带有并发行为的 UML 状态机图的形式语义

陈华豪[1,2],蒋建民[1,2],谢嘉成[1,2],陈卓然[1,2],唐国富[1,2]
(1. 成都信息工程大学 软件工程学院,四川 成都 610200;
2. 软件自动生成与智能服务四川省重点实验室,四川 成都 610200)

摘　要:在软件开发过程中,UML(统一建模语言)状态机图是目前最流行的建模形式之一,它属于半形式化模型,无法用形式化的方法进行推理。为了能对 UML 状态机图进行推理,现有工作采用 Petri 网、时序逻辑语言 XYZ/E、动态描述逻辑、Z(Object-Z)语言、CHAM 化学抽象机等作为状态机图的形式语义,但这些语义都是行为语义,并没有从结构上直接形成体现真并发的形式语义。该文提出一种新的模型——统一结构模型作为带有并发行为的 UML 状态机图的形式语义,该模型不会增加或减少状态机图的任何信息。基于统一结构模型首先定义了状态机图的格局(全局状态),用于表现状态机图的执行过程,并且给出了 UML 状态机图的格局的转换规则,说明格局如何在状态机图中执行,在此基础上给出了状态机图的可达性算法,然后还对状态机图的死锁等性质进行了介绍,最后开发出一个原型工具,实现了状态机图的可达性分析,并用实例说明了该方法的应用。

关键词:统一建模语言;状态机图;形式化模型;并发行为;可达性;死锁
中图分类号:TP3-0;TP301.2　　**文献标识码**:A　　**文章编号**:1673-629X(2024)05-0087-08
doi:10.20165/j.cnki.ISSN1673-629X.2024.0045

Formal Semantics of UML State Machine Diagrams with Concurrent Behaviors

CHEN Hua-hao[1,2],JIANG Jian-min[1,2],XIE Jia-cheng[1,2],CHEN Zhuo-ran[1,2],TANG Guo-fu[1,2]
(1. School of Software Engineering,Chengdu University of Information Technology,Chengdu 610200,China;
2. Sichuan Key Laboratory of Software Automatic Generation and Intelligent Service,Chengdu 610200,China)

Abstract:In software development,UML (Unified Modelling Language) state machine diagrams are one of the most popular forms of modelling,which are semi-formal models and cannot be reasoned about in a formal way. In order to be able to reason about UML state machine diagrams,existing work uses Petri nets,temporal logic language XYZ/E,dynamic description logic,Z (Object-Z) language,CHAM chemical abstraction machine,etc. as the formal semantics of the state machine diagrams,but these semantics are behavioural semantics,and there is no formal semantics embodying the true concurrency directly from the structure. We propose a new model,the Unified Structural Model,as the formal semantics of UML state machine diagrams with concurrency behaviours,which does not add or subtract any information from the state machine diagrams. Based on the Unified Structural Model,firstly,the configuration (global state) of a state machine diagram is defined,which is used to represent the execution process of a state machine diagram,and the transformation rules of the configuration of a UML state machine diagram are given to illustrate how the configuration is executed in a state machine diagram,based on which the accessibility algorithms of state machine diagrams are given. Then properties such as deadlocks of state machine diagrams are also introduced. Finally,a prototype tool is developed that implements the accessibility analysis of state machine diagrams,and the application of the methodology is illustrated with examples.

Key words:unified modeling language (UML);state machine diagram;formal models;concurrent behavior;accessibility;deadlock

0　引　言

统一建模语言(UML)是国际对象管理组织(OMG)建立的一种面向对象分析与设计的标准建模语言,它以各种简便和直观的图进行建模,并且可以作为软件开发整个过程的建模语言,现已成为软件开发过程中不可或缺的工具。实践已经证明,在使用 UML

收稿日期:2023-08-12　　**修回日期**:2023-12-13
基金项目:科技部重点研发计划(2022YFB3305104);国家自然科学基金(61772004);成都信息工程大学人才科研基金(KYTZ202009)
作者简介:陈华豪(1997-),男,硕士研究生,CCF 会员(P2693G),研究方向为软件开发方法与形式化方法;通讯作者:蒋建民(1972-),男,教授,博士,硕导,CCF 会员(92854M),研究方向为软件开发方法与形式化方法。

进行建模时,清晰严格的形式语义有利于软件开发工程师对软件系统进行严格的建模,从而保证软件开发过程的正确性。UML 状态机图(StateMachines)是统一建模语言的重要组成部分之一[1],在软件开发过程中被广泛使用,它可以用于从需求分析到软件开发完成再到后续测试的各个阶段。

为了对 UML 状态机图模型进行形式化的推理,可以采用 Petri 网、时序逻辑语言 XYZ/E、动态描述逻辑、Z(Object-Z)语言、CHAM 化学抽象机、TM 等模型作为状态机图的形式语义。已有一些研究人员对带有并发行为的状态机图给出了形式语义,但这些语义都是行为语义,它们将涉及并发行为的状态根据状态前后变迁的具体条件分别处理,没有从结构上直接形成体现真并发的形式语义。

该文提出一种新的模型——统一结构模型作为带有并发行为的 UML 状态机图的形式语义。统一结构不会增加或减少状态机图的任何信息,它不但可以从结构上描述系统的并发行为,而且可以统一描述 UML 状态机图的各种伪状态。基于统一结构模型,给出了状态机图的格局形式化定义,并且在此基础上给出了格局之间的转换规则用于刻画状态机图的执行过程。该文研究了带有并发行为的 UML 状态图的性质,比如死锁性质。也开发了分析状态机图的原型工具,用实例演示了状态机图的格局以及格局之间的转换,实现了状态机图的可达性分析。

1 相关工作

一些研究者对 UML 状态机图给出了自己的形式语义。在国内,薛丰等[2]用 UML 状态机图对铁路车站计算机联锁软件进行建模,并采用 Petri 网进行形式化定义与验证,从而保证软件的正确性和可靠性。邹崇理等[3]高度评价了唐稚松院士开发的时序逻辑语言 XYZ/E,重点陈述了其对传统时态逻辑和通常动态逻辑的创新之处以及时序逻辑语言 XYZ/E 比量化动态逻辑 QDL 具有更加简明直观的形式化语义。陈振庆等[4]提出基于动态描述逻辑 UML 状态机图形式化方法,在此基础上定义了状态可达性与动作包含关系,并证明了这些动作包含关系的正确性。于晓玲等[5]论述了用 Petri 网、时序逻辑语言 XYZ/E、动态描述逻辑描述状态机图的优缺点以及应用场景。Object-Z 是在 Z 语言语法和语义基础上对面向对象的扩展,但由于其缺乏完整的动态描述能力,马莉等[6]提出 DTL-Real-TimeObject-Z 规格语言,可有效描述操作的时态驱动、事件补偿、操作补偿,并通过责任授权模型的描述示例说明了该语言具有良好的形式化表达能力。崔光霁[7]提出了一种使用 CHAM 化学抽象机形式化语言对状态机图进行形式化的方法,与传统化学学科思想结合,将状态机图分为简单状态机图、分层状态机图和并发状态机图,并分别论述了其形式化描述的方法。曾一等[8]则以相反的角度,从形式化代码中提取 UML 状态机图,定义了一系列提取规则,并且验证了这些规则的正确性。杨志伟等[9]以一个五元组作为状态机图的形式化模型,然后分别论述了在状态机图中的几种覆盖准则,并提出新的测试准则来有效模拟用户的实际操作。李耀[10]等以有限状态机理论为基础,结合 Z 语言给出风险时间状态机的定义以及风险时间状态机的格局的转换机制与规则,为铁路信号系统软件测试形式化建模提供理论基础。该文所提出的格局的转换规则,是基于统一结构模型且适用于更一般的 UML 状态机图。

在国外,Liu 等[11]为状态机图全部特性提供了一个形式化语义,重点考虑了状态机图的非确定性以及 UML 状态机图之间的通信问题,开发了一个工具用于状态机图的各种属性进行模型检查,可有效发现不同状态机图之间通信的错误。Al-Fedaghi[12]提出一种新的建模方式——TM 模型作为状态机图的形式语义,建立状态和状态机概念的精确定义,给状态机图提供丰富的描述。Kherbouche[13]提出从 UML 状态机图的基本组件到正式工作流语言 YAWL 的转换。这种转换通过对 YAWL 的映射简化了状态机图的语义,使状态机图的语义更加清晰以及使状态机模型的验证和分析更加容易。Kochaleema 等[14]讨论了如何利用一种简单合理的形式化方法来增强 UML 的建模能力,建模者可直接从 UML 的状态机图来设计转换矩阵,再将转换矩阵转换为 LTS(标签转换系统),最后用 CTL(线性时态逻辑)或 LTL(计算树逻辑)模型检查算法完成形式化验证,从而简化建模的形式化验证过程。对于状态机图的细化,Syriani 等[15]首先介绍了状态机图中的一些概念和性质,然后提出了状态机图的一些新的细化规则,并且从形式上证明了这些规则的合理性,最后在一个建模示例中验证了这些细化规则,在保留原始结构和可达性的基础上,以保证广泛的开发自由,这些细化关系的应用有助于状态机图模型的重复使用,并且按照逐步细化的开发方法可以更好地设计状态机图模型。由于状态机图复杂的层次结构,要准确、快速地识别每张图表中是否符合要求的部分较为困难,Takuma Kimura 等[16]提出一种方法,利用模型检查技术半自动化地识别和测量状态机图中是否存在未满足要求的部分,该方法可以让教育工作者知道哪些图不符合要求,以及不符合哪些要求,并成功验证了该方法的有效性。

该文采用统一结构模型作为形式化模型定义状态

机图的形式语义,该模型语义简洁清晰,对于 UML 状态机图多层次的结构、各种类型的元素都可以描述。

2　状态机图的介绍

状态机图用来描述软件系统在各个时刻所处的状态信息,主要用来描述一个类、接口等对象在其生命周期中的各种状态以及状态之间的转换关系和转换顺序。

状态分为简单状态、复合状态和伪状态。简单状态是状态机图中最基本的状态。复合状态封装了子状态机图,该子状态机图分为 AND 状态和 OR 状态。AND 状态表示在子状态机图中可以并行执行,换句话说,子状态机图的多条线路可以被同时激活,OR 状态表示当子状态机图有多条线路的时候,只能执行其中一个。伪状态表示该状态不是真正意义上的状态,伪状态中不会有执行任何动作,除了简单状态和复合状态之外的所有状态都是伪状态。转换表示从一个状态到另一个状态之间的过程,一个转换可以表示为“触发事件[监护条件]/动作”,该文不考虑外部事件的触发,只考虑内部动作的转换。

图 1 是一个简单的状态机图。其中,A,D,E,G 和 H 是简单状态。Init′,B′,C′,Init1′, Fin1′,I′ 和 Fin′ 是伪状态。a,b,c,d,e,f,g, h,i 和 j 是引起状态转换的动作。F 是复合状态, F 里面的状态以及相关转换组成 F 的子状态机图。Fin1′ 和 Fin′ 是结束状态。对于 Init′ 到 A,Init1′ 到 G,H 到 Fin1′ 可以看作是特殊的无动作的状态转换。

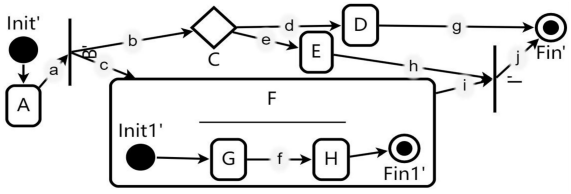

图 1　一个简单的状态机图

状态机图描述了各状态之间的相互转换关系和顺序。把每个动作都看成是互不相等的,动作的执行顺序可以表现为状态的转换顺序。

软件开发时,软件系统状态之间直观的事件转换顺序有利于程序开发人员把握各个事件的执行顺序。

状态转换时的动作、监护条件和所必须触发的事件都可以在状态机图中进行精确限制,程序员在进行软件开发时可以参考状态机图,从而在编写程序时避免软件系统进入一些由于非法事件引起的状态。

3　形式化模型

3.1　统一结构

定义 1:一个统一结构模型是一个元组 US = 〈ME, $\prec, \overset{1}{\omega}, \cdots, \overset{n}{\omega}, \lambda_m, \lambda_d, \overset{1}{\tau}, \cdots, \overset{m}{\tau}$〉,其中:

(1) ME,模型元素的有限集合;

(2) $\prec \subseteq$ ME × ME,如非反射性的部分命令,一种包含关系;

(3) $\forall i \in \{1,2,\cdots,n\}, \overset{i}{\omega} \subseteq$ ME × ME,依赖关系;

(4) $\lambda_m \subseteq$ ME × ME,模型元素的约束条件;

(5) $\lambda_d \subseteq$ ME × ($\prec \cup \overset{1}{\omega} \cup \cdots \cup \overset{n}{\omega}$),依赖性的约束条件;

(6) $\forall j \in \{1,2,\cdots,m\}, \overset{1}{\tau} \subseteq$ ME,模型元素的类型集合,如: $\forall e \in$ ME, $\exists \tau \in \{\overset{1}{\tau}, \cdots, \overset{n}{\tau}\}: e \in \tau$。

对于所有的 $x, y \in$ ME, $x \overset{i}{\omega} y (i \in \{1,2,\cdots,n\})$,称为一个依赖,解释为 x 依赖于 y(注意, i 表示依赖关系的类型)。$x \prec y$ 表示 x 包含在 y 中。如果 $x \prec z$, $y \prec z$,它们被简单地用 $x, y \prec z$ 表示,表示 x 和 y 都包含在 z 中。对于所有 $w, v \in$ ME,符号 $v \not\prec w$ 表示 w 不包含 v。元组 $\overset{1}{\tau}, \cdots, \overset{m}{\tau}$ 是模型元素的分组结构,用来对模型元素进行分类。

对于任何 $(x,y) \in \prec \cup \overset{1}{\omega} \cup \cdots \cup \overset{n}{\omega}$, x, y 分别称为依赖关系 (x,y) 的前元素和后元素。

对于任何 $y \in$ ME 的前置 ′y 和后置 y′ 分别定义为:

(1) $'y = \{x \in \text{ME} \mid (x,y \in (x,y) \in \prec \cup \overset{1}{\omega} \cup \cdots \cup \overset{n}{\omega}\}$;

(2) $y' = \{x \in \text{ME} \mid (y,x \in (y,x) \in \prec \cup \overset{1}{\omega} \cup \cdots \cup \overset{n}{\omega}\}$。

可以采用统一结构模型作为状态机图形式化模型,下面进行举例说明,图 1 可以表示为统一结构 SM = 〈ME, $\prec, \omega, \lambda_m, \lambda_d, \overset{\text{Action}}{\tau}, \overset{\text{IState}}{\tau}, \overset{\text{BState}}{\tau}, \overset{\text{FoState}}{\tau}, \overset{\text{CState}}{\tau}, \overset{\text{JState}}{\tau}, \overset{\text{FState}}{\tau}$〉,其中:

- $\prec$ = {(Init1′,F), (G,F), (f,F), (H,F), (Fin1′,F)}
- ω = {(A,Init1′), (B′,A), (C′,B′), (D,C′), (E,C′), (F,B′), (G,Init1′), (H,G), (Fin1′,H), (I′,E), (I′,F), (Fin′,D), (Fin′,I′)}
- $\lambda_m = \varnothing$
- λ_d = {(0, (A,Init′)), (a, (B′,A)), (b, (C′,B′)), (c, (F,B′)), (d, (D,C′)), (e, (E,C′)), (0, (G,Init1′)), (f, (H,G)), (0, (Fin1′,H)), (g, (Fin′,D)), (h, (I′,E)), (i, (I′,F)), (j, (Fin′,I′))}
- $\overset{\text{Action}}{\tau}$ = {a,b,c,d,e,f,g,h,i,j}

- $^{\text{IState}}\tau = \{\text{Init}', \text{Init1}'\}$
- $^{\text{BState}}\tau = \{A, D, E, F, G, H\}$
- $^{\text{FoState}}\tau = \{B'\}$
- $^{\text{CState}}\tau = \{C'\}$
- $^{\text{JState}}\tau = \{I'\}$
- $^{\text{FState}}\tau = \{\text{Fin1}', \text{Fin}'\}$

在状态机图中，$\prec$ 表示复合状态包含其子状态机图中的某个状态或某个动作，例如：复合状态 F 包含状态 Init1′，前者是包含的状态，后者是被包含的状态。ω 表示各状态之间的依赖关系，例如：状态 A 依赖于状态 Init′，前面是依赖的状态，后面是被依赖的状态。λ_d 表示状态之间的约束条件，设 $(a,(B,A)) \in \lambda_d$，a 表示一个动作，A，B 表示状态，表示从状态 A 执行动作 a 到达状态 B。例如：从状态 Init′ 执行空动作 0 到达状态 A。$^{\text{Action}}\tau$ 表示所有带动作的转换。$^{\text{IState}}\tau$ 表示所有初始状态。$^{\text{BState}}\tau$ 表示所有基本状态，在文中，简单状态和复合状态都是基本状态。$^{\text{CState}}\tau$ 表示选择状态。$^{\text{FoState}}\tau$ 表示分叉状态。$^{\text{JState}}\tau$ 表示汇合状态。$^{\text{FState}}\tau$ 表示结束状态。初始状态、选择状态、分叉状态、汇合状态、结束状态都是伪状态。

很明显，统一结构从结构上能很好地描述状态机图，并且没有丢失状态机图的任何信息。这就是引入统一结构的原因。实际上，统一结构可以描述任何一种 UML 图，这是笔者团队近期的工作。

注意：大写字母表示状态，小写字母表示动作。为方便说明，0 表示一个空动作，即不执行任何动作，用来表示初始状态到其后置状态执行的动作，以及结束状态的前置状态到该结束状态执行的动作。

3.2 格 局

格局表示状态机图的执行过程的全局状态，状态机图的执行是一个格局转换到另一个格局。

定义 2：设 SM $= \langle \text{ME}, \prec, \omega, \lambda_m, \lambda_d, {}^{\text{Action}}\tau, {}^{\text{IState}}\tau, {}^{\text{BState}}\tau, {}^{\text{FoState}}\tau, {}^{\text{CState}}\tau, {}^{\text{JState}}\tau, {}^{\text{FState}}\tau \rangle$ 是一个状态机图。SM 的一个格局是一个三元组 $\Gamma = \langle \Sigma, \Delta, \Lambda \rangle$，其中：

(1) $\Sigma \subseteq {}^{\text{BState}}\tau \cup {}^{\text{IState}}\tau \cup {}^{\text{FState}}\tau \cup {}^{\text{CState}}\tau \cup {}^{\text{FoState}}\tau \cup {}^{\text{JState}}\tau$ 是当前状态集；

(2) $\Delta \subseteq {}^{\text{Action}}\tau$ 是即将执行动作集；

(3) $\Lambda \subseteq {}^{\text{Action}}\tau$ 是已执行动作集。

公式的初始格局为 $\Gamma_0 = \langle \Sigma_0, \Delta_0, \Lambda_0 \rangle$，$\Sigma_0 = \{ {}^{\text{IState}}\tau \setminus \{S_0 \in {}^{\text{IState}}\tau \mid (S_0, S) \in \prec, S \in {}^{\text{BState}}\tau\}$，$\Delta_0 = 0$，$\Lambda_0 = \varnothing$。如图 1 所示，$\Gamma_0 = \langle \{\text{init}'\}, \{0\}, \varnothing \rangle$ 表示该状态机图的初始格局。当前状态集中包含结束状态的格局为结束格局。

这里定义了格局，状态机图的执行就是格局之间的转换，下面给出格局的转换规则。

定义 3：设 SM $= \langle \text{ME}, \prec, \omega, \lambda_m, \lambda_d, {}^{\text{Action}}\tau, {}^{\text{IState}}\tau, {}^{\text{BState}}\tau, {}^{\text{FoState}}\tau, {}^{\text{CState}}\tau, {}^{\text{JState}}\tau, {}^{\text{FState}}\tau \rangle$ 是一个状态机图。$\Gamma_1 = \langle \Sigma_1, \Delta_1, \Lambda_1 \rangle$ 和 $\Gamma_2 = \langle \Sigma_2, \Delta_2, \Lambda_2 \rangle$ 是 SM 的两个格局。存在如下三种情况：

(1) 如果 $\exists S \in \Sigma_1, \exists S_1 \in {}^{\text{IState}}\tau, (S_1, S) \in \prec, \Sigma_2 = \Sigma_1 \cup \{S_1\}$，$\Delta_2 = \Delta_1 \cup \{x \in {}^{\text{Action}}\tau \mid S' \in \text{ME}, (x, (S', S_1) \in \lambda_d\}$，$\Lambda_2 = \Lambda_1$，则称 Γ_1 进入复合状态而得到格局 Γ_2，表示为 $\Gamma_1 \xrightarrow{0_+} \Gamma_2$。$0_+$ 表示进入复合状态的动作。

(2) 如果 $\exists S \in \text{ME}, \exists S_1 \in {}^{\text{FState}}\tau \cap \Sigma_1, (S_1, S) \in \prec, \Sigma_2 = (\Sigma_1 \setminus \{S_1\} \setminus \{S\}) \cup \{S \in \text{ME} \mid S' \in \text{ME}, (S', S) \in \omega\}$，$\Delta_2 = \Delta_1 \setminus \{x \in {}^{\text{Action}}\tau \mid S \in \text{ME}, S' \in \text{ME}, (x, (S', S) \in \lambda_d\} \cup \{y \in {}^{\text{Action}}\tau \mid S' \in \text{ME}, S'' \in \text{ME}, (y, (S'', S') \in \lambda_d\}$，$\Lambda_2 = (\Lambda_1 \setminus \{x \in {}^{\text{Action}}\tau \mid S \in \text{ME}, S' \in \text{ME}, (x, (S, S') \in \lambda_d\} \setminus \{y \in {}^{\text{Action}}\tau \mid S'' \in \text{ME}, (y, (S_1, S'') \in \lambda_d\}) \cup \{z \in {}^{\text{Action}}\tau \mid S_2 \in \text{ME}, (z, (S_2, S) \in \lambda_d\}$，则称 Γ_1 退出复合状态而得到格局 Γ_2，表示为 $\Gamma_1 \xrightarrow{0_-} \Gamma_2$。$0_-$ 表示退出复合状态的动作。

(3) 如果 $\exists a \in \Delta_1, \exists S_1 \in \Sigma_1, \nexists S \in \text{ME}$，使得 $(S, S_1) \in \prec$，并且以下条件之一成立：

(a) $S_1 \in {}^{\text{BState}}\tau, \exists S_2 \in \text{ME}, \exists (S_2, S_1) \in \omega, \exists (a, (S_2, S_1)) \in \lambda_d, \Sigma_2 = (\Sigma_1 \setminus \{S_1\}) \cup \{S_2\}$，$\Delta_2 = \Delta_1 \setminus \{a\} \cup \{x \in {}^{\text{Action}}\tau \mid S' \in \text{ME}, (x, (S', S_2)) \in \lambda_d\}$，$\Lambda_2 = \Lambda_1 \cup \{a\} \setminus \{y \in {}^{\text{Action}}\tau \mid S'' \in \text{ME}, (y, (S_1, S'')) \in \lambda_d\}$；

(b) $S_1 \in {}^{\text{CState}}\tau, \exists S_2 \in \text{ME}, \exists (S_2, S_1) \in \omega, \exists (a, (S_2, S_1)) \in \lambda_d, \Sigma_2 = (\Sigma_1 \setminus \{S_1\}) \cup \{S_2\}$，$\Delta_2 = (\Delta_1 \setminus \{x \in {}^{\text{Action}}\tau \mid S' \in \text{ME}, (x, (S', S_1)) \in \lambda_d\}) \cup \{y \in {}^{\text{Action}}\tau \mid (y, (S'', S_2)) \in \lambda_d, S'' \in \text{ME}\}$，$\Lambda_2 = \Lambda_1 \cup \{a\} \setminus \{z \in {}^{\text{Action}}\tau \mid S'' \in \text{ME}, (z, (S_1, S'')) \in \lambda_d\}$；

(c) $S_1 \in {}^{\text{JState}}\tau, \exists S_2 \in \text{ME}, \exists (S_2, S_1) \in \omega, (a, (S_2, S_1)) \in \lambda_d, \Sigma_2 = (\Sigma_1 \setminus \{S_1\}) \cup \{S_2\}$，$\Delta_2 = (\Delta_1 \setminus \{x \in {}^{\text{Action}}\tau \mid S \in \text{ME}, (x, (S, S_1)) \in \lambda_d\}) \cup \{y \in {}^{\text{Action}}\tau \mid S' \in \text{ME}, (y, (S', S_2)) \in \lambda_d\}$，$\Lambda_2 = \Lambda_1 \cup \{a\} \setminus \{z \in {}^{\text{Action}}\tau \mid S'' \in \text{ME}, (z, (S_1, S'')) \in \lambda_d\}$，则称 Γ_1 可以通过执行动作 a 演化成 Γ_2，表示为 $\Gamma_1 \xrightarrow{a} \Gamma_2$。

下面对规则进行简要概述，在复合状态中，先进行

预处理,在状态机图中存在两种情况。(1)表示当进入复合状态时,要求新的格局的当前状态集包含该复合状态的子状态中的初始状态,即将执行动作集包含该初始状态的后置动作,已执行动作集不变;(2)表示当退出复合状态时,要求新的格局的当前状态集包含该复合状态即将要进入的状态,即将执行动作集包含该复合状态的后置状态的后置动作,已执行动作集包含该复合状态的后置动作。

在处理所有的复合状态后,对于(3)条件(a)要求新的格局的当前状态集包含该简单状态即将要进入的状态,即将执行动作集包含该状态中即将要进入的后置状态的后置动作,已执行动作集包含该状态的后置动作。条件(b)要求新的格局的当前状态集包含该选择状态即将要进入的状态,即将执行动作集包含该状态的即将要进入的状态的后置动作。已执行动作集包含该状态的后继可执行动作。条件(c)要求新的格局的当前状态集包含该汇合状态的后置状态,即将执行动作集不包含该状态的所有前置动作并且包含该状态的后置状态的后置动作,已执行动作集包含该状态的后置动作。对于当前状态集包含汇合状态的格局,由动作 a 演化到下一个格局,则要求该格局已执行动作集必须包含该状态的所有前置动作集。条件 a,b,c 的动作都包含空动作 0。

定理 1:设 $SM=\langle ME,\prec,\omega,\lambda_m,\lambda_d,\overset{Action}{\tau},\overset{IState}{\tau},\overset{BState}{\tau},\overset{FoState}{\tau},\overset{CState}{\tau},\overset{JState}{\tau},\overset{FState}{\tau}\rangle$ 是一个状态机图,$\Gamma=\langle\Sigma,\Delta,\Lambda\rangle$ 是 SM 的格局。如果 $\overset{Action}{\tau},\overset{IState}{\tau},\overset{BState}{\tau},\overset{FoState}{\tau},\overset{CState}{\tau},\overset{JState}{\tau},\overset{FState}{\tau}$ 都是有限的,则计算 SM 的格局 Γ 是可判定的。

证明:$\overset{Action}{\tau},\overset{IState}{\tau},\overset{BState}{\tau},\overset{FoState}{\tau},\overset{CState}{\tau},\overset{JState}{\tau},\overset{FState}{\tau}$ 是有限的,因此格局 Γ 的当前状态集 Σ 是有限的,即 Σ 是可判定的。$\overset{Action}{\tau}$ 是有限的,因此 Γ 即将执行动作集 Δ 和已执行动作集 Λ 是有限的,即 Δ,Λ 是可判定的。所以 SM 的格局 Γ 是可判定的。

该定理表明存在一个算法,可以计算出一个状态机图的所有可达格局(见定义 4)。如下给出了状态机图的可达性算法。

算法 1:状态机图的计算格局(可达性)算法。

输入:一个状态机图 SM。

(1)求出初始格局,即将执行动作集置为 $\{0\}$,已执行动作集置为 $\varnothing$;

(2)循环遍历整个状态机图,当该状态是初始状态但不是某个复合状态的子状态时,把该状态加入初始格局的当前状态集;

(3)将初始格局赋值给下一个格局,开始循环;

(4)当进入复合状态时,该复合状态的子状态——初始状态加入格局的当前状态集中,该初始状态后置动作加入格局的即将执行动作集中,格局的已执行动作集不变;

(5)当退出复合状态时,该复合状态的后置状态加入格局的当前状态集中,并且从当前状态集中减去该复合状态以及该复合状态的子状态——结束状态,该复合状态的后置状态的后置动作加入格局的即将执行动作集中,并且减去该复合状态的后置动作,格局的已执行动作集减去该复合状态的前置动作以及该复合状态的子状态——结束状态的前置动作;

(6)当前状态为基本状态时,该基本状态的后置状态加入格局的当前状态集中,并且从当前状态集减去该基本状态,该基本状态的后置状态的后置动作加入格局的即将执行动作集中,并且减去该基本状态的后置动作,该基本状态的后置动作加入格局的已执行动作集中,并且减去该基本状态的前置动作;

(7)当前状态为选择状态时,该选择状态可达的后置状态加入格局的当前状态集中,并且从当前状态集中减去该选择状态,该选择状态可达的后置状态的后置动作加入格局的即将执行动作集中,并且减去该选择状态的所有后置动作,该选择状态可执行的后置动作加入格局的已执行动作集中,并且减去该选择状态的前置动作;

(8)当前状态为汇合状态时,该汇合状态的后置状态加入格局的当前状态集中,并且从当前状态集中减去该汇合状态,该汇合状态的后置状态的后置动作加入格局的即将执行动作集中,并且减去该汇合状态的后置动作,该汇合状态的后置动作加入格局的已执行动作集中,并且减去该汇合状态所有的前置动作;

(9)上一个格局赋值给下一个格局,并且继续遍历整个状态机图,返回第 4 步,直到格局的可执行动作集为空集,结束。

输出:SM 中所有可达格局的集合 Conf(SM)。

与其它并发模型一样,上述计算格局(可达性)算法的时间复杂度是指数级的。

根据格局的定义,表 1 为图 1 中的格局转换情况。由于 $\{C'\}\subseteq\overset{CState}{\tau}$,以 Γ_{10} 开始应该分成两种情况。

表 1　图 1 的格局转换情况

序号	当前状态集	即将执行动作集	已执行动作集	格局
$\Gamma_0=\langle\Sigma_0,\Delta_0,\Lambda_0\rangle$	$\Sigma_0=\{Init'\}$	$\Delta_0=\{0\}$	$\Lambda_0=\varnothing$	$\Gamma_0=\langle\{Init'\},\{0\},\varnothing\rangle$
$\Gamma_1=\langle\Sigma_1,\Delta_1,\Lambda_1\rangle$	$\Sigma_1=\{A\}$	$\Delta_1=\{a\}$	$\Lambda_1=\{0\}$	$\Gamma_1=\langle\{A\},\{a\},\{0\}\rangle$
$\Gamma_2=\langle\Sigma_2,\Delta_2,\Lambda_2\rangle$	$\Sigma_2=\{B'\}$	$\Delta_2=\{b,c\}$	$\Lambda_2=\{a\}$	$\Gamma_2=\langle\{B'\},\{b,c\},\{a\}\rangle$
$\Gamma_3=\langle\Sigma_3,\Delta_3,\Lambda_3\rangle$	$\Sigma_3=\{F\}$	$\Delta_3=\{b,i\}$	$\Lambda_3=\{c\}$	$\Gamma_3=\langle\{F\},\{b,i\},\{c\}\rangle$
$\Gamma_4=\langle\Sigma_4,\Delta_4,\Lambda_4\rangle$	$\Sigma_4=\{F,Init1'\}$	$\Delta_4=\{b,i,0\}$	$\Lambda_4=\{c\}$	$\Gamma_4\langle\{F,Init1'\},\{b,i,0\},\{c\}\rangle$
$\Gamma_5=\langle\Sigma_5,\Delta_5,\Lambda_5\rangle$	$\Sigma_5=\{F,G\}$	$\Delta_5=\{b,i,f\}$	$\Lambda_5=\{c,0\}$	$\Gamma_5=\langle\{F,G\},\{b,i,f\},\{c,0\}\rangle$

续表 1

序号	当前状态集	即将执行动作集	已执行动作集	格局
$\Gamma_6=\langle\Sigma_6,\Delta_6,\Lambda_6\rangle$	$\Sigma_6=\{F,H\}$	$\Delta_6=\{b,i,0\}$	$\Lambda_6=\{c,f\}$	$\Gamma_6=\langle\{F,H\},\{b,i,0\},\{c,f\}\rangle$
$\Gamma_7=\langle\Sigma_7,\Delta_7,\Lambda_7\rangle$	$\Sigma_7=\{F,\mathrm{Fin1}'\}$	$\Delta_7=\{b,i\}$	$\Lambda_7=\{c,0\}$	$\Gamma_7=\langle\{F,\mathrm{Fin1}'\},\{b,i\},\{c,0\}\rangle$
$\Gamma_8=\langle\Sigma_8,\Delta_8,\Lambda_8\rangle$	$\Sigma_8=\{I'\}$	$\Delta_8=\{b,j\}$	$\Lambda_8=\{i\}$	$\Gamma_8=\langle\{I'\},\{b,0\},\{i\}\rangle$
$\Gamma_9=\langle\Sigma_9,\Delta_9,\Lambda_9\rangle$	$\Sigma_9=\{C'\}$	$\Delta_9=\{j,d,e\}$	$\Lambda_9=\{i,b\}$	$\Gamma_9=\langle\{C'\},\{j,d,e\},\{i,b\}\rangle$
$\Gamma_{10}=\langle\Sigma_{10},\Delta_{10},\Lambda_{10}\rangle$	$\Sigma_{10}=\{D\}$	$\Delta_{10}=\{j,g\}$	$\Lambda_{10}=\{i,d\}$	$\Gamma_{10}=\langle\{D\},\{j,g\},\{i,d\}\rangle$
$\Gamma_{11}=\langle\Sigma_{11},\Delta_{11},\Lambda_{11}\rangle$	$\Sigma_{11}=\{I',\mathrm{Fin}'\}$	$\Delta_{11}=\{j\}$	$\Lambda_{11}=\{i,g\}$	$\Gamma_{11}=\langle\{I',\mathrm{Fin}'\},\{j\},\{i,g\}\rangle$
$\Gamma_{10}=\langle\Sigma_{10},\Delta_{10},\Lambda_{10}\rangle$	$\Sigma_{10}=\{E\}$	$\Delta_{10}=\{j,h\}$	$\Lambda_{10}=\{i,e\}$	$\Gamma_{10}=\langle\{E\},\{j,h\},\{i,e\}\rangle$
$\Gamma_{11}=\langle\Sigma_{11},\Delta_{11},\Lambda_{11}\rangle$	$\Sigma_{11}=\{I'\}$	$\Delta_{11}=\{j\}$	$\Lambda_{11}=\{i,h\}$	$\Gamma_{11}=\langle\{I'\},\{j\},\{i,h\}\rangle$
$\Gamma_{12}=\langle\Sigma_{12},\Delta_{12},\Lambda_{12}\rangle$	$\Sigma_{12}=\{\mathrm{Fin}'\}$	$\Delta_{12}=\varnothing$	$\Lambda_{12}=\{j\}$	$\Gamma_{12}=\langle\{\mathrm{Fin}'\},\varnothing,\{j\}\rangle$

当状态机图的格局的当前状态集包含状态 B' 时，由于 $B'\in{}^{\mathrm{FoState}}\tau$，因此状态机图的执行在后续将会有无数种格局，在此只保留了其中一种情况。

4 状态机图的性质

定义 4：设 $\mathrm{SM}=\langle \mathrm{ME},\prec,\omega,\lambda_m,\lambda_d,{}^{\mathrm{Action}}\tau,{}^{\mathrm{IState}}\tau,{}^{\mathrm{BState}}\tau,{}^{\mathrm{FoState}}\tau,{}^{\mathrm{CState}}\tau,{}^{\mathrm{JState}}\tau,{}^{\mathrm{FState}}\tau\rangle$ 是一个状态机图。设 $\Gamma=\langle\Sigma,\Delta,\Lambda\rangle$ 是 SM 的格局，并设 $\Gamma_0=\langle\Sigma_0,\Delta_0,\Lambda_0\rangle$ 是 SM 的初始格局。如果 SM 中存在一个格局 Γ'，且存在一个格局序列 $\Gamma_1',\cdots,\Gamma_{n-1}'$，使得 $\Gamma_0'\xrightarrow{a_1'}\Gamma_1'\cdots\Gamma_{n-1}'\xrightarrow{a_n'}\Gamma'$，其中 $a_i'\in{}^{\mathrm{Action}}\tau\cup\{0_+\}\cup\{0_-\}\cup\{0\},i\in\{1,2,\cdots,n\}$，则称格局 Γ' 在 SM 中是可达的。如果在 SM 中存在两个格局 Γ' 和 Γ''，且存在格局序列 $\Gamma_1'',\cdots,\Gamma_{n-1}''$ 使得 $\Gamma_0''\xrightarrow{a_1''}\Gamma_1''\cdots\Gamma_{n-1}''\xrightarrow{a_n''}\Gamma_n''$，其中 $a_i''\in{}^{\mathrm{Action}}\tau\cup\{0_+\}\cup\{0_-\}\cup\{0\},i\in\{1,2,\cdots,n\}$，则称格局 Γ'' 可达至格局 Γ'，表示为 $\Gamma''\xrightarrow{*}\Gamma'$。$\mathrm{Conf}(\mathrm{SM})$ 表示 SM 中所有可达格局的集合。

定义 5：设 $\mathrm{SM}=\langle \mathrm{ME},\prec,\omega,\lambda_m,\lambda_d,{}^{\mathrm{Action}}\tau,{}^{\mathrm{IState}}\tau,{}^{\mathrm{BState}}\tau,{}^{\mathrm{FoState}}\tau,{}^{\mathrm{CState}}\tau,{}^{\mathrm{JState}}\tau,{}^{\mathrm{FState}}\tau\rangle$ 是一个状态机图。设 $\Gamma=\langle\Sigma,\Delta,\Lambda\rangle\in\mathrm{Conf}(\mathrm{SM})$。如果 $\Delta=\varnothing$，并且 $\Sigma\subseteq{}^{\mathrm{FState}}\tau$，则称 Γ 是终止的；如果 Γ 不是终止的并且不存在一个格局 Γ'，使得 $\Gamma'\xrightarrow{*}\Gamma$，则称 Γ 是死的。如果在 SM 中的任意格局 Γ 是终止的或者存在一个终止格局 Γ_f，使得 $\Gamma\xrightarrow{*}\Gamma_f$，则称 SM 是弱终止的。如果 $\exists\Gamma\in\mathrm{Conf}(\mathrm{SM})$ 使得 Γ 是死的，则称 SM 是死锁的。如果不存在 $\Gamma\in\mathrm{Conf}(\mathrm{SM})$ 使得 Γ 是死的，则称 SM 是无死锁的。

命题 1：设 $\mathrm{SM}=\langle \mathrm{ME},\prec,\omega,\lambda_m,\lambda_d,{}^{\mathrm{Action}}\tau,{}^{\mathrm{IState}}\tau,{}^{\mathrm{BState}}\tau,{}^{\mathrm{FoState}}\tau,{}^{\mathrm{CState}}\tau,{}^{\mathrm{JState}}\tau,{}^{\mathrm{FState}}\tau\rangle$ 是一个状态机图。

(1) 如果 SM 是弱终止，则称 SM 是无死锁的。

(2) 如果 $\Gamma=\langle\Sigma,\Delta,\Lambda\rangle\in\mathrm{Conf}(\mathrm{SM})$，并且 $\Sigma\not\subseteq{}^{\mathrm{FState}}\tau\wedge\Delta\neq\varnothing$，但 Δ 中的动作并未执行，设 $S\in\mathrm{ME}$，$S'\in\Sigma$，且 $(S',S)\notin\prec$，则称 S 是死的。

证明：由定义 4、定义 5 直接可得。

命题 1 表明：(1) 一个无死锁的状态机图比弱终止的状态机图是更一般的状态机图；(2) 如果结束格局的当前状态集中的状态不都是结束状态，并且该格局的即将执行动作集不为空，但除结束状态之外的状态的前置动作不都在已执行动作集中，则该状态是死的。该性质可以用于检测状态机图中的死锁。

命题 2：设 $\mathrm{SM}=\langle \mathrm{ME},\prec,\omega,\lambda_m,\lambda_d,{}^{\mathrm{Action}}\tau,{}^{\mathrm{IState}}\tau,{}^{\mathrm{BState}}\tau,{}^{\mathrm{FoState}}\tau,{}^{\mathrm{CState}}\tau,{}^{\mathrm{JState}}\tau,{}^{\mathrm{FState}}\tau\rangle$ 是一个状态机图。如果 ${}^{\mathrm{JState}}\tau=\varnothing$，则 SM 是无死锁的。

证明：由 ${}^{\mathrm{JState}}\tau=\varnothing$，表明状态机图中不存在汇合状态，就不存在多分支等待执行的情况，则一定不存在一个死的格局，即 SM 是无死锁的。

命题 2 表明汇合状态这类伪状态是导致死锁的根本原因。

5 工具及实例研究

开发的原型工具是在使用 JavaScript 的 GoJS 图形库作为基础可视化操作界面，配合 jQuery 进行开发。JavaScript 是一种解释性的脚本语言，可以基于对象进行开发，不依赖于具体的操作系统，仅需要浏览器的支持。GoJS 是用于构建交互图表和图形的 JavaScript 和 TypeScript 库。此工具拥有 UML 状态机图的基本绘图和错误检测功能，并且正在逐步完善。图 2 是工具界面。工具地址：http://124.220.63.75。以下用实例说明文中的方法。

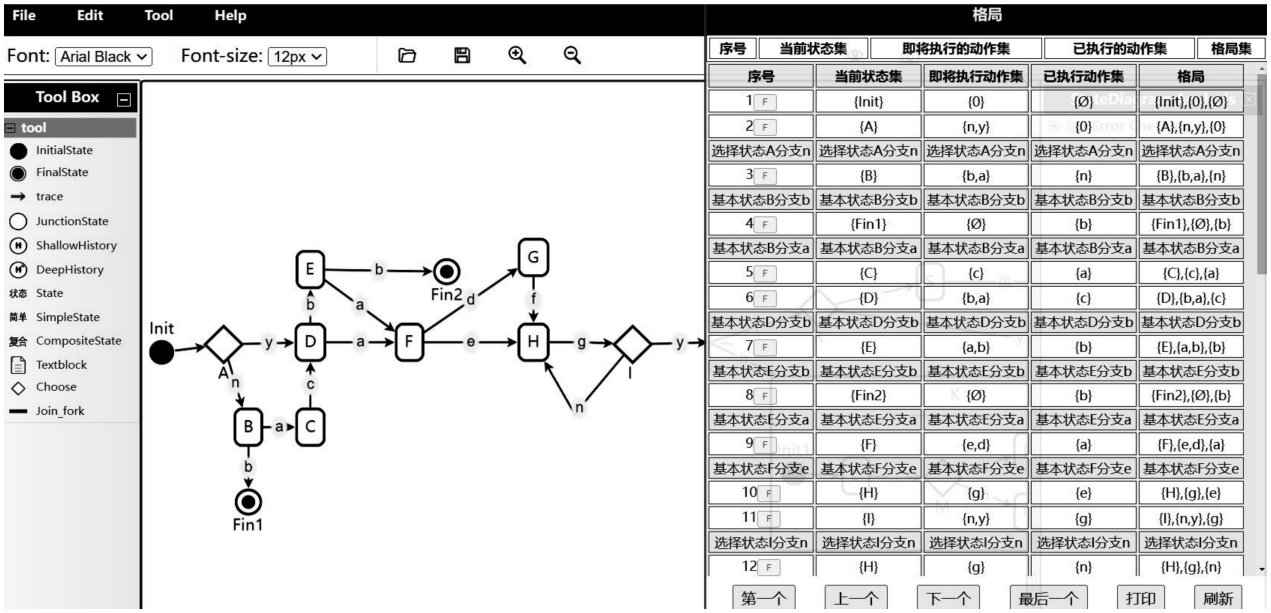

图 2　工具界面

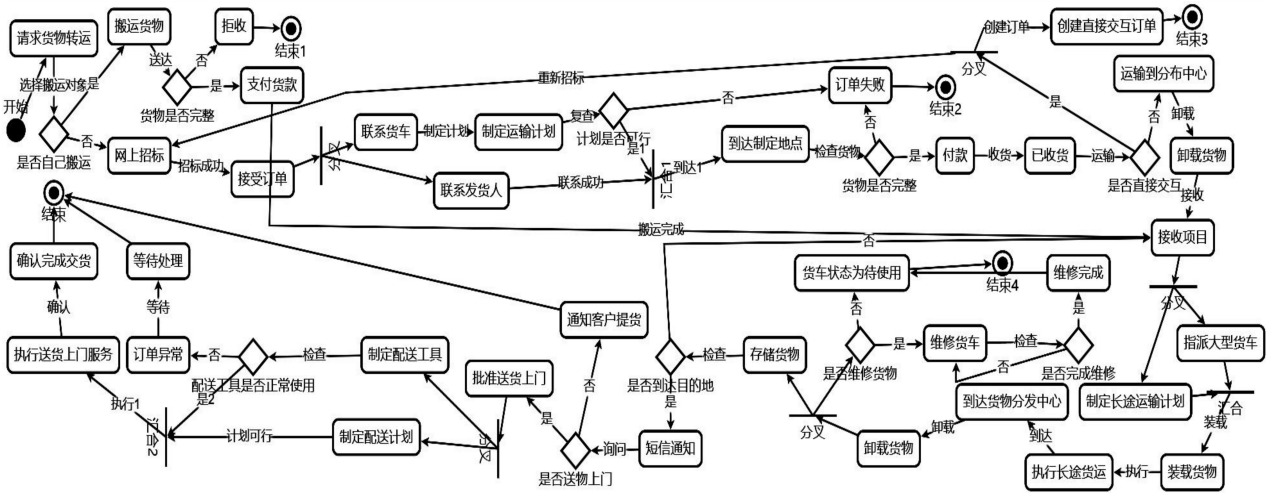

图 3　某公司的货物转运的状态机图

图 3 是某公司的货物转运的状态机图。客户要求该公司在全国范围内将货物从一个地方运至另一个地方。货物物流状态转换系统是一个复杂的系统,在此只考虑主要功能。客户下单之后可以选择物流公司是否派出卡车接收货物。在不选择的情况下,可以自己将其运往物流中转站。到达中转站后,两者都必须检查货物是否完整。一般情况下,货物被堆放在物流中转站。距离收货人地址较远就指派重型货车运输,否则就使用轻型货车运输,这样便于节省成本。已经到达距离收货地址最近的物流中转站后,客户可以自己选择是否需要送货上门,一般这段流程都需要单独付费。在物流转运状态系统中,由于重型货车的购买价格较为高昂,必须对车辆状态进行有效管理。一般的物流中转站配有汽车维修部门,当货物到达指定中转站后,需等待维修部门检查维修且出示货车的可正常使用的报告,该货车才可以再次投入运作。

为了方便说明,在表 2 中给出了图 3 所有的结束格局的当前状态集中的状态以及相关动作的字母表示。

表 2　状态和动作对应的字母表

状态	结束 1	汇合 1	结束 2	结束 3	结束 4	汇合 2	结束
字母	Fin1	Jo1	Fin2	Fin3	Fin4	Jo2	Fin
动作	是 1	联系成功	到达 1	是 2	计划可行	执行 1	
字母	a	b	c	d	e	f	

以下是图 3 的所有结束格局:

$\Gamma_0 = \langle \{Fin1\}, \varnothing, \{0\} \rangle$

$\Gamma_1 = \langle \{Jo1, Fin2\}, \{c\}, \{b\} \rangle$

$\Gamma_2 = \langle \{Fin3\}, \varnothing, \{0\} \rangle$

$\Gamma_3 = \langle \{Fin4\}, \varnothing, \{0\} \rangle$

$\Gamma_4 = \langle \{Jo2, Fin\}, \{f\}, \{e\} \rangle$

$\Gamma_5 = \langle \{Fin\}, \varnothing, \{0\} \rangle$

根据命题 1,如果该状态机图的结束格局为 Γ_0,

Γ_2,Γ_3,Γ_5,则不会有状态引起死锁。如果该状态机图的结束格局为 Γ_1,Γ_4,对于格局 Γ_1,当前状态集中除了 Fin2,还有 Jo1,即当前状态集中的状态不都是结束状态;可执行动作集中包含 c,即不为空集; Jo1 前置动作是 a 和 b,但已执行动作集中只包含动作 b,即 Jo1 的所有前置动作并不都在格局 Γ_1 的已执行动作集中,因此 Jo1 是死的。同理,对于格局 Γ_4, Jo2 也是死的。

6 结束语

该文采用统一结构模型作为状态机图的形式语义,可以从结构上描述状态机图。在此基础上,定义了状态机图的格局,并且根据格局的定义,给出了格局之间的相互转换规则,以及给出了状态机图的可达性算法,然后研究了状态机图的一些性质。最后开发了状态机图的建模和分析工具,用实例说明了带有并发行为的状态机图在软件开发过程中的应用。在接下来的工作中将继续完善该工具,使它成为一个成熟的工具。

参考文献:

[1] Object Management Group. OMG unified modeling Language™(OMG UML) version 2.5.1[M/OL]. https://www.omg.org/spec/UML/2.5.1/PDF.

[2] 薛 丰,杨 扬,谢 林. 基于 UML 建模的计算机联锁进路模块 Petri 网验证[J]. 铁路计算机应用,2017,26(4):10-14 .

[3] 邹崇理. 时序逻辑程序语言 XYZ/E 的创新性[J]. 重庆理工大学学报:社会科学版,2018,32(9):9-15.

[4] 陈振庆,罗兰花. 基于动态描述逻辑的 UML 状态图形式化方法[J]. 计算机工程,2011,37(13):55-57.

[5] 于晓玲,杨海波. UML 状态图的形式化方法的分析和比较[J]. 计算机与数字工程,2014,42(8):1488-1492.

[6] 马 莉,钟 勇,霍颖瑜. DTL-Real-Time Object-Z 形式化规格说明语言及其责任授权模型描述[J]. 计算机科学,2014,41(4):184-189.

[7] 崔光霁. UML 状态图的形式化研究[D]. 太原:太原理工大学,2011.

[8] 曾 一,周 欣,周 吉. 基于形式化规格说明的 UML 状态图提取[J]. 计算机应用研究,2011,28(5):1767-1769.

[9] 杨志伟,吴 兵. 基于 UML 状态图的软件测试充分性准则研究[J]. 计算机技术与发展,2013,23(8):43-46.

[10] 李 耀,张晓霞,郭 进,等. 铁路信号系统软件测试建模方法[J]. 西南交通大学学报,2022,57(2):392-400.

[11] LIU S,LIU Y,ANDRÉ E,et al. A formal semantics for complete UML state machines with communications[C]//International conference on integrated formal methods. Berlin:Springer,2013:331-346.

[12] AL-FEDAGHI S. Modeling the semantics of states and state machines[J]. arXiv:2007.07138,2020.

[13] KHERBOUCHE M, BOUAFIA K, MOLNÁR B. Transformation of UML state machine to YAWL[C]//2019 ninth international conference on intelligent computing and information systems (ICICIS). Cairo:IEEE,2019:215-220.

[14] KOCHALEEMA K H,MANSOOR S,SANTHOSHKUMAR G. Automatic formal model generation from UML diagrams-an implementation experience[C]//2022 IEEE Delhi section conference (DELCON). New Delhi:IEEE,2022:1-6.

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

[16] KIMURA T. A method to semi-automatically identify and measure unmet requirements in learner-created state machine diagrams[C]//2023 IEEE 35th international conference on software engineering education and training (CSEE&T). Tokyo:IEEE,2023.