第29卷 第4期 计 算 机 学 报 Vol. 29 No. 4
2006年4月 CHINESE JOURNAL OF COMPUTERS Apr. 2006

基于场景规约的构件式系统设计分析与验证

胡 军 于笑丰 张 岩 王林章 李宣东 郑国梁

(计算机软件新技术国家重点实验室 南京 210093)

(南京大学计算机科学与技术系 南京 210093)

摘 要 使用接口自动机及接口自动机网络来描述构件式系统的行为设计模型,使用UML顺序图表示基于场景的需求规约,对系统设计阶段的构件交互行为的动态兼容性进行形式化分析和检验.通过对接口自动机网络状态空间的分析,给出了一系列算法以检验系统行为的存在一致性以及几种不同形式的强制一致性性质,包括前向强制一致性、逆向强制一致性以及双向强制一致性等.

关键词 构件式系统设计;接口自动机;模型检验;顺序图;统一建模语言(UML)

中图法分类号 TP311

Checking Component-Based Designs for Scenario-Based Specifications

HU Jun YU Xiao-Feng ZHANG Yan WANG Lin-Zhang LI Xuan-Dong ZHENG Guo-Liang

(*State Key Laboratory for Novel Software Technology, Nanjing* 210093)

(*Department of Computer Science and Technology, Nanjing University, Nanjing* 210093)

Abstract Component-based system design is becoming more and more popular in software engineering. Checking the important behavioral properties formally in the design phase is an effective way to improve the system reliability. In this paper, the authors consider the problem of checking component-based system designs for scenario-based specifications. Specifically, the authors use the interface automata networks to model the component-based system designs which include a set of interface automata synchronized by shared actions, and the scenario-based specifications are specified by UML sequence diagrams. Based on investigating the reachability graph of the state space of the interface automata networks, the authors develop several algorithms to check the existential consistency and mandatory consistency including the forward, backward and bidirectional consistency.

Keywords component-based design; interface automata; model checking; sequence diagrams; UML

1 引 言

构件式软件系统设计方法已经成为目前软件工程中的研究热点.它通过构造或复用软件模块来构建复杂的软件系统,简化开发过程,提高开发效率.组合系统的整体行为通过构件之间的交互来完成,这种交互通常描述为某种基于场景的规约.因此,对组合构件之间的接口数据的兼容性和接口行为的兼容性进行分析和检验是构件式系统设计方法中提高

收稿日期:2005-01-17;修改稿收到日期:2005-11-15.本课题得到国家自然科学基金(60425204,60233020,60573085)、国家"九七三"重点基础研究发展规划项目基金(2002CB312001)和江苏省自然科学基金(BK2004080)资助.胡 军,男,1973年生,博士研究生,主要研究方向为软件工程、形式化方法、构件式软件设计、嵌入式软件建模与验证.E-mail:hujun@seg.nju.edu.cn.于笑丰,男,1975年生,博士研究生,主要研究方向为软件工程、模型驱动转换与验证、Web服务.张 岩,男,1974年生,博士研究生,主要研究方向为软件工程、形式化方法、面向服务的计算、软件度量.王林章,男,1973年生,博士,主要研究方向为软件工程、模型驱动的软件测试方法、软件度量.李宣东,男,1963年生,教授,博士生导师,主要研究领域为软件工程、形式化方法、软件验证.郑国梁,男,1936年生,教授,博士生导师,主要研究领域为软件工程、面向对象设计方法学.

系统可靠性的一个非常重要的途径.

有效地复用已有构件需要某种语言来准确地描述构件的接口性质并支持在设计阶段构件组合的相关性质的检查.通常的形式化方法(如类型系统)对接口性质的检查局限在输入输出数值的静态检查上.实际上构件的设计者可能会基于范围更广泛的外部环境假设,比如,在面向对象的类设计中,要求在调用类的某个方法之前,必须首先调用相关的初始化方法;在有实时性要求的嵌入式软件中,一个控制构件常常需要假定其传感器的输入是按照确定的频率更新等等,这就存在一个时序要求的前提假设,因此需要有语义更为丰富的接口模型来描述.在本文中我们使用了一种基于自动机的形式化语言——接口自动机(Interface Automata,IA)[1]及其自动机网络来描述构件式系统的行为设计模型.

统一建模语言(Unified Modeling Language,UML)[2]是目前软件工业界的一个标准.它是一种半形式化的可视化建模语言,提供了一系列有效的符号表示,使人们可以从不同的角度和不同的抽象层次对复杂的软件系统进行建模与分析.在构件式系统设计中,我们可以使用 UML 中的顺序图来描述用户的需求规约.顺序图通过在多个构件之间的消息发送和接收的序列来表明在系统行为的一个场景中这些构件是如何进行交互的.

在构件式系统设计阶段,构件之间交互行为的动态兼容性检查不仅包括构件验证接口时序方面的要求,还包括检查组合接口的行为是否满足相应的需求规约;如组合接口的行为是否一定满足某个顺序图规约;是否一定不满足某个顺序图或者是否满足多个顺序图之间的条件约束关系等等.本文工作的主要目的就是对构件式组合系统动态行为是否满足给定的需求规约的检验问题进行研究.

本文第 2 节中给出几种典型的一致性验证问题的非形式化描述;第 3 节中给出 UML 顺序图模型的形式化定义;第 4 节给出接口自动机和作为系统组合行为描述的接口自动机网络的形式化定义以及相应的兼容可达图;第 5 节和第 6 节则分别对存在一致性、强制一致性等检验问题进行详细分析,并给出相应的验证算法;最后一节介绍相关工作以及我们进一步的研究方向.

2 一致性验证的相关问题

所谓场景(scenario)就是描述系统中相关的各个构件之间如何进行交互以完成我们所关心的操作的某一个系统功能.通常,场景描述中给出了构件之间消息传递的先后交互次序,并不涉及系统的实现细节,形式上直观易懂.因此,作为一种有效的系统设计需求描述形式,基于场景的系统规约模型在工业界和学术界都得到了广泛的应用与关注,不仅可以作为设计人员与用户之间交流的一种良好工具,同时也可以作为一种规范的系统文档,以供进一步的分析与验证.常见的用来描述场景的模型语言包括 MSC(Message Sequence Charts)、UML 中的顺序图(sequence diagram)、协作图(collaboration diagram)以及 LSC(Live Sequence Charts)[3]等.对于系统的动态行为,我们不仅关心某一个场景需求是否在系统行为中出现,同时也关心多个系统场景需求之间的逻辑关系.本文的研究主要是针对以下几个一致性验证问题(见图 1):

(1)存在一致性(existential consistency)验证.用于验证某个特定的场景 D 是否在系统的所有行为中至少出现一次,或者某个指定的场景 D 是否在系统的所有行为中一定不会出现;

(2)前向强制一致性(forward mandatory consistency)验证.用于验证当某个条件场景 D_1 出现时,场景 D_2 一定会随之在系统后续行为中发生;

(3)逆向强制一致性(backward mandatory consistency)验证.用于验证当某个条件场景 D_1 出现时,场景 D_2 一定在 D_1 之前就在系统的行为中发生;

(4)双向强制一致性(bidirectional mandatory consistency)验证.用于验证当两个条件场景 D_1,D_2 在系统一个行为中先后出现时,在这两个场景之间一定有 D_3 发生.

图 1 中直观地描述了上述几种情形的一致性验证形式.每个一致性验证问题的形式语义分析将在本文后面几节中进行详细讨论.

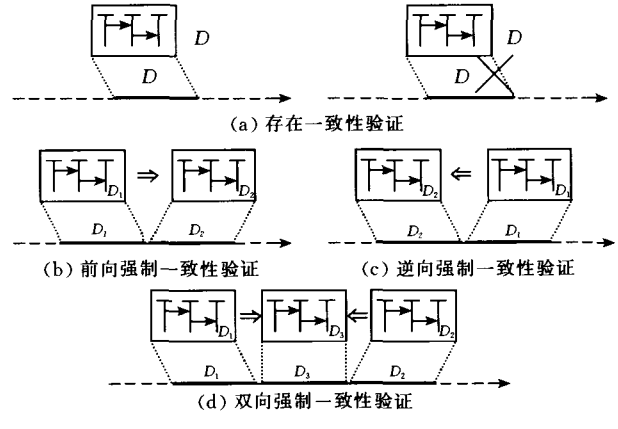


图 1 几种不同形式的一致性验证问题

3 UML 顺序图模型

本文中，我们的构件式系统设计使用了 UML 顺序图模型来描述系统运行过程中可能出现的一个行为场景. 直观上看，顺序图中的各个构件沿着水平方向排列，纵向的纬度是时间轴，从上至下表示时间的增长. 每一个构件都有相应的生命线，每一个消息都用带箭头的水平线来表示，如图 2 中所示. 我们暂不考虑带选择分支的情形.

图 2 描述了航空机载防撞告警系统 TCAS(Traffic Collision Avoidance System)中的一个飞行冲突探测与避让的场景. 图中右侧的注释表示各消息的全名. TCAS 系统用于防止本机与其临近空域中飞行的航空器之间发生危险接近以及与地面障碍物之间危险碰撞. 其通过接收空中交通控制系统中的二次雷达信号以及机载二次雷达应答机的信息来判断本机与它机之间的飞行间隔是否违反了规定的标准间隔，并向飞行管理系统和飞行员提供相应的告警信息以及可能的避让机动动作，同时进行航迹监控. 此系统有三个基本组成构件：航迹探测单元(Track Detector)、冲突管理单元(Collision Manager)和告警控制单元(Alarm Control). 其中，航迹探测单元主要负责获取相邻空域中飞行的飞机的航迹信息，根据飞行计划对获得的航迹信息进行识别和匹配，并判断其是否与本机之间的间隔违反标准间隔；冲突管理单元主要负责产生告警信息和告警解除信息，包括不同的告警等级和警报形式，并根据冲突航迹信息给出可能的避让机动飞行动作的建议；告警控制单元则负责从飞行管理系统中获取本机相关的飞行信息，并向飞行管理系统和飞行人员传递告警信息和告警解除信息. TCAS 系统的基本防撞告警功能就是通过这三个构件之间不同的交互来完成的. 如图 2 中的场景描述了下面情形：当 Track Detector 发现邻近空域中有某架飞机与本机之间的间隔小于标准间隔，则向 Collision Manager 给出相关的航迹和危险间隔信息；Collision Manager 通过判断形成相应的告警信息和避让动作建议，传递给 Alarm Control；Alarm Control 则将飞行管理系统所采取的避让动作信息返回给 Collision Manager；之后当飞机之间的间隔距离已经符合安全间隔时，Track Detector 向 Collision Manager 给出安全间隔信息，经 Collision Manager 判断，然后向 Alarm Control 给出警告消除的信息，并由 Alarm Control 返回相应的飞行信息. 显然，与文字的描述相比，顺序图更加清楚直观地表示了上述的交互次序和信息.

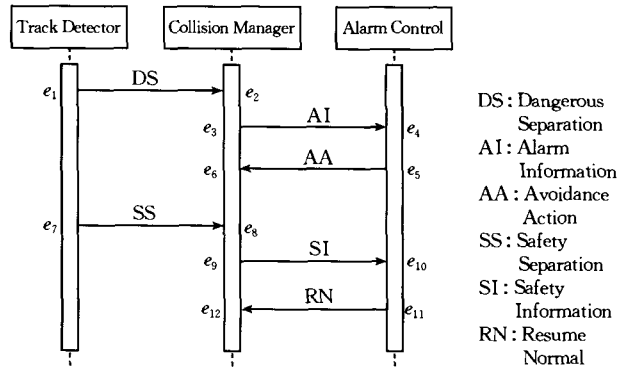


图 2　TCAS 系统中飞行冲突探测场景的 UML 顺序图模型

顺序图中的消息发送以及消息的接收使用事件来表示. 不失一般性，假定每一个顺序图对应于一个或多个可视序列[4]. 每个可视序列中的元素都是一个事件对(e_1,e_2)；其中(e_1,e_2)表示事件 e_1 在事件 e_2 之前发生. 在图中只要当出现满足下述关系时，事件 e_1 就会被视为在事件 e_2 之前发生：

(1)因果关系. e_1 为消息发送事件，e_2 为此消息对应的接收事件；

(2)控制关系. 在同一个构件的生命线轴上，事件 e_1 出现于事件 e_2 的上方，且 e_2 是消息发送事件. 这个事件顺序表明一个发送事件可以等待其它事件的发生. 但是一个构件只能控制自身发送消息事件的时间先后；

(3)FIFO 顺序关系. 在同一个构件的生命线轴上，接收事件 e_1 出现于接收事件 e_2 的上方，并且相对应的发送事件 e_1' 和 e_2' 同时出现在另外一个构件的生命线轴上，且 e_1' 位置高于 e_2'.

对于顺序图中的任何消息而言，可能是同步的，也可能是异步的. 异步的消息会消耗时间；同步的消息则在其发送和接收事件之间不会有时间延迟. 本文中假定顺序图中的消息都是同步消息. 我们使用这种构件接口之间消息交互的顺序图作为构件式系统某个行为场景的规约说明.

定义 1.　一个顺序图是五元组 $D=(C,E,M,L,W)$，其中：

C 表示有穷的构件集；

E 表示有穷的事件集；

M 表示有穷的消息集. 对每一个消息 $m\in M$，分别使用 $m!$ 和 $m?$ 来表示消息 m 的发送和接收事件；对任一个事件 $e\in E$，都对应一个消息 $m\in M$ 的发送和接收事件，即 $\tau(e)=m!$ 或者 $\tau(e)=m?$；

L 表示 $E\to C$ 是一个标记函数，将每一个事件

$e\in E$ 分配给一个构件 $L(e)\in C$;

W 表示元素形式为 (e,e') 的有穷集,其中 e, $e'\in E$,且 $e\neq e'$,表示 D 中的每一个可视序列对.

就图 2 中的 TCAS 系统的顺序图模型而言,其形式化定义表示如下:

$C=\{$Track Detector, Collision Manager, Alarm Control$\}$;

$E=\{e_1,e_2,e_3,e_4,e_5,e_6,e_7,e_8,e_9,e_{10},e_{11},e_{12}\}$;

$M=\{$DS,AI,AA,SS,SI,RN$\}$;

$\tau(e_1)=$DS!;$\tau(e_2)=$DS?;$\tau(e_3)=$AI!;$\tau(e_4)=$AI?;

$\tau(e_5)=$AA!;$\tau(e_6)=$AA?;$\tau(e_7)=$SS!;$\tau(e_8)=$SS?;

$\tau(e_9)=$SI!;$\tau(e_{10})=$SI?;$\tau(e_{11})=$RN!;$\tau(e_{12})=$RN?;

$L=\{(e_1$,Track Detector$),(e_2$,Collision Manager$),(e_3$,Collision Manager$),(e_4$,Alarm Control$),(e_5$,Alarm Control$),(e_6$,Collision Manager$),(e_7$,Track Detector$),(e_8$,Collision Manager$),(e_9$,Collision Manager$),(e_{10}$,Alarm Control$),(e_{11}$,Alarm Control$),(e_{12}$,Collision Manager$)\}$;

$W=\{(e_1,e_2),(e_1,e_7),(e_2,e_3),(e_7,e_8),(e_2,e_8),(e_2,e_9),(e_3,e_4),(e_3,e_9),(e_6,e_9),(e_6,e_{12}),(e_8,e_9),(e_9,e_{10}),(e_4,e_5),(e_4,e_{10}),(e_4,e_{11}),(e_5,e_6),(e_5,e_{11}),(e_{10},e_{11}),(e_{11},e_{12})\}$.

顺序图刻画了系统运行的一个场景,其运行过程就表现为一个事件的序列 $e_0\hat{}e_1\hat{}\cdots\hat{}e_m$,其中事件 e_{i+1} 在事件 e_i 之后发生$(1\leq i\leq m-1)$. 由于在控制关系中,可能存在不确定的接收消息的先后次序,因此一个顺序图可能允许多个事件序列.

定义 2. 对任何一个顺序图 $D=(C,E,M,L,W)$,事件序列 $e_0\hat{}e_1\hat{}\cdots\hat{}e_n$ 是 D 的一个有效事件迹(trace)当且仅当以下的条件被满足:

(1)E 中所有的事件都在这个序列中出现,而且每个事件仅发生一次;即 $\{e_0,e_1,\cdots,e_n\}=E$,且对于任意 $i,j(i\neq j,0\leq i,j\leq n)$,有 $e_i\neq e_j$,

(2)$e_0,e_1,\cdots,e_n$ 满足 W 定义的可视序列,即对任何 e_i 和 e_j,当 $(e_i,e_j)\in W$ 时,一定有 $0\leq i<j\leq n$.

如:事件序列 $e_1\hat{}e_7\hat{}e_2\hat{}e_3\hat{}e_8\hat{}e_4\hat{}e_5\hat{}e_6\hat{}e_9\hat{}e_{10}\hat{}e_{11}\hat{}e_{12}$ 与 $e_1\hat{}e_2\hat{}e_7\hat{}e_8\hat{}e_3\hat{}e_4\hat{}e_5\hat{}e_6\hat{}e_9\hat{}e_{10}\hat{}e_{11}\hat{}e_{12}$ 都是顺序图 2 中的有效事件迹.

4 接口自动机和构件式系统设计

4.1 接口自动机(Interface Automata,IA)

构件对外表现的接口性质包括静态的和动态的两个方面. 静态的是指接口参数的数量、类型的要求. 动态的是指接口与外界环境之间交互的时序要求,可以看作是构件与构件环境之间某种形式的通信协议. 接口自动机就是用来刻画软件构件接口的时序特征的一种形式化语言. 它描述了使用构件时其对外界环境的输入假设和输出保证,即构件内方法被调用的先后次序以及构件向外环境输出调用信息或结果的次序.

接口自动机采用与传统组合兼容性不同的思想,认为:构件的设计总是基于一定的环境假设,当两个自动机组合时,它们的环境假设也应该同时组合在一起. 那么只要存在某个环境使组合假设得以满足,则这二者可兼容. 其语法形式与 I/O 自动机[5]相似,都基于有限转换系统. 系统的每个状态上可能有输入、输出或内部动作. 在接口自动机中,明确地表示出在当前状态上只有哪些输入动作是可接收的,而其它的输入动作是不可接收的,即非法的输入. 构件对环境的假设就是通过对系统状态上的某些输入动作做出限制来达到的. 如图 3 所示.

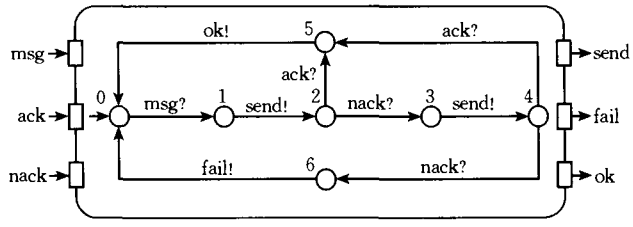


图 3 通信组件 Communication 的接口自动机模型图

图 3 中给出了一个用于消息传送的服务构件 Communication 直观形式的接口自动机模型. 此构件提供一个外部调用方法 msg,当此方法被调用后,构件返回 ok 或 fail 来表示消息发送成功与否. 为了完成这个服务,它需要与一个通信信道交互,调用信道的 send 方法,并获得返回的 ack,然后返回通信成功 ok;若连续两次 send 都收到 nack,则返回通信失败 fail. 从图中可以看出:输入动作 msg 仅在状态 0 可以接受,在其余的状态上则为非法;这就刻划了对环境的假设:一旦环境调用了 msg 方法,则在下一次调用 msg 之前,环境必须要等到 Communication 返回 ok 或者 fail. 其余状态如此类推. 图中的!表示输出动作,?表示输入动作.

接口自动机的形式化定义如下.

定义 3. 接口自动机是一个六元组 $P=\langle V_P,V_P^{\mathrm{Init}},A_P^{\mathrm{I}},A_P^{\mathrm{O}},A_P^{\mathrm{H}},\Gamma_P\rangle$,其中:

V_P 表示状态集,每一个状态为 $v_i(0\leq i\leq|V_P|)$;

$V_P^{\mathrm{Init}}\subseteq V_P$ 表示初始状态集,V_P^{Init} 至多包含一个状态 v_P^{Init};如果 V_P^{Init} 为空集,则自动机 P 为空;

$A_P^{\mathrm{I}},A_P^{\mathrm{O}},A_P^{\mathrm{H}}$ 分别表示输入、输出和内部动作集,

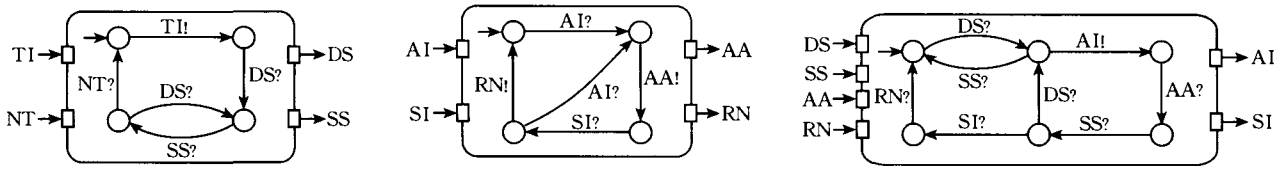
三者交集为空；当动作 $a\in A_P^{\mathrm{I}}, A_P^{\mathrm{O}}$ 或者 A_P^{H} 时，相应的转换 (v,a,v') 分别称为输入、输出或中间转换；

$\Gamma_P\subseteq V_P\times A_P\times V_P$ 表示所有的转换的集合.

P 的所有动作集表示为 $A_P=A_P^{\mathrm{I}}\cup A_P^{\mathrm{O}}\cup A_P^{\mathrm{H}}$. 对于某一状态 $v\in V_P$，当存在转换 $(v,a,v')\in\Gamma_P, v'\in V_P$ 时，称动作 $a\in A_P$ 在此状态上是可激活的. 接口自动机不要求对任一状态 $v\in V_P$，都有 $A_P^{\mathrm{I}}=A_P^{\mathrm{I}}(v)$. 对于一个接口自动机 $P=\langle V_P, V_P^{\mathrm{Init}}, A_P^{\mathrm{I}}, A_P^{\mathrm{O}}, A_P^{\mathrm{H}}, \Gamma_P\rangle$，状态转换序列 $v_0\xrightarrow{a_0}v_1\xrightarrow{a_1}\cdots\xrightarrow{a_{n-1}}v_n\xrightarrow{a_n}v_{n+1}$ 是 P 的一个行为当且仅当 $v_0\in V_P^{\mathrm{Init}}$，且对每一个 $i(0\le i\le n)$，有 $(v_i,a_i,v_{i+1})\in\Gamma_P$.

4.2 接口自动机网络(Interface Automata Networks, IAN)

我们使用接口自动机网络作为构件式系统的设计模型. 系统中每个构件的行为都使用一个相应的接口自动机表示. 整个系统由一个接口自动机集组成，并通过构件接口之间交互的共享动作来进行同步组合. 图 4 中所示即为 TCAS 系统中三个构件 Track Detector, Collision Manager, Alarm Control 的接口动态行为的部分状态转化图.



(a) Track Detector IA　　(b) Alarm Control IA　　(c) Collision Manager LA

TI: Track Information; NT: No Tracks; DS: Dangerous Separation; AI: Alarm Information; AA: Avoidance Action; SS: Safety Separation; SI: Safety Information; RN: Resume Normal

图 4　TCAS 系统的接口自动机网络模型图

为描述方便，下文中使用 IA 表示接口自动机，用 IAN 表示接口自动机网络. 以下给出接口自动机网络相关的形式化定义. 我们记 $shared(P_i,P_j)=A_{P_i}\cap A_{P_j}=(A_{P_i}^{\mathrm{O}}\cap A_{P_j}^{\mathrm{I}})\cup(A_{P_i}^{\mathrm{I}}\cap A_{P_j}^{\mathrm{O}})(1\le i,j\le n, i\ne j)$ 为 P_i 和 P_j 之间的共享动作. 所谓的共享动作就是一个 IA 的输出动作正好是另外一个 IA 的输入动作；动作名正好相同只是在不同的 IA 中分属于输入动作集或输出动作集. 组合之后出现的共享动作即作为 IAN 的内部动作，它实际上包含了一对输入和输出动作. 构件之间的交互就是通过共享动作来表达，并且每个共享动作一定是在两个构件之间进行，不允许出现三个及以上的构件共享同一个动作. 接口自动机其它的组合细节参看文献[1].

定义 4.　一个接口自动机网络 IAN 是二元组 $N=(H,S)$，其中：

$H=\{P_1,P_2,\cdots,P_n\}$ 表示可组合的接口自动机集；

$S=\{shared(P_i,P_j)\mid 1\le i,j\le n, i\ne j\}$，表示所有的共享的动作集.

IAN 的状态集、动作集及组合状态之间的转换定义如下.

定义 5.　$N=(H,S)$ 是一个 IAN，其中 $H=\{P_1,P_2,\cdots,P_n\}$，且 $P_i=\langle V_{P_i}, V_{P_i}^{\mathrm{Init}}, A_{P_i}^{\mathrm{I}}, A_{P_i}^{\mathrm{O}}, A_{P_i}^{\mathrm{H}}, \Gamma_{P_i}\rangle(1\le i\le n)$；

N 的状态集为 $V_N=(V_{P_1}\times V_{P_2}\times\cdots\times V_{P_n})$，其中每一个组合状态形为 $\bar v=(v_1,v_2,\cdots,v_n)$，$(v_i\in V_{P_i}, 1\le i\le n)$；

N 的初始状态集为 $V_N^{\mathrm{Init}}\subset V_N$，$V_N^{\mathrm{Init}}$ 中最多包含一个初始状态为 $\bar v^{\mathrm{Init}}=(v_{P_1}^{\mathrm{Init}}, v_{P_2}^{\mathrm{Init}},\cdots,v_{P_n}^{\mathrm{Init}})$；

N 的动作集为 $A_N=A_N^{\mathrm{I}}\cup A_N^{\mathrm{O}}\cup A_N^{\mathrm{H}}$，其中：输入动作集为 $A_N^{\mathrm{I}}=(\bigcup\limits_{1\le i\le n}A_{P_i}^{\mathrm{I}})/S$；输出动作集为 $A_N^{\mathrm{O}}=(\bigcup\limits_{1\le i\le n}A_{P_i}^{\mathrm{O}})/S$；内部动作集为 $A_N^{\mathrm{H}}=(\bigcup\limits_{1\le i\le n}A_{P_i}^{\mathrm{H}})\cup S$.

定义 6.　$N=(H,S)$ 是一个 IAN，其中 $H=\{P_1,P_2,\cdots,P_n\}$，且 $P_i=\langle V_{P_i}, V_{P_i}^{\mathrm{Init}}, A_{P_i}^{\mathrm{I}}, A_{P_i}^{\mathrm{O}}, A_{P_i}^{\mathrm{H}}, \Gamma_{P_i}\rangle(1\le i\le n)$；组合状态 $\bar v=(v_1,v_2,\cdots,v_n)$，$\bar v'=(v_1',v_2',\cdots,v_n')$. 当满足以下任一个条件的时候，系统可以通过动作 a 从状态 $\bar v$ 到达状态 $\bar v'$，用 $\bar v\xrightarrow{a}\bar v'$ 来表示：

(1)对于一个动作 $a\notin S$，在 $P_i(1\le i\le n)$ 存在一个转换 $(v_i,a,v_i')\in\Gamma_i$，而且对于任何 $j(j\ne i, 1\le j\le n)$，有 $v_j=v_j'$；

(2)对于一个动作 $a\in shared(P_i,P_j)(1\le i,j\le n, i\ne j)$，在 P_i 中存在一个转换 $(v_i,a!,v_i')\in\Gamma_{P_i}$ ($a!$ 表示 a 是输出动作)；同时，在 P_j 中存在一个转换 $(v_j,a?,v_j')\in\Gamma_{P_j}$ ($a?$ 表示 a 是输入动作)，且对于任何的 $r(r\ne i,j; 1\le r\le n)$，有 $v_r=v_r'$.

由于输入动作并非在每一个状态上都是可以激活的，所以 IAN 中可能会出现这种情况：在某一个组合状态上一个 IA 相对于另外一个 IA 有输出动作时，后者并没有相应的输入动作对应接收. 这实际上正好表达了两个具有共享接口的构件对外界环境

假设相互之间有矛盾的部分.这类组合状态称之为非法状态.在检验 IAN 的行为时,需要去除组合状态集中可达的非法状态.而可达的合法状态表示两个接口正常的输入、输出交互的部分.IAN 的非法状态集定义如下.

定义 7. $N=(H,S)$是一个 IAN,其中 $H=\{P_1,P_2,\cdots,P_n\}$,且 $P_i=\langle V_{P_i},V_{P_i}^{\text{Init}},A_{P_i}^{\text{I}},A_{P_i}^{\text{O}},A_{P_i}^{\text{H}},\Gamma_{P_i}\rangle(1\leq i\leq n)$;其非法状态集合为

$$illegal(N)=\{(v_1,v_2,\cdots,v_n)\in N\mid \exists(v_i,v_j)(i\neq j;1\leq i,j\leq n),\ \exists a\in shared(P_i,P_j),(a\in A_{P_i}^{O}(v_i)\wedge a\notin A_{P_j}^{I}(v_j)\vee a\in A_{P_j}^{O}(v_j)\wedge a\notin A_{P_i}^{I}(v_i))\}.$$

只要某一个组合状态上所有可能的共享动作中有一个出错,即认为此组合状态为非法状态.从非法状态集出发,由文献[1]中给出的逆向可达性算法,可构造出只包含兼容组合状态的 IAN,即 $comp(N)$.在 $comp(N)$的状态集中不再包含非法状态和那些从初始状态出发不可达的状态.

兼容的 IAN 的行为定义如下.

定义 8. 状态转换序列 $\bar{v}_0\xrightarrow{a_0}\bar{v}_1\xrightarrow{a_1}\cdots\xrightarrow{a_{n-1}}\bar{v}_n\xrightarrow{a_n}\bar{v}_{n+1}$是 $comp(N)$的一个行为当且仅当 $\bar{v}_0\in V_{comp(N)}^{\text{Init}}$,且对每一个 $i(0\leq i\leq n)$,有$(\bar{v}_i,a_i,\bar{v}_{i+1})\in\Gamma_{comp(N)}$.

4.3 兼容状态空间的可达图

现在构件式系统设计建模为接口自动机网络(IAN),基于场景的系统规约使用 UML 顺序图来描述.本质上,为了解决上述各种一致性验证问题,我们需要比较兼容 $comp(N)$的行为中事件序列与相应顺序图中的事件迹之间的关系,为此以下给出 $comp(N)$行为的轨迹以及事件序列之间的投影关系.设 $comp(N)=(H;S)$是兼容 IAN,状态转换序列$\alpha=\bar{v}_0\xrightarrow{a_0}\bar{v}_1\xrightarrow{a_1}\cdots\xrightarrow{a_{n-1}}\bar{v}_n\xrightarrow{a_n}\bar{v}_{n+1}$是 $comp(N)$的一个行为.提取行为 α 中包含的动作序列$a_0\wedge a_1\wedge\cdots\wedge a_{n-1}\wedge a_n$,替换每一个内部动作$a_i(0\leq i\leq n)$为一对输出和输入动作 $a_i?\wedge a_i!(0\leq i\leq n)$,其余的保持不变.将每一个输出(输入)动作 a 视为一个发送(接收)事件 e,我们就得到一个相应的事件序列 $e_0\wedge e_1\wedge\cdots\wedge e_r$ $(r\geq n)$,称其为行为 α 的轨迹.注意到在顺序图中的消息实际上就对应于 $comp(N)$中的某一个共享动作;消息的发送(接收)事件就是某个 IA 的一个输出(输入)动作.但是在顺序图的形式化定义 $D=(C,E',M,L,W)$中,对于一个 $e'\in E'$只是一个事件的名字,事件的真正的内容是 $\tau(e')$;而接口自动机中的输入和输出动作就是代表真正的事件.因此在进行事件比较时,实际上是需要比较 e 和 $\tau(e')$.设 σ 为行为 α 的轨迹,且 σ_1是 σ 的形如$e_0\wedge e_1\wedge\cdots\wedge e_m$的子串,对于一个有效事件迹形如 $f_0\wedge f_1\wedge\cdots\wedge f_n$ 的顺序图 D,若存在 $e_{k_0},e_{k_1},\cdots,e_{k_n}$同时满足:

(1) $0=k_0<k_1<\cdots<k_n=m$;

(2) $\tau(f_0)=e_{k_0}$,对于其它的 $\tau(f_i)(0<i\leq n)$有且只有一个 $e_{k_j}=\tau(f_i)(0<j\leq n)$;且

(3) 对于任何的 $i(o\leq i\leq n)$,任何的 $j\neq k_p(0\leq j\leq m,0\leq p\leq n)$,有 $\tau(f_i)=e_j$,

则称 σ_1是 σ 中关于 D 的一个投影.若 σ_1进一步满足:对于任何 $i(o\leq i\leq n)$,都有 $\tau(f_i)=e_k$,我们称 σ_1是 σ 中关于 D 的一个合法投影.在这种情况下,子串 σ_1事实上正好代表了由顺序图 D 所描述的场景在行为中的一个发生.

基于 $comp(N)$,我们可以构造一个相应的可达图 $G=(V;T)$其中:

V 是有穷的节点集,$comp(N)$中每一个状态 $\bar{v}_i$就对应于 V 中的一个节点 v_i,起始节点 v_0就是 $\bar{v}_{comp(N)}^{\text{Init}}$;

T 是有穷的边集,$comp(N)$中的每一个转换$(\bar{v}_i,a_i,\bar{v}_{i+1})\in\Gamma_{comp(N)}$就对应到图 G 中一条从节点 v_i到节点 v_{i+1}的转换边 $t_i=(v_i,l_i,v_{i+1})$,l_i是转换边上的标记,对应于转换中的动作 a_i.

显然,$comp(N)$的任一个行为就对应其可达图 G 中的一条路径.不妨设 $\rho=t_0\wedge t_1\wedge\cdots\wedge t_n$是 G 中的任一条路径,其相应边上的标记序列为 $l_0\wedge l_1\wedge\cdots\wedge l_n$,我们称可达图中的边序列 $t_i\wedge t_{i+1}\wedge\cdots\wedge t_{i+k}(0\leq i\leq n-k)$为路径 ρ 的子路径.

由可达图的构造可知,对于图中的任一条路径 ρ 的边上的标记序列,不妨设其形如 $l_0\wedge l_1\wedge\cdots\wedge l_m$,都对应于一个动作序列 $a_0\wedge a_1\wedge\cdots\wedge a_m$.使用与构造 $comp(N)$行为轨迹的同样方法,对于转换边上的标记是内部动作的,替换成一对输出和输入动作,其余的保持不变,且将每一个输出(输入)动作 a 视为一个发送(接收)事件 e,我们就得到一个动作序列 $a_0\wedge a_1\wedge\cdots\wedge a_s(s\geq m)$和相应的事件序列 $e_0\wedge e_1\wedge\cdots\wedge e_s(s\geq n)$.我们称得到的事件序列为路径 ρ 的轨迹.对于子路径$t_i\wedge t_{i+1}\wedge\cdots\wedge t_j$而言,其相应的轨迹我们用 $\sigma(t_i,t_j)(i\leq j)$来表示.这样,每一个转换边标记要么是一个输入或输出事件,要么对应于一对输入和输出事件;我们用 $\varphi(l_k)$来表示 l_k对应的事件对.

5 存在一致性验证

对于 $comp(N)$中任一行为的轨迹 σ 而言,可达

图 G 中一定存在一条路径，其轨迹正好就是 σ. 因此，从本质上看，前述的那些一致性验证问题都可以通过检验可达图中相关路径轨迹与相应顺序图中有效事件迹之间的关系来完成. 但是由于可达图中可能存在环路，其中路径数目可能是无穷的，路径的长度也可能是无穷的，我们希望只需要通过检查有穷的路径就可以得到相关一致性验证的结果. 虽然可达图的结点和边都是有穷的，但是通常的深度优先搜索算法框架并不足以解决这些一致性验证问题. 因为通常图的深度优先搜索算法在碰到环路的时候，仅仅是简单地判断当前节点以前是否被访问过，若是便回溯，不再作任何处理. 这样简单处理的结果将使得某些满足以下条件的有穷带环路径在遍历过程中被漏掉，即被顺序图 D 描述的场景片段一部分是在环路内部出现，另一部分则是在环路外出现. 为此，以下引进投影路径的概念.

5.1　投影路径

给定一个顺序图 $D=(C,E,M,L,W)$，对可达图中的任一条路径 $\rho=t_0{}^\wedge t_1{}^\wedge\cdots{}^\wedge t_n$，其轨迹可表示为 $\sigma(t_0,t_n)$. 若路径的 $\frac{1}{2}$ 同时满足以下条件，称其为关于 D 的投影路径：

(1)存在一个子路径的轨迹 $\sigma(t_i,t_n)(0\leq i\leq n)$，它是 $\sigma(t_0,t_n)$ 关于 D 的一个投影；

(2)对任何的 $j,k(0\leq j<k<i)$，ρ 中有 $t_j\neq t_k$；且

(3)对任何相邻的 $j,k(i\leq j<k\leq n,\varphi(l_j)\in E,\varphi(l_k)\in E)$，$\rho$ 中有 $t_g=t_h(j<g<h<k)$.

第一个条件表示路径 ρ 的末端部分可能恰好包含了由顺序图 D 描述的场景的一个出现；第二和第三个条件则对路径 ρ 中前后两段的环路做出了限制. 若 $\sigma(t_i,t_n)$ 进一步满足是关于 D 的合法投影，这意味着 ρ 的末端的确是顺序图 D 描述的场景的一个出现，则此时我们称投影路径 ρ 满足顺序图 D. 由以上条件可以判断，可达图 G 中的投影路径是有穷的，每一条投影路径的长度也是有穷的.

5.2　存在一致性验证

现在，首先我们来讨论存在一致性验证问题，即某个特定的场景是否在系统的所有行为中至少出现一次，或者某个指定的场景是否在系统的所有行为中一定不会出现. 具体的说，我们需要检查在 $comp(N)$ 的所有行为中，顺序图 D 所描述的场景是否至少出现一次，此时我们说兼容的接口自动机网络 $comp(N)$ 满足 D；或者顺序图 D 所描述的场景一定不会出现，此时我们说 $comp(N)$ 不满足 D. 前者表达了系统设计具有某种活性性质(liveness)，后者则表达了系统设计中所要考虑的安全性质(safety). 比如说，在 TCAS 系统中，我们希望所设计的系统是有效的，也就是说当发现周围空域中的确出现与本机之间的飞行间隔违反了标准的时候，那么系统进行告警和提供避让飞行动作建议的场景一定会发生，即顺序图 2 中描述的场景一定要出现；同时，我们也希望系统是安全可靠的，绝对不允许出现在没有获得有效的危险间隔航迹数据之前，就向飞行管理系统发出虚假的告警信息，即图 5 中所示的场景一定不可以在系统运行过程中出现.

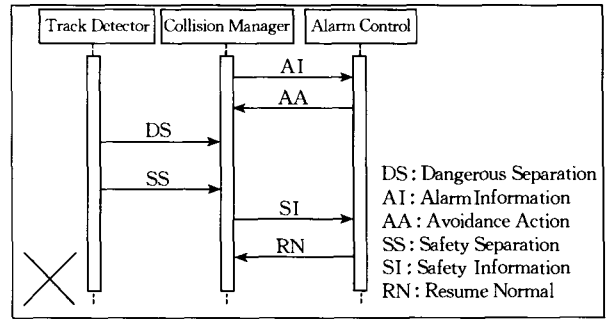


图 5　TCAS 系统中禁止出现的报假警的场景

基于以上关于投影路径的讨论，我们可以得到定理 1.

定理 1.　设 $comp(N)$ 是兼容的接口自动机网络，G 是其可达图，则对于一个顺序图 D 而言，$comp(N)$ 满足 D 当且仅当 G 中存在一条投影路径满足 D.

定理证明见附录. 定理 1 表明，存在一致性验证问题可以通过检查可达图 G 中所有的投影路径是否满足给定的顺序图 D 来给出检验结果. 基于定理 1，以下给出相应的检验算法，如图 6 中所示. 在此算法中仍然使用从初始节点开始对可达图进行深度优先遍历的框架，不过现在目的是在可达图中找出所有的投影路径，并检查其是否满足顺序图 D. 当前所访问的路径存放在 *current_path* 中，布尔变量 *satisfied* 表示在图 G 中是否存在一条投影路径满足顺序图 D. 算法结束时，若 *satisfied* 为 true 时，表明可达图中至少存在一条投影路径满足 D；*satisfied* 为 false 时，表明可达图中不存在任何一条满足 D 的投影路径. *error_trace*[]集合中记录的是某些路径片段，每一个路径片段所对应的事件序列与顺序图中的消息事件名和数量是一致的，只是事件发生的先后次序在某些位置上发生了错位. 记录这些出错的消息事件的交互序列可以用来对系统设计模型进一步理解和改进. 在算法遍历的过程中，每发现一个新的节点，算法就检查当前路径与新节

点是不是构成一条投影路径，若是，则进一步检查此路径是否满足顺序图. 若简单路径不满足 D，则记录相应出错的序列. 只有当新发现的节点使得当前路径构成一个投影路径的前缀时，才会被加入 *curren_path* 中，否则算法回溯. 所谓一个投影路径的前缀是指对于图 G 中某一条路径 ρ 而言，如果存在一个路径片段 ρ' 使得 $\rho^\wedge\rho'$ 成为一个投影路径，则称路径 ρ 为一个投影路径的前缀. 因为图中的投影路径是有限的，所以以上基于深度优先遍历的算法是可终止的，且算法复杂度与图中投影路径的数量和长度成正比，因此算法是有效的.

```
current_path:=⟨v0⟩; satisfied:= false; error_trace[]:=⟨⟩;
repeat
  node:= current_path 中的最后一个节点;
  if node 没有新的后继节点;
  then 删除 current_path 中的最后一个节点;
  else begin node:= node 的一个新的后继节点;
       if node 对应的当前路径是一条投影路径
       then begin
            if 此投影路径满足 D then satisfied:= true;
            else 将路径中为投影的片段放入 error_trace;
          end
       if node 对应的当前路径是一条投影路径的前缀
       then 将 node 加入到 current_path 中;
     end
until current_path=⟨⟩;
if satisfied then return true else return false.
```

图 6 存在一致性验证算法

6 强制一致性验证

6.1 前向强制一致性验证

对于前向强制一致性验证问题，我们使用 $S_F(D_1, D_2)$ 来表示由两个场景 D_1，D_2 所构成的系统行为规约：当条件场景 D_1 在系统行为中出现时，场景 D_2 一定随之在后续的行为中出现. 比如：在 TCAS 系统中，一旦发现周围空域中存在别的飞机与本机之间的飞行间隔小于标准，那么必须产生告警信息和相应的避让飞行动作建议. 图 7 中直观地描述了这种前向一致性，即当场景 D_1（发现有危险间隔的航迹）发生时，系统运行的后续场景 D_2 必然是提供告警信息和避让飞行动作建议的交互行为.

下面给出前向强制一致性检验问题的形式化定义.

设兼容自动机网络 $comp(N)$ 中任一行为形如 $\bar{v}_0 \xrightarrow{a_0} \bar{v}_1 \xrightarrow{a_1} \cdots \xrightarrow{a_{i-1}} \bar{v}_i \xrightarrow{a_i} \cdots \xrightarrow{a_{m-1}} \bar{v}_m$，当其满足以下条件时，则称 $comp(N)$ 满足 $S_F(D_1, D_2)$：当子序列 $\bar{v}_i \xrightarrow{a_i} \bar{v}_{i+1} \xrightarrow{a_{i+1}} \cdots \xrightarrow{a_{m-1}} \bar{v}_m$ 的轨迹正好是 D_1 的一个出现时，则 $comp(N)$ 中存在一个后续子序列 $\bar{v}_m \xrightarrow{a_m} \bar{v}_{m+1} \xrightarrow{a_{m+1}} \cdots \xrightarrow{a_{n-1}} \bar{v}_n$ $(m \le n)$ 满足其轨迹正好是 D_2 的一个出现.

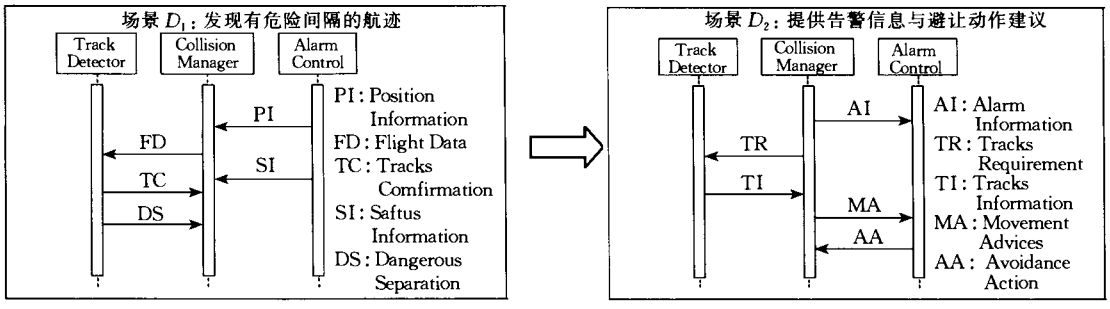


图 7 TCAS 中前向强制一致性验证场景

基于 $comp(N)$ 相应的可达图 G，以下引入前向扩展投影路径.

设 $t_0^\wedge t_1^\wedge \cdots^\wedge t_m^\wedge \cdots^\wedge t_n$ 为可达图中任一条路径，若其满足：子路径 $t_0^\wedge t_1^\wedge \cdots^\wedge t_m$ 是满足 D_1 的投影路径，且子路径 $t_{m+1}^\wedge t_{m+2}^\wedge \cdots^\wedge t_n$ 是关于 D_2 的投影路径，则称此路径为关于 D_2 的前向扩展投影路径. 如果 $\sigma(t_{m+1}, t_n)$ 进一步满足是关于 D_2 的合法投影，那么称此前向扩展投影路径满足 D_2. 对于 G 中任何一条满足 D_1 的投影路径 $\rho = t_0^\wedge t_1^\wedge \cdots^\wedge t_k$，我们可以构造一个关于 D_2 的前向扩展投影路径集合 $\Theta(\rho)$：

$$\Theta(\rho) = \{t_0^\wedge t_1^\wedge \cdots^\wedge t_k^\wedge \cdots^\wedge t_j' \mid t_i' = t_i (0 \le i \le k) \wedge t_{k+1}^\wedge t_{k+2}^\wedge \cdots^\wedge t_j' \text{是关于} D_2 \text{的投影路径}\}.$$

由上述投影路径的定义可知：集合 $\Theta(\rho)$ 中每一条路径是有穷的，路径的总数也是有穷的.

定理 2. $comp(N)$ 满足前向强制一致性规约 $S_F = (D_1, D_2)$ 当且仅当对其可达图 G 中任一条满足 D_1 的投影路径 ρ 而言，集合 $\Theta(\rho)$ 中存在一条前向扩展投影路径满足 D_2.

定理证明见附录. 基于定理 2，可以给出相应的检验算法框架，如图 8 中所示. 此算法同样是基于深度优先搜索的框架，其基本思想为：从可达图 G 的初始节点 v_0 开始，首先判断当前访问的节点与 *current_path* 所对应的路径是否成为一个满足 D_1 的投影路径；若是，则继续检查此路径所构成的前向

扩展投影路径集合 $\Theta(current\_path^{\wedge}node)$ 中是否存在一条路径满足 D_2. 每次访问一个新节点，需要判断 $current\_path^{\wedge}node$ 是否成为一个满足 D_1 的投影路径的前缀，若是，则将此节点加入 $current\_path$ 中；否则，算法进行回溯. 与存在一致性验证算法中类似，在此算法以及后面的几个算法框架中，也可以使用 *error_trace* 来记录相关的出错序列的信息，为描述方便，此处和后面的算法描述中不再提及. 此算法的复杂度与可达图中投影路径以及相关的前向扩展投影路径的长度和数量成正比.

```
current_path := ⟨v0⟩;
repeat
  node := current_path 中的最后一个节点;
  if node 没有新的后继节点
  then 删除 current_path 中的最后一个节点;
  else begin
       node := node 的一个新的后继节点;
       if 当前路径 current_path^node 是满足 D1 的投影路径
       then begin
            检查Θ(current_path^node)中是否存在一个路径
                满足 D2;
            if no return false;
          end
       if 当前路径current_path^node 是一个满足 D1 的投影
                路径的前缀
       then 将 node 加入到 current_path 中;
     end
until current_path = ⟨⟩;
return true.
```

图 8　前向强制一致性验证算法

6.2　逆向强制一致性验证

对于逆向强制一致性验证问题，我们使用 $S_B(D_1, D_2)$ 来表示由两个场景 D_1, D_2 所构成的系统行为规约：当条件场景 D_1 在系统行为中出现时，则场景 D_2 一定在 D_1 之前在系统的行为中发生. 比如在 TCAS 系统中的冲突告警和避让行为的场景之前，必须要有一个对发现的航迹信息进行识别和确认的过程，这是为了防止二次雷达应答机接收的杂波信号造成伪航迹信息，导致报假警的情形. 图 9 中直观描述了当场景 D_1（冲突告警与避让）发生的时候，系统行为的前驱场景 D_2 一定是一个对发现的航迹进行识别和确认的过程.

下面给出逆向强制一致性验证问题的形式化定义.

设一个兼容的自动机网络 $comp(N)$ 中的任一个行为形如 $\bar{v}_0 \xrightarrow{a_0} \bar{v}_1 \xrightarrow{a_1} \cdots \xrightarrow{a_{i-1}} \bar{v}_i \xrightarrow{a_i} \cdots \xrightarrow{a_{m-1}} \bar{v}_m$，当其满足以下条件时，则称 $comp(N)$ 满足 $S_B(D_1, D_2)$：当子序列 $\bar{v}_i \xrightarrow{a_i} \bar{v}_{i+1} \xrightarrow{a_{i+1}} \cdots \xrightarrow{a_{m-1}} \bar{v}_m$ 的轨迹正好是 D_1 的一个出现时，则在 $comp(N)$ 存在一个序列 $\bar{v}_0 \xrightarrow{a'_0} \bar{v}'_1 \xrightarrow{a'_1} \cdots \xrightarrow{a'_{j-1}} \bar{v}'_j \xrightarrow{a'_j} \cdots \xrightarrow{a'_{i-1}} \bar{v}_i$ 满足其子序列 $\bar{v}'_j \xrightarrow{a'_j} \bar{v}'_{j+1} \xrightarrow{a'_{j+1}} \cdots \xrightarrow{a'_{i-1}} \bar{v}_i$ 的轨迹正好是 D_2 的一个出现. 与构造前向扩展投影路径类似，基于 $comp(N)$ 相应的可达图 G，对于任一条满足 D_1 的投影路径 $\rho = t_0{}^{\wedge}t_1{}^{\wedge}\cdots{}^{\wedge}t_m{}^{\wedge}\cdots{}^{\wedge}t_n$，其中 $\sigma(t_m, t_n)$ 是 D_1 的一个出现，路径的前半段 $t_0{}^{\wedge}t_1{}^{\wedge}\cdots{}^{\wedge}t_{m-1}$ 用 $front(\rho)$ 表示，我们可以构造一个相应的逆向扩展投影路径集合 $\Delta(\rho)$：

$$\Delta(\rho) = \{t'_0{}^{\wedge}t'_1{}^{\wedge}\cdots{}^{\wedge}t'_m{}^{\wedge}\cdots{}^{\wedge}t'_n \mid t'_i = t_i (m \le i \le n) \wedge t'_0{}^{\wedge}t'_1{}^{\wedge}\cdots{}^{\wedge}t'_{m-1} \text{是关于 } D_2 \text{的投影路径}\}.$$

若路径 $t'_0{}^{\wedge}t'_1{}^{\wedge}\cdots{}^{\wedge}t'_{m-1}$ 是满足 D_2 的投影路径，则称集合 $\Delta(\rho)$ 中存在一条路径满足 D_2. 由投影路径的定义可知：集合 $\Delta(\rho)$ 是有穷的.

定理 3.　$comp(N)$ 满足逆向强制一致性规约 $S_B(D_1, D_2)$ 当且仅当对其可达图 G 中任一条满足 D_1 的投影路径 ρ 而言，集合 $\Delta(\rho)$ 中存在一条逆向扩展投影路径满足 D_2.

定理证明见附录. 基于定理 3，同样我们可以给出相应的检验算法，其基本框架与图 7 中验证前向强制一致性的算法是一致的，不同之处在于每次都是检查当前路径所构造的逆向扩展投影路径集 $\Delta(current\_path^{\wedge}node)$ 中是否存在一条路径是满足 D_2 的；算法的复杂度也是与可达图中满足 D_1 的投影路径以及相关的逆向扩展投影路径的长度和数量成正比. 算法的复杂度与可达图中满足 D_1 的投影路

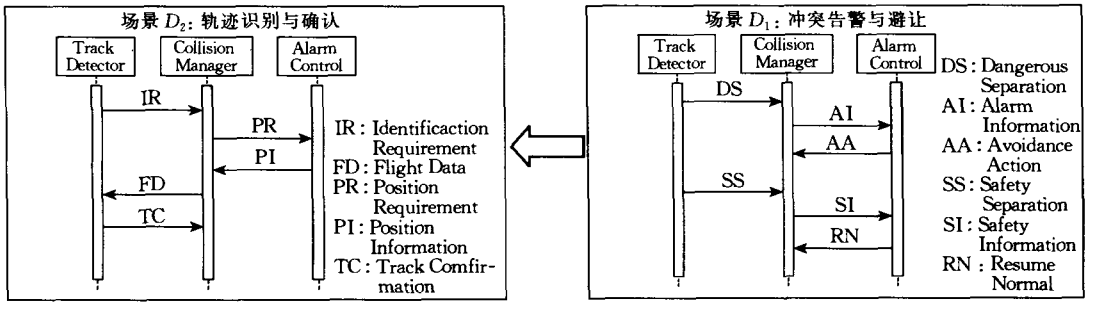


图 9　TCAS 中逆向强制一致性验证场景

径的长度和数量成正比.

6.3 双向强制一致性验证

对于双向强制一致性验证问题,我们使用 $S_D(D_1,D_2,D_3)$来表示由三个场景 D_1,D_2,D_3所构成的系统行为规约:当两个条件场景 D_1,D_2在系统一个行为中先后出现时,则在这两个场景之间一定有 D_3发生.图 10 中直观地描述了在 TCAS 系统中,在系统发出告警信息和避让动作建议的场景 D_1与恢复正常飞行动作的场景 D_2之间必须有一个判断当前本机与相关的飞机之间的飞行间隔是否已经满足规定的标准的场景 D_3,否则系统是不可靠的.

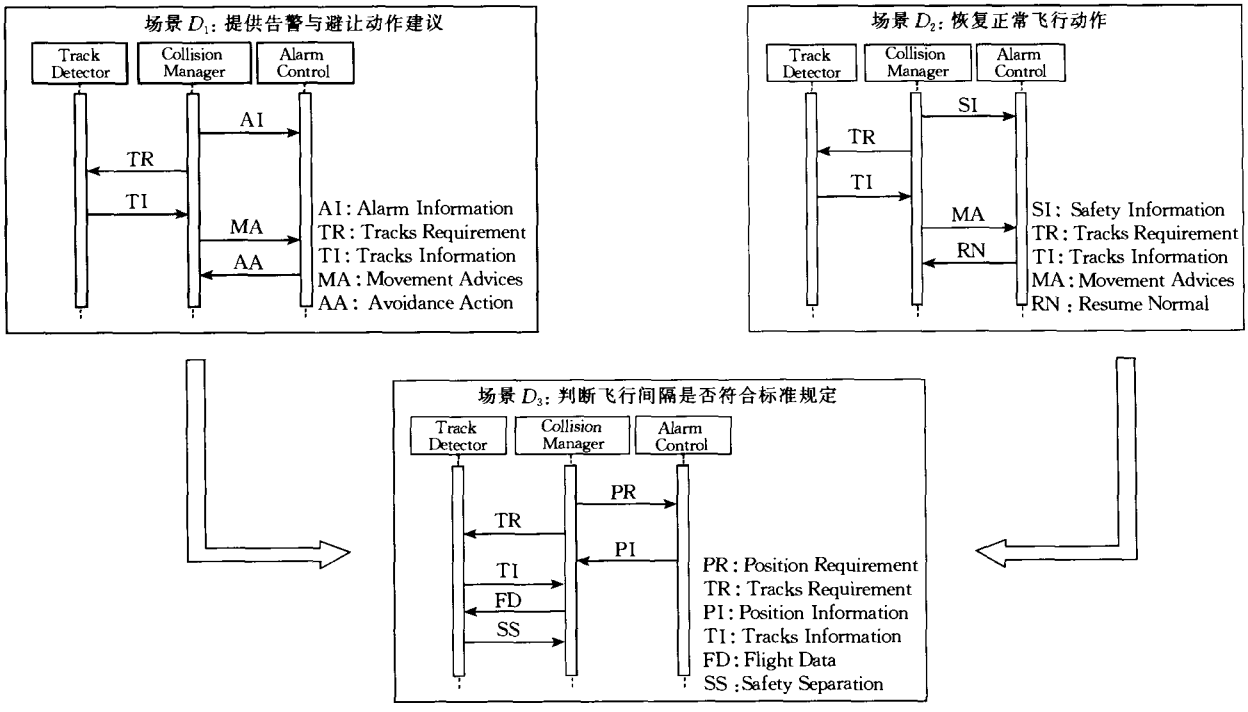


图 10 TCAS 中双向强制一致性验证场景

下面给出双向强制一致性验证问题的形式化定义.

设兼容自动机网络 $comp(N)$中任一个行为形如 $\bar{v}_0\xrightarrow{a_0}\bar{v}_1\xrightarrow{a_1}\cdots\xrightarrow{a_{k-1}}\bar{v}_k\xrightarrow{a_k}\cdots\xrightarrow{a_{l-1}}\bar{v}_l\xrightarrow{a_l}\cdots\xrightarrow{a_{m-1}}\bar{v}_m\xrightarrow{a_m}\cdots\xrightarrow{a_{n-1}}\bar{v}_n$,其中所有的 $\bar{v}_i(l\leq i\leq m)$都是不相同的,且子序列 $\bar{v}_k\xrightarrow{a_k}\bar{v}_{k+1}\xrightarrow{a_{k+1}}\cdots\xrightarrow{a_{l-1}}\bar{v}_l$的轨迹正好是 D_1的一个出现,子序列 $\bar{v}_m\xrightarrow{a_m}\bar{v}_{m+1}\xrightarrow{a_{m+1}}\cdots\xrightarrow{a_{n-1}}\bar{v}_n$的轨迹也正好是 D_2的一个出现;当此行为满足以下条件时,则称 $comp(N)$满足 $S_D(D_1,D_2,D_3)$:如果不存在一个子序列 $\bar{v}_i\xrightarrow{a_i}\bar{v}_{i+1}\xrightarrow{a_{i+1}}\cdots\xrightarrow{a_{j-1}}\bar{v}_j(l\leq i<j\leq m)$的轨迹是 D_1或者 D_2的一个出现,则在 $comp(N)$中存在一个子序列 $\bar{v}_l\xrightarrow{a_l}\bar{v}_{l+1}\xrightarrow{a_{l+1}}\cdots\xrightarrow{a_{m-1}}\bar{v}_m$,其轨迹正好是场景 D_3的一个出现.

基于构造扩展投影路径的类似思想,以下引入组合投影路径的概念.对 $comp(N)$的可达图 G 中的一条路径 $t_0{}^\wedge t_1{}^\wedge\cdots{}^\wedge t_m{}^\wedge\cdots{}^\wedge t_k{}^\wedge\cdots{}^\wedge t_n$,若其满足以下条件,称之为一条满足 D_1,D_2的组合投影路径:

(1)子路径 $t_0{}^\wedge t_1{}^\wedge\cdots{}^\wedge t_m$是满足 D_1的投影路径;

(2)子路径 $t_{m+1}{}^\wedge t_{m+2}{}^\wedge\cdots{}^\wedge t_n$是一条满足 D_2的投影路径,其中 $\sigma(t_{k+1},t_n)$就是 D_2的出现;

(3)$\sigma(t_{m+1},t_{k-1})$不是任一个 D_1或者 D_2的出现.

对于任何一条满足以上条件的组合投影路径 ρ,我们用 $middle(\rho)$来表示组合路径的中间路径片段 $t_{m+1}{}^\wedge t_{m+2}{}^\wedge\cdots{}^\wedge t_{k-1}$且可以构造一个组合投影路径集合 $\Omega(\rho)$:

$$\Omega(\rho)=\{t_0'{}^\wedge t_1'{}^\wedge\cdots{}^\wedge t_m'{}^\wedge\cdots{}^\wedge t_k'{}^\wedge\cdots{}^\wedge t_n'\mid t_i'=t_i(0\leq i\leq m)\wedge t_j'=t_j(k\leq j\leq n)\wedge\sigma(t_{m+1}',t_{k-1}')\text{是关于 }D_3\text{ 的一个投影.}$$

若 $\sigma(t_{m+1}',t_{k-1}')$是关于 D_3的一个合法投影,则称集合 $\Omega(\rho)$中存在一条路径满足 D_3.由投影路径的定义可知,集合 $\Omega(\rho)$是有穷的.

定理 4. $comp(N)$满足双向强制一致性规约 $S_D(D_1,D_2,D_3)$当且仅当对其可达图 G 中任一条满足 D_1,D_2的组合投影路径 ρ 而言,集合 $\Omega(\rho)$中存在一条路径满足 D_3.

定理证明见附录.基于定理 4,可以给出相应的

双向强制性检验算法. 其基本框架与前面两种强制一致性验证的算法是类似的，不同之处在于每次都是检查当前路径是否成为满足 D_1，D_2 的组合投影路径，然后继续检查所构造的组合投影路径集合 $\Omega(current\_path^{\wedge}node)$ 中是否存在一条路径是满足 D_3 的. 算法的复杂度是与可达图中满足 D_1，D_2 的组合投影路径的长度和数量成正比.

7　结束语

我们使用接口自动机及接口自动机网络来描述构件式系统的设计模型，使用 UML 顺序图表示基于场景的需求规约，对系统设计阶段的构件交互行为的动态兼容性进行了形式化检验. 从 2001 年接口自动机作为一种用来描述构件接口动态交互行为的形式语言被首次提出以来，已有不少相关的研究工作. de Alfaro 等继续在接口自动机的基础上进行时间约束和资源约束表达的扩充，给出了时间接口自动机(Timed Interface Automata, TIA)[6] 和资源接口(Resource Interface, RI)[7]，分别用于对实时环境和有限资源环境中构件接口交互行为和组合性质进行形式化的描述与验证. 相应的工具实现有 Wen 等人给出的从接口自动机到 I/O 自动机的转换方法[8]. 由于接口描述通常比相应的实现要简单得多，通过转换前后的对比，发现接口自动机模型的状态空间的确要比通常的 I/O 状态机模型小得多，这也是使用接口自动机进行模型检验的好处之一. Lee 等在 PtolemyII 系统中使用接口自动机设计了一个扩展的类型系统框架[9]，通过接口自动机之间的组合运算，对构件之间的交互进行兼容性检查，但没有进一步对系统行为是否满足某些需求规约进行检验. 其它基于自动机对构件式系统设计模型进行检验的研究工作还有：Schafer 等[10] 使用 UML 的状态机作为系统设计模型，UML 合作图作为规约说明，然后将状态机转换成模型检验工具 SPIN[11] 的输入，合作图转换成 Büchi 自动机集，使用 SPIN 进行可满足性的验证. Bultan 等给出了一个对 Web 服务组合的行为进行规约和验证的形式化框架[12]. Web 服务的行为使用 Mealy 机建模，规约使用 Web 服务之间的会话(conversation)来表达；通过自动机的组合和包含关系进行验证等. 相比较上述研究，本文的工作是在接口自动机组合兼容性检验的基础上对构件式系统设计模型的动态兼容性质给出了一个更为完整的验证框架. 我们的算法不仅可以对系统的动态行为的存在一致性性质进行验证，还可以对多个顺序图规约之间不同形式的强制一致性的条件性质进行验证. 此外，我们的工作是直接建立在对顺序图分析的基础上，与通常使用逻辑语言来表达系统相关性质相比，顺序图在工业界应用非常广泛、直观、易于使用和理解，这对于形式化方法和验证工具的推广使用具有非常重要的意义.

目前，构件式的软件开发方法正逐渐成为软件工程中的主流技术，如 CORBA，J2EE，.Net 等都是面向行为、以接口为中心的软件体系结构. 而以 Web 服务为代表的面向服务的计算模式(Service Oriented Computing, SOC)就是一种典型的构件式架构，它由一组通过互联网连接的具有模块性、可组合性、能动性和反应性的软件实体组成的开放式软件系统. 接口自动机构建了一个严格的形式系统来描述软件构件对环境的要求及其自身的行为特征，为研究基于构件式思想的系统行为提供了有力的工具支持. 此外，在嵌入式系统和实时系统领域，嵌入式软件系统的规模和复杂度的增大使得其已成为决定系统可靠性的主要因素. 相比较商业软件系统而言，嵌入式系统更能体现出构件式的体系结构特征，因而需要进一步考虑带时间约束和资源约束的系统接口行为的验证问题[13,14]. 这也是我们正在进行的研究工作之一. 基于本文工作以及文献[15,16]的研究，我们正在设计和实现一个模型分析与验证原型工具 Mecesd(Model cheker for component-based embedded software designs). 工具主要由图形用户接口模块、前端处理模块、非实时行为一致性验证模块、实时行为一致性模块、资源分析与验证模块、能耗分析与验证模块、核心算法模块、结果处理与显示模块以及验证信息数据库等几个部分构成. 它可以应用于基于构件的嵌入式软件开发的设计阶段，对设计者通常所关心的一些有关功能以及非功能方面的性质进行有效的分析和验证，最后的输出结果可以包含相应的出错反馈信息以进行系统设计的进一步修改和提高.

参　考　文　献

1　de Alfaro L., Henzinger T. A. Interface automata. In: Proceedings of the Joint 8th European Software Engineering Conference and the 9th ACM SIGSOFT International Symposium on the Foundations of Software Engineering, Vienna, Austria, 2001, 109～120

2　Booch G., Rumbaugh J., Jacobson I.. The Unified Modeling

Language User Guide, 2nd. Boston: Addison-Wesley, 2005

3 Damn W., Harel David.. LSCs: Breathing life into message sequence charts. Formal Methods in System Design, 2001, 19(1): 45～80

4 Peled D. A.. Software Reliability Methods. Springer, 2001

5 Lynch N. A.. Input/output automata: Basic, timed, hybrid, probabilistic, dynamic. In: Proceedings of the 14th International Conference on Concurrency Theory, Marseille, France, 2003, 187～188

6 de Alfaro L., Henzinger T. A., Stoelinga M.. Timed interfaces. In: Proceedings of the 2nd International Conference on Embedded Software, Grenoble, France, 2002, 108～122

7 Chakrabarti A., de Alfaro L., Henzinger T. A., Stoelinga M.. Resource interfaces. In: Proceedings of the 3rd International Conference on Embedded Software, Philadelphia, PA, USA, 2003, 117～133

8 Wen Y., Wang J., Qi Z.. Bridging refinement of interface automata to forward simulation of I/O automata. In: Proceedings of the 6th International Conference on Formal Engineering Method, Seattle, USA, 2004, 259～273

9 Lee E. A., Xiong Y.. System-level types for component-based design. In: Proceedings of the 1st International Workshop on Embedded Software, Tahoe City, CA, USA, 2001, 237～253

10 Schafer T., Knapp A., Merz S.. Model checking UML state machines and collaborations. Electronic Notes in Theoretical Computer Science, 2001, 55(3): 19～24

11 Holzmann G. J.. The model checker SPIN. IEEE Transactions on Software Engineering, 1997, 23(5): 279～295

12 Bultan T., Fu X., Hull R., Su J.. Conversation specification: A new approach to design and analysis of e-service composition. In: Proceedings of the 12th International World Wide Web Conference(WWW), Budapest, Hungary, 2003, 403～410

13 Li X., Cui M., Pei Y., Zhao J., Zheng G.. Timing analysis of UML activity diagrams. In: Proceedings of the 4th International Conference on the Unified Modeling Language, Toronto, Canada, 2001, 62～75

14 Li X., Lilius J.. Checking compositions of UML sequence diagrams for timing inconsistency. In: Proceedings of the 7th Asia-Pacific Software Engineering Conference, Singapore, 2000, 154～161

15 Hu J., Yu X., Zhang Y., Zhang T. *et al*. Scenario-based verification for component-based embedded software designs. In: Proceeding of the 34th International Conference on Parallel Processing Workshops, Oslo, Norway, 2005, 240～247

16 Hu J., Yu X., Zhang Y., Zhang T., Li X., Zheng G.. Scenario-based verification for real-time component-based embedded software designs. In: Proceedings of the IFIP International Conference on Embedded and Ubiquitous Computing, Nagakasi, Japan, 2005, 395～404

附录. 定理证明.

定理 1. 设 $comp(N)$ 是兼容的接口自动机网络，G 是其可达图，则对于一个顺序图 D 而言，$comp(N)$ 满足 D 当且仅当 G 中存在一条投影路径满足 D.

证明. 充分性. 当 $comp(N)$ 满足 D 时，由可达图 G 的构造方法可知，G 中存在一条路径 $\rho=t_0\hat{}t_1\hat{}\cdots\hat{}t_j\hat{}\cdots\hat{}t_k\hat{}\cdots\hat{}t_n$ 满足：其中 $\sigma(t_j,t_k)$ 是关于 D 的合法投影. 考虑路径 $\rho_1=t_0\hat{}t_1\hat{}\cdots\hat{}t_j\hat{}\cdots\hat{}t_k$，若子路径 $\rho_1=t_0\hat{}t_1\hat{}\cdots\hat{}t_{j-1}$ 中存在片段 $t_g\hat{}t_{g+1}\hat{}\cdots\hat{}t_h(0\leq g,h\leq j-1)$ 满足 $t_g=t_h$ 则在原路径中用 t_g 替换掉片段 $t_g\hat{}\cdots\hat{}t_h$. 反复进行此环路替换操作，直至 $t_0\hat{}t_1\hat{}\cdots\hat{}t_{j-1}$ 中每一个 $t_k(0\leq k\leq j-1)$ 都不相同. 然后，考虑子路径 $t_j\hat{}t_{j+1}\hat{}\cdots\hat{}t_k$，若其中存在片段 $t_u\hat{}t_{u+1}\hat{}\cdots\hat{}t_v(j<u,v<k)$ 满足：(1)不存在一个 $\varphi(l_i)\in E(E$ 是 D 中的事件集$)(u\leq i\leq v)$；(2)$t_u=t_v$，则用 t_u 替换掉原来路径中的片段 $t_u\hat{}\cdots\hat{}t_v$. 因为 $\sigma(t_j,t_k)$ 是关于 D 的合法投影，由上述构造方法可知，经过替换操作所得到的路径 ρ' 是一个满足 D 的投影路径.

必要性. 由可达图 G 的构造和投影路径的定义可直接证明. 证毕.

定理 2. $comp(N)$ 满足前向强制一致性规约 $S_F=(D_1,D_2)$ 当且仅当对其可达图 G 中任一条满足 D_1 的投影路径 ρ 而言，集合 $\Theta(\rho)$ 中存在一条前向扩展投影路径满足 D_2.

证明. 充分性. 当 $comp(N)$ 满足前向强制一致性规约 $S_F=(D_1,D_2)$ 时候，由可满足性定义以及可达图 G 的构造可知，对 G 中的任一条路径 ρ（不妨设其形如 $t_0\hat{}t_1\hat{}\cdots\hat{}t_i\hat{}\cdots\hat{}t_m$）而言，若其满足 $\sigma(t_i,t_m)$ 是关于 D_1 的一个合法投影的时候，则存在一条后继的子路径 ρ'，形如 $t_{m+1}\hat{}t_{m+2}\hat{}\cdots\hat{}t_n$，满足 $\sigma(t_{m+1},t_n)$ 是关于 D_2 的一个合法投影. 现在，我们考虑若在子路径 $t_0\hat{}t_1\hat{}\cdots\hat{}t_{j-1}$ 中存在一个路径片段 $t_j\hat{}t_{j+1}\hat{}\cdots\hat{}t_k$ $(0\leq j,k\leq i-1)$ 满足 $t_j=t_k$，即形成一个环路，则在原路径 ρ 中用 t_j 替换掉路径片段 $t_j\hat{}\cdots\hat{}t_k$. 在 $t_0\hat{}t_1\hat{}\cdots\hat{}t_{i-1}$ 中反复进行上述环路替换操作，使得最后的子路径 $t_0\hat{}t_1\hat{}\cdots\hat{}t_{i-1}$ 中的每一个 $t_j(0\leq j\leq i-1)$ 都不相同. 再考虑子路径 $t_i\hat{}t_{i+1}\hat{}\cdots\hat{}t_m$，若其中存在片段 $t_g\hat{}t_{g+1}\hat{}\cdots\hat{}t_h(i<g,h<m)$ 满足条件：(1)不存在一个 $\varphi(l_k)\in E_1(g\leq k\leq h)(E_1$ 是 D_1 中的事件集$)$；(2)$t_g=t_h$，则用 t_g 替换掉原路径中的 $t_g\hat{}\cdots\hat{}t_h$ 片段. 同样，在 $t_i\hat{}t_{i+1}\hat{}\cdots\hat{}t_m$ 中反复进行上述操作，直至其中没有满足上述条件的片段. 由投影路径的定义可知，此时我们就从路径 ρ 中得到了一条满足 D_1 的投影路径 $\rho_1:t_0\hat{}\cdots\hat{}t_i\hat{}\cdots\hat{}t_m$. 对于 ρ 的后续路径 $\rho'=t_{m+1}\hat{}t_{m+2}\hat{}\cdots\hat{}t_n$，我们考虑若是其中存在片段 $t_u\hat{}t_{u+1}\hat{}\cdots\hat{}t_v(m+1<u,v<n)$ 满足条件：(1)不存在一个 $\varphi(l_k)\in E_2(E_2$ 是 D_2 中的事件集$)(u\leq k\leq v)$；(2)$t_u=t_v$，则用 t_u 替换掉原来路径中的片段 $t_u\hat{}\cdots\hat{}t_v$. 反复进行此替换操作，最终得到后续路径 ρ_1'. 因为 $\sigma(t_{m+1},t_n)$ 是关于 D_2 的合法投影，这意味着 ρ_1' 是一条满足 D_2 的投影路径. 由上述构造方法可知，对于 G 中任一条满足 D_1 的投影路径 ρ_1 而言，存在一条路径 $\rho_1\hat{}\rho_1'$ 是前向扩展投影路径集合 $\Theta(\rho_1)$ 中的一个元素，且满足 D_2.

必要性. 由前向扩展投影路径的定义以及 $comp(N)$ 与可达图 G 的构造关系直接可证. 证毕.

定理 3. $comp(N)$ 满足逆向强制一致性规约 $S_B(D_1, D_2)$ 当且仅当对其可达图 G 中任一条满足 D_1 的投影路径 ρ 而言，集合 $\Delta(\rho)$ 中存在一条逆向扩展投影路径满足 D_2.

证明. 充分性. 当 $comp(N)$ 满足逆向强制一致性规约 $S_B(D_1, D_2)$ 的时候，由可满足性定义和可达图 G 的构造可知，任取 G 中的一条路径 ρ，形如 $t_0 \hat{} t_1 \hat{} \cdots \hat{} t_i \hat{} \cdots \hat{} t_m$，若其满足 $\sigma(t_i, t_m)$ 为 D_1 的合法投影，则在 G 中存在一条路径形如 $\rho' = t'_0 \hat{} t'_1 \hat{} \cdots \hat{} t'_j \hat{} \cdots \hat{} t'_{i-1}$（其中：$t'_0 = \overline{v}_0 \xrightarrow{a'_0} \overline{v}'_1$，$t'_{i-1} = \overline{v}'_{i-1} \xrightarrow{a'_{i-1}} \overline{v}_i$），满足 $\sigma(t'_j, t'_{i-1})$ 为 D_2 的合法投影. 考虑对 ρ 中的两段子路径 $t_0 \hat{} \cdots \hat{} t_{i-1}$ 和 $t_i \hat{} \cdots \hat{} t_m$，分别采用定理 2 中的环路替换方法，可以得到一条满足 D_1 的投影路径 $\rho_1 = t_0 \hat{} \cdots \hat{} t_i \hat{} \cdots \hat{} t_m$，其中 $\sigma(t_i, t_m)$ 仍为 D_1 的合法投影. 对 ρ' 中的两段子路径 $t'_0 \hat{} \cdots \hat{} t'_{j-1}$ 与 $t'_j \hat{} \cdots \hat{} t'_{i-1}$ 同样也作定理 2 中的环路替换操作，可以得到一条满足 D_2 的投影路径 $\rho'_1 = t'_0 \hat{} \cdots \hat{} t'_j \hat{} \cdots \hat{} t'_{i-1}$，其中 $\sigma(t'_j, t'_{i-1})$ 仍为 D_2 的合法投影. 注意到 t'_{i-1} 与 t_i 可以通过共同的节点 $\overline{v}_i$ 连接起来，因此，组合路径 $t'_0 \hat{} \cdots \hat{} t'_j \hat{} \cdots \hat{} t'_{i-1} \hat{} t_i \hat{} \cdots \hat{} t_m$ 就是集合 $\Delta(\rho_1)$ 中的一条逆向扩展投影路径. 也就是说，对满足 D_1 的投影路径 ρ_1 而言，$\Delta(\rho_1)$ 中存在一条满足 D_2 的逆向扩展投影路径.

必要性. 由逆向扩展投影路径的定义以及 $comp(N)$ 与可达图 G 的构造关系直接可证. 证毕.

定理 4. $comp(N)$ 满足双向强制一致性规约 $S_D(D_1, D_2, D_3)$ 当且仅当对其可达图 G 中任一条满足 D_1, D_2 的组合投影路径 ρ 而言，集合 $\Omega(\rho)$ 中存在一条路径满足 D_3.

证明. 充分性. 当 $comp(N)$ 满足双向强制一致性规约 $S_D(D_1, D_2, D_3)$ 的时候，由可满足性定义和可达图 G 的构造可知，任取一条 G 中的路径 ρ，形如 $t_0 \hat{} t_1 \hat{} \cdots \hat{} t_k \hat{} \cdots \hat{} t_l \hat{} \cdots \hat{} t_m \hat{} \cdots \hat{} t_n$，其中：(1) $t_{l+1} \hat{} \cdots \hat{} t_{m-1}$ 中的每一个 t_i ($0 < i < m$) 都不相同；(2) $\sigma(t_k, t_l)$ 是 D_1 的合法投影；(3) $\sigma(t_{m+1}, t_n)$ 是 D_2 的合法投影，那么，ρ 满足当 $t_{l+1} \hat{} \cdots \hat{} t_{m-1}$ 中不存在一个子路径 $t_i \hat{} \cdots \hat{} t_j$ ($l+1 \leq i, j \leq m-1$) 的 $\sigma(t_i, t_j)$ 为 D_1 或者 D_2 的投影时，则在 G 中存在一个子路径 $t'_{l+1} \hat{} t'_{l+2} \hat{} \cdots \hat{} t'_{m-1}$（其中：$t'_{l+1} = \overline{v}_{l+1} \xrightarrow{a'_{l+1}} \overline{v}'_{l+2}$，$t'_{m-1} = \overline{v}'_{m-1} \xrightarrow{a'_{m-1}} \overline{v}_m$）满足 $\sigma(t'_{l+1}, t'_{m-1})$ 是 D_3 的合法投影. 现在我们考虑将三个路径片段 $t_0 \hat{} \cdots \hat{} t_{k-1}$，$t_k \hat{} \cdots \hat{} t_l$ 和 $t_m \hat{} \cdots \hat{} t_n$ 分别采用定理 2 中的环路替换方法，可以构造出一个满足 D_1, D_2 的组合投影路径 $\rho' = t_0 \hat{} \cdots \hat{} t_k \hat{} \cdots \hat{} t_l \hat{} \cdots \hat{} t_m \hat{} \cdots \hat{} t_n$. 然后，我们对子路径 $t'_{l+1} \hat{} t'_{l+2} \hat{} \cdots \hat{} t'_{m-1}$ 也作类似的环路替换操作. 最后我们构造一条路径：$\rho'' = t_0 \hat{} \cdots \hat{} t_k \hat{} \cdots \hat{} t_l \hat{} t'_{l+1} \hat{} t'_{l+2} \hat{} \cdots \hat{} t'_{m-1} \hat{} t_m \hat{} \cdots \hat{} t_n$，由组合投影路径的定义可知，$\rho''$ 是 $\Omega(\rho')$ 中的一条组合投影路径，且满足 D_3，即对于 G 中的任一条满足 D_1, D_2 的组合投影路径 ρ' 而言，$\Omega(\rho')$ 中存在一条路径满足 D_3.

必要性. 由双向扩展投影路径定义以及 $comp(N)$ 与可达图 G 的构造关系直接可证. 证毕.

HU Jun, born in 1973, Ph. D. candidate. His research interests include software engineering, formal methods, component-based software designs, embedded software modeling and verification.

YU Xiao-Feng, born in 1975, Ph. D. candidate. His research interests include software engineering, model-driven transformation and verification, Web-service.

ZHANG Yan, born in 1974, Ph. D. candidate. His research interests include software engineering, formal methods, service-oriented computation, software measurement.

WANG Lin-Zhang, born in 1973, Ph. D.. His research interests include software engineering, model-driven software testing, software measurement.

LI Xuan-Dong, born in 1963, professor, Ph. D. supervisor. His research interests include software engineering, formal methods, software verificiation.

ZHENG Guo-Liang, born in 1936, professor, Ph. D. supervisor. His research interests include software engineering, object-oriented design methodology.

Background

This work is supported by the National Natural Science Foundation of China under grant No. 60425204: "Software Engineering Methodology", No. 60233020: "Testing and Verification of Real-Time Software Systems", and No. 60573085: "Research on UML Model Transformation and Tools Based MDA"; National Basic Research Program of China (973 Program) under grant No. 2002CB312001: "Research on Formal Theory and Methodology of Internetware"; and also supported by National Natural Science Foundation of Jiangsu Province under grant No. BK2004080: "Timing Analysis of Scenario-Based Specifications".

The intended purpose of those projects is to explore basic laws and theories for the services-oriented software systems and component-based software systems, and to investigate the modeling and verification techniques which are automatic and scalable. The research group has done a lot of research works in the past few years in this field, and have dozens of papers published in the international conferences and national journals.

The contribution of this paper is as following: Since the scenario-based specification and modeling techniques have been considered as the basic activities in modern software development, the paper mainly focuses on how to establish the effective verification techniques for the problem of checking the component-based software system designs for scenario-based specifications described by the UML sequence diagrams.