outFlow(reset=false);
    event failure;
    transition
        failure : working->working:=false;
    assertion
        outFlow:=if working then inFlow else false;
end
```

图 7　类 B 的平展化结果

Fig. 7　Flattening result of Class B

图 8 给出了实例类的示例，类 A 的所有属性同上；类 C 具有 3 个声明元素，包括 1 个类 A 的实例 a，2 个原子元素 *inFlow* 和 *outFlow*，此外还有一个断言 *outFlow*：=if *working* then *inFlow* else false。

```
class A
    Boolean working(init=true);
    event failure;
    transition
        failure : working->working:=false;
end
class C
    A a;
    Boolean inFlow,outFlow(reset=false);
    assertion
        outFlow:=if working then inFlow else false;
end
```

图 8　类 A 和类 C

Fig. 8　Class A and Class C

```
class C
    Boolean a.working(init=true);
    Boolean inFlow,outFlow(reset=false);
    event a.failure;
    transition
        a.failure : a.working->a.working:=false;
    assertion
        outFlow:=if working then inFlow else false;
end
```

图 9　类 C 的平展化结果

Fig. 9　Flattening result of Class C

同样地，由于类 A 不是继承类且不存在其他类实例，说明类 A 本身就是一个平展化的类。而类 C 中有类 A 的实例

a，因此，我们需要新建一个类 A 的副本，并在类 A 的副本中所有已命名对象前添加上带有点后缀的实例名 a，然后将类 A 的副本复制到类 C 中，类 C 的平展化结果如图 9 所示，类 C 中新增的属性加粗标注。

3.4 Class 平展化算法的实现

类的平展化方法的核心模块是平展化模块，该模块的核心算法称为类的平展化算法，该算法对存放了类的关键结点信息的数据结构 Map〈String，ClassInfo〉*allClassMap* 进行平展化处理，具体步骤如算法 1 所示。

算法 1 类的平展化算法

输入：Classes' AltaRica 3.0 model(C)

输出：Classes' flattened AltaRica 3.0 model(C_{flat})

```
1. function flattenClass(allClassMap)
2.     for all C∈allClassMap ＃遍历所有类
3.       C_flat←C. deepCopt(); ＃深度复制类 C
4.       if C. isExistSuperClass then ＃平展化继承类
5.         for all classname∈superClassList ＃遍历父类
6.           superClass←C. getSuperclass(classname);
7.           mergeClass(superClass, C_flat); ＃合并父类信息
8.         done
9.       end if
10.    if C. isExistClassInstance then ＃平展化实例类
11.      classInstanceMap←C. getClassInstanceMap();
12.      for all instanceList∈classInstanceMap
13.        C_instance←getInstanceClass(); ＃获取实例类
14.        for all q∈instanceList ＃遍历类的所有实例
15.          C_clone←C_instance. deepCopt();
16.          addPrefix(q, C_clone); ＃添加前缀
17.          mergeClass(C_withPrefix, C_flat);
18.        done
19.      done
20.    end if
21.    allClassMap. put(C_flat); ＃C_flat存入 allClassMap
22. done
23. end function
```

类的平展化算法主要分为以下 4 个步骤。

(1)复制。类 C 中的简单变量、事件、参数和观察者声明，以及行为语句(转换和断言)在平展化过程中不会发生变化，直接复制到保存平展化信息的类 C_{flat} 中。算法 1 中的第 3 行使用深度复制方法将类 C 中的所有信息复制到 C_{flat} 中。

(2)平展化继承类。如果类 C 是继承类，则将 C 的所有父类信息依次拷贝到 C_{flat} 中。算法 1 中的第 4－9 行调用方法获取到 C 的所有父类的信息并将其合并到 C_{flat} 中。

(3)平展化实例类。如果类 C 中存在其他类的实例，则在所有实例对应类的副本中所有已命名对象前添加上带有点后缀的实例名，然后合并到 C_{flat} 中。算法 1 中的第 10－20 行实现了对类 C 中所有实例的平展化。

(4)将平展化后的类 C_{flat} 添加到 *allClassMap* 中，替换掉原来未平展化的类。

针对一个具体的 Class，算法的时间主要消耗在以下几个部分：首先，遍历类 C 中的所有父类，并将所有平展化后的父类信息拷贝到 C_{flat} 中，假设平均每个类继承 m 个父类，平均每个父类中有 n 条信息，则当前的时间复杂度为 $O(m*n)$。为避免递归，需要对用户输入的文本信息进行限制，即要求子类信息在父类信息之后，从而保证在对子类进行平展化时，其父类信息已经进行平展化处理。去除递归方法后，算法的时间复杂度大幅降低。其次，需要遍历类 C 中的所有类实例，将对应的平展化后的类 Q_{flat} 添加上前缀并复制到 C_{flat} 中。假设平均每个类中有 k 个类实例，平均每个类实例对应 z 条需要添加前缀的信息，平均每个类实例对应的类有 p 条需要拷贝的信息，易知 $p>z$，因此时间复杂度为 $O(k*p)$。同样地，为避免递归方法，需要对用户输入的文本信息进行限制，即要求被引用的类在引用类之前，从而保证在平展化时类中引用的类实例对应的类已经进行了平展化处理。综上所述，该算法的时间复杂度为 $O(n^2)$。

该算法的空间复杂度主要体现在存放所有类信息的数据结构 *allClassMap*，其中存放了类名及其对应的 ClassInfo，数据结构 ClassInfo 包含了一个类的所有关键信息，主要包括类名、父类名列表、类实例映射、基本声明元素列表、事件列表、转换列表和断言列表。由于映射的存在，其空间复杂度为 $O(n^2)$。在对每个 ClassInfo 进行平展化的过程中，为了提高效率，即避免将 ClassInfo 中的普通声明元素、转换和断言拷贝到 C_{flat} 中，本文算法深度复制了每个 ClassInfo，保存平展化过程中不变的信息(普通声明元素、转换和断言)，修改或添加平展化后会变化的信息(继承类、类实例)。假设共有常数 q 个类，易知每个 ClassInfo 所占空间为 $O(n^2)$，因此该算法的空间复杂度为 $O(q*n^2)$。综上所述，该算法的空间复杂度为 $O(n^2)$。

4 实例研究

本节主要通过实验来对算法的有效性进行验证。

4.1 实验对象和环境

实验采用第 2.2 节中的冷却系统(CoolingSystem)、文献[13]中的机轮刹车系统(WheelBrakeSystem)、文献[3]中的灌溉系统(IrrigationSystem)、文献[14]中的起落架系统(LandGearSystem)来对类的平展化算法进行验证。AltaRica 3.0 模型信息如表 2 所列。

表 2 AltaRica 3.0 模型信息

Table 2 Information of AltaRica 3.0 model

模型	Class 数量	扩展类数量	类实例数量
IrrigationSystem	3	0	0
CoolingSystem	4	2	0
WheelBrakeSystem	17	6	14
LandGearSytem	18	0	21

本实验运用一种基于 ANLTR 的 AltaRica 3.0 模型中类的平展化方法对 4 个模型进行类的平展化操作，4 个模型的主要区别是 Class 数量、扩展类数量和类实例数量，以验证不同复杂度下算法的平展化效率。

本文所涉及的实验数据均是在一台 CPU 为 Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz、内存为 8GB、操作系统为 Windows 10 家庭版的计算机上得到的，算法基于 eclipse 平台，使用 Java 语言来实现。

4.2　实验结果及分析

本节将对实验结果进行分析，并通过冷却系统实例来详细说明使用 AltaRica 3.0 语言对系统进行建模的过程。

4.2.1　实验结果

为了定量分析类的平展化方法的时间复杂度和空间复杂度，实验对 4 个模型调用类的平展化方法消耗的时间和内存进行了记录，实验的消耗信息如表 3 所列。

表 3　不同模型在类的平展化过程中的时间和内存消耗

Table 3　Time and memory consumption of different models in the process of class flattening

模型	时间消耗/ms	内存消耗/kB
IrrigationSystem	399	20 829
CoolingSystem	430	20 211
WheelBrakeSystem	1 895	47 350
LandGearSytem	4 759	100 375

如表 3 和图 10 所示，在扩展类数量和类实例数量相当的情况下，灌溉系统(IrrigationSystem)和冷却系统(CoolingSystem)的时间消耗接近；相比之下，机轮刹车系统(WheelBrakeSystem)中扩展类数量和类实例数量较多，因此在进行类的平展化时，本文方法的时间消耗较高；起落架系统(LandGearSystem)中没有扩展类，但类实例数量较多，并且起落架系统每个类的规模较大，因此该系统的时间消耗比其他 3 个系统高得多。因此，影响类的平展化方法的时间消耗的主要因素是扩展类数量、类实例数量以及类的规模。

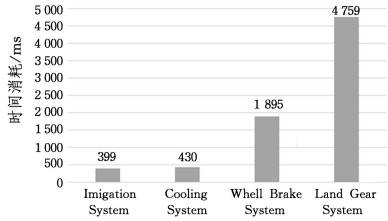

图 10　不同模型在类的平展化过程中的时间消耗

Fig. 10　Time consumption of different models in process of class flattening

如表 3 和图 11 所示，在 Class 数量相当的情况下，灌溉系统(IrrigationSystem)和冷却系统(CoolingSystem)的内存消耗接近；相比之下，机轮刹车系统(WheelBrakeSystem)和起落架系统(LandGearSystem)的 Class 数量较多，在进行类的平展化时内存消耗较大；此外，虽然机轮刹车系统和起落架系统的 Class 数量相当，但由于起落架系统每个 Class 的规模较大，因此内存消耗更大。因此，影响类的平展化方法的内存消耗的主要因素是类的数量和类的规模。

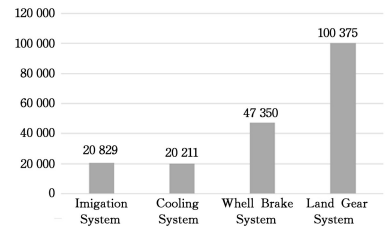

图 11　不同模型在类的平展化过程中的内存消耗

Fig. 11　Memory consumption of different models in process of class flattening

4.2.2　实例分析

由于篇幅限制，本节仅选取第 2.2 节所介绍的冷却系统的建模过程和平展化结果进行详细说明。

首先，由于水泵和阀门在发生故障后可以进行修复，因此需要构建通用的可修复组件类 RepairableComponent，状态转换过程如图 12 所示。

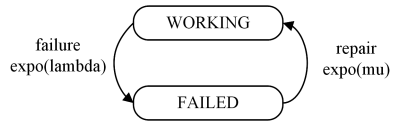

图 12　可修复组件的状态转换图

Fig. 12　States' transitions of repairable component

当事件 *failure* 发生时，组件由 WORKING 状态变为 FAILED 状态；当事件 *repair* 发生时，组件由 FAILED 状态变为 WORKING 状态。该组件具有属性 *vsWorking*(组件状态)、*pLambda*(故障率)、*pMu*(修复率)。当事件 *evFailure* 发生时，*vsWorking* 的值由 true 变为 false；当事件 *evRepair* 发生时，*vsWorking* 的值由 false 变为 true。可修复组件对应的 AltaRica 3.0 模型如图 13 所示。

```
class RepairableComponent
  Boolean vsWorking (init = true);
  parameter Real pLambda = 1.0e-5;
  parameter Real pMu = 1.0e-2;
  event evFailure (delay = exponential(pLambda));
  event evRepair(delay = exponential(pMu))
  transition
    evFailure: vsWorking -> vsWorking := false;
    evRepair: not vsWorking -> vsWorking := true;
end
```

图 13　可修复组件的 AltaRica 3.0 模型

Fig. 13　AltaRica 3.0 model of repairable component

然后，对水泵进行建模。水泵除了具有可修复组件的特性外，还具有独有的属性，如 *vfInFlow*(输入流)、*vfOutFlow*(输出流)。当泵正常工作时，输出流等于输入流，否则输出流为 false。水泵对应的 AltaRica 3.0 模型如图 14 所示。

```
class Pump
  extends RepairableComponent;
  Boolean vfInFlow, vfOutFlow (reset = false);
  assertion
    vfOutFlow := if vsWorking then vfInFlow else false;
end
```

图 14　水泵的 AltaRica 3.0 模型

Fig. 14　AltaRica 3.0 model of pump

接着，对阀门进行建模。阀门除了具有可修复组件的特性外，还具有其他的属性，如 *vfLeftFlow*，*vfRightFlow*。与水泵的不同之处在于阀门可以双向流通，即冷却剂可以从阀门的左侧流向右侧，也可以从右侧流向左侧。阀门对应的 AltaRica 3.0 模型如图 15 所示。

```
class Valve
  extends RepairableComponent (pLambda = 1.0e-4);
  Boolean vfLeftFlow, vfRightFlow (reset = false);
  assertion
    if vsWorking then vfLeftFlow :=: vfRightFlow;
end
```

图 15　阀门的 AltaRica 3.0 模型

Fig. 15　AltaRica 3.0 model of valve

最后，对蓄水罐进行建模。其属性有 *vsIsEmpty*(蓄水罐状态)、*vfOutFlow*(输出流)。当事件 *evGetEmpty*(罐变空)

发生时，变量 *vsIsEmpty* 的值由 false 变为 true。蓄水罐对应的 AltaRica 3.0 模型如图 16 所示。

```
class Tank
  Boolean vsIsEmpty (init = false);
  Boolean vfOutFlow (reset = true);
  event evGetEmpty;
  transition
    evGetEmpty: not vsIsEmpty -> vsIsEmpty := true;
  assertion
    vfOutFlow := not vsIsEmpty;
end
```

图 16 蓄水罐的 AltaRica 3.0 模型

Fig. 16 AltaRica 3.0 model of tank

该冷却系统由一个蓄水罐、两条冷却剂输送线路(各包含两个阀门、一个水泵)，以及一个反应堆组成，其主块对应的 AltaRica 3.0 模型已在第 2.2 节进行展示。由 4 个 class(RepairableComponent, Pump, Valve, Tank)和 1 个 block(CoolingSystem)所组成的模型便是整个冷却系统的 AltaRica 3.0 模型。

完成冷却系统建模后，调用类的平展化方法进行处理，其结果如图 17 所示。冷却系统的实验结果表明，类的平展化方法成功实现了对该系统共 4 个类(包括 2 个继承类 Pump 和 Valve)的平展化，符合实验预期。

Class_CompleteInfo.alt

```
1 class RepairableComponent
2   Boolean vsWorking(init=true);
3   parameter Real pLambda = 1.0e-5;
4   parameter Real pMu = 1.0e-2;
5   event evFailure(delay=exponential(pLambda));
6   event evRepair(delay=exponential(pMu));
7   transition
8     evFailure :  vsWorking ->vsWorking:=false;
9     evRepair :  not  vsWorking ->vsWorking:=true;
10 end
11 class Valve
12   Boolean vfLeftFlow,vfRightFlow(reset=false);
13   Boolean vsWorking(init=true);
14   parameter Real pLambda = 1.0e-5;
15   parameter Real pMu = 1.0e-2;
16   event evFailure(delay=exponential(pLambda));
17   event evRepair(delay=exponential(pMu));
18   transition
19     evFailure :  vsWorking ->vsWorking:=false;
20     evRepair :  not  vsWorking ->vsWorking:=true;
21   assertion
22     if vsWorking then vfLeftFlow:=:vfRightFlow;
23 end
24 class Tank
25   Boolean vsIsEmpty(init=false);
26   Boolean vfOutFlow(reset=true);
27   event evGetEmpty;
28   transition
29     evGetEmpty :  not  vsIsEmpty ->vsIsEmpty:=true;
30   assertion
31     vfOutFlow := not vsIsEmpty;
32 end
33 class Pump
34   Boolean vfInFlow,vfOutFlow(reset=false);
35   Boolean vsWorking(init=true);
36   parameter Real pLambda = 1.0e-5;
37   parameter Real pMu = 1.0e-2;
38   event evFailure(delay=exponential(pLambda));
39   event evRepair(delay=exponential(pMu));
40   transition
41     evFailure :  vsWorking ->vsWorking:=false;
42     evRepair :  not  vsWorking ->vsWorking:=true;
43   assertion
44     vfOutFlow := if(vsWorking) then vfInFlow else false;
45 end
```

图 17 冷却系统类的平展化结果

Fig. 17 Class flattening result of cooling system

以上实验对冷却系统、机轮刹车系统、灌溉系统、起落架系统共 4 个系统进行建模，并调用类的平展化方法进行平展化处理，对 4 个系统的特性、时间及内存消耗进行分析；此外，实例分析部分通过冷却系统对建模过程和结果进行了详细的说明。实验结果表明，类的平展化方法能够实现对中小规模系统的类的平展化，系统中扩展类的数量和类实例的数量是影响方法时间消耗的主要因素，类的数量和规模是影响方法内存消耗的主要因素。

5 相关工作

AltaRica 是一款致力于安全分析的高级建模语言。该语言的第一个版本 AltaRica(LaBRI)[15] 使得建立基本概念成为可能，但对于工业规模的模型来说，资源消耗过大。第二个版本 AltaRica Data-Flow[16] 基于模式自动机创建[17]，针对这一版本，已开发多个模型评估工具，如故障树[18] 编译器、马尔可夫链[19] 编译器、关键事件序列[20] 生成器等。AltaRica Data-Flow 目前已成为多个工业集成建模和仿真环境的核心，如 Cecilia OCAS(达索航空)、Simfia[21] 和 SafetyDesigner 等。AltaRica 3.0 是该语言的最新版本，它引入了一个基础数学模型 GTS，它是用来对 AltaRica 3.0 进行安全评估的形式化语义模型；此外，AltaRica 3.0 引入了面向对象的概念，即类的概念，从而允许对组件的重用。

目前国内对 AltaRica 3.0 的研究尚处于起步阶段，鲜有学者对 AltaRica 3.0 的建模方法和过程进行深入研究；而国外对 AltaRica 3.0 的研究起步较早，并积累了一定的经验，研发了相应的模型评估工具平台，如 OpenAltaRica[22] 等。国内外相关的研究如下：文献[4]详细介绍了 AltaRica 3.0 语言的语法结构，给出了 AltaRica 3.0 的 E-BNF(语法规范)，这是本文算法所基于的语法文件 AltaRica3.g4 的基础；文献[3]介绍了类的平展化算法的思想，但没有公开详细的实现方法和步骤；文献[12]提出了一种基于 ANTLR 的 AltaRica 3.0 模型平展化算法，给出了算法思想和伪代码，但在算法的平展化过程中需要反复遍历类的语法分析树以获取需要的信息，时空复杂度较高。

结束语 本文提出了一种 AltaRica 3.0 模型中的类的平展化方法，该方法基于自定义的语法文件，通过 ANTLR 工具生成相应的词法和语法解析器，对模型输入文件进行词法和语法分析并生成抽象语法树 AST，随后遍历类的语法树并将所有类的信息保存到自定义的数据结构中，然后通过操作该数据结构实现对所有类的平展化。经过实验验证，该方法能够实现对中小规模系统的类的平展化操作。

进一步的研究工作包括：1)通过更多、更大规模的系统来检验方法的有效性，并针对存在的问题进行改进，从而提高方法的通用性；2)进一步优化保存类的关键信息的数据结构 ClassInfo，提高内存使用率；3)将类的平展化方法应用到文献[10]提出的块的平展化方法中，进而实现 AltaRica 3.0 模型的平展化，即对系统层次结构的平展化；4)现有的平展化方法没有友好的界面交互，目前我们正在实现一个自主版本的建模仿真工具(Modeling and Simulation Tool)，之后会将 S2ML 到 GTS 的平展化方法集成到该工具中。

参 考 文 献

[1] MARCO B, ADOLFO V. Design and Safety Assessment of Critical Systems[M]. Auerbach Publications, 2010.

[2] LISAGOR O, KELLY T, NIU R. Model-based safety assess-

ment：Review of the discipline and its challenges[C]// The Proceedings of 2011 9th International Conference on Reliability，Maintainability and Safety. Guiyang，2011：625-632.

[3] PROSVIRNOVA T. AltaRica 3.0：a model-based approach for safety analyses[D]. Ecole Polytechnique，2014.

[4] BATTEUX M，PROSVIRNOVA T，RAUZY A. AltaRica 3.0 Language Specification[R]. AltaRica Association，2015.

[5] PROSVIRNOVA T，BATTEUX M，BRAMERET P，et al. The AltaRica 3.0 project for model-based safety assessment[J]. IFAC Proceedings Volumes，2013，46(22)：127-132.

[6] PROSVIRNOVA T，RAUZY A. Automated generation of minimal cut sets from AltaRica 3.0 models[J]. IJCCBS，2015，6(1)：50-80.

[7] RAUZY A B. Guarded transition systems：a new states/events formalism for reliability studies[J]. Proceedings of the Institution of Mechanical Engineers，Part O：Journal of Risk and Reliability，2008，222(4)：495-505.

[8] SHANE S，WOJTEK K. Model Transformation：The Heart and Soul of Model-Driven Software Development [J]. Software IEEE，2003，20(5)：42-45.

[9] CZARNECKI K，HELSEN S. Classification of Model Transformation Approaches[C]// Workshop on Generative Techniques in the Context of Model-driven Architecture. 2003.

[10] CHEN S，HU J，WANG L S. A Flattening Algorithm for AltaRica 3.0 Model Based on ANTLR[J]. Journal of Chinese Computer Systems，2020，41(7)：1476-1487.

[11] TERENCE P. The Definitive ANTLR 4 Reference [M]. The Pragmatic Bookshelf，2013.

[12] PROSVIRNOVA T，RAUZY A. AltaRica 3.0 project：compile Guarded Transition Systems into Fault Trees[C]// European Safety and Reliability Conference. ESREL，2013.

[13] HU J，CHEN S，WANG M M. Research on Transformation and Vertification Method of AltaRica 3.0 Model to Promela Model [J]. Computer Engineering and Science，2017，39(4)：708-716.

[14] FRÉDÉRIC B，VIRGINIE W. The Landing Gear System Case Study[C]// International Conference on Abstract State Machines. Springer International Publishing，2014.

[15] POINT GÔRAUZY A. AltaRica：Constraint automata as a description language [J]. European Journal of Automatisation，1999，33(8/9)：1033-1052.

[16] BOITEAU M，DUTUIT Y，RAUZY A，et al. The AltaRica data-flow language in use：modeling of production availability of a multi-state system[J]. Reliability Engineering & System Safety，2006，91(7)：747-755.

[17] RAUZY A. Mode automata and their compilation into fault trees [J]. Reliability Engineering & System Safety. 2002，78(1)：1-12.

[18] ERICSON C A. Fault tree analysis[C]// System Safety Conference. Florida，1999.

[19] STEWART W J. Introduction to the Numerical Solution of Markov Chains[M]// Introduction to the Numerical Solution of Markov Chains. DBLP，1994.

[20] GRIFFAULT A，POINT G，KUNTZ F，et al. Symbolic computation of minimal cuts for AltaRica models[R]. LaBRI，2011.

[21] ZHANG F K，HE Y F，GU Q F. Research on Security Analysis Method of HUD System Based on Model Drive[J]. Avionics Technology，2014(3)：52-56.

[22] SYSTEM X. The OpenAltaRica Platform-Getting Started[EB/OL]. (2017-09-04)[2020-05-20]. https://www.openaltarica.fr/docs/The%20OpenAltaRica%20Platform%20-%20Getting%20Started.pdf.

QI Jian，born in 1997，postgraduate，is a member of China Computer Federation. His main research interests include software verification and system security analysis.

HU Jun，born in 1973，Ph.D，associate professor，master supervisor，is a member of China Computer Federation. His main research interests include model-driven system security analysis，software verification and embedded system design.