Software Engineering Group
Department of Computer Science
Nanjing University
http://seg.nju.edu.cn

Technical Report No. NJU-SEG-2014-CJ-005

2014-CJ-005

基于博弈论的开放环境下场景规约监控语义

张鹏程，李宣东，李雯睿

《中国科学》（信息科学）2014

中国科学: 信息科学 2014 年 第 44 卷 第 2 期: 263–283
www.scichina.com info.scichina.com
《中国科学》杂志社
SCIENCE CHINA PRESS
论 文

基于博弈论的开放环境下场景规约监控语义

张鹏程[①②*], 李宣东[①], 李雯睿[③]

① 南京大学计算机软件新技术国家重点实验室, 南京 210093
② 河海大学计算机与信息学院, 南京 211100
③ 南京晓庄学院数学与信息技术学院, 南京 211171
* 通信作者. E-mail: pchzhang@nju.edu.cn

收稿日期: 2012–09–20; 接受日期: 2013–08–12
国家自然科学基金 (批准号: 61202097, 61202136, 91318301)、国家高技术研究发展计划 (批准号: 2012AA011205)、中国博士后基金 (批准号: 2012T50489, 2011M500897) 和教育部博士点专项基金 (批准号: 20120094120009) 资助项目

摘要 在开放环境中, 环境和系统本身行为的改变可能使得软件系统的实现不再满足原来规约, 从而最终导致软件失效的发生. 运行时监控是一种轻量级的形式化动态验证技术, 已成为开放环境下检测软件失效的基本手段. 针对基于场景的规约属性序列图, 从博弈论的角度定义其多值监控语义: 满足、无限可控、系统有限可控、系统紧急可控、环境有限可控、环境紧急可控和违例. 通过多值监控语义的定义, 监控器能够根据当前轨迹尽可能早地检测到系统失效或异常, 并提供足够信息为失效的预防和恢复服务. 实例研究表明了属性序列图多值监控语义的实用价值, 并显示了其广泛的应用前景.

关键词 开放环境 场景规约 属性序列图 多值监控语义 博弈结构

1 引言

软件是对现实世界中问题空间与解空间的具体描述, 是客观事物的一种反映, 由于现实世界是不断变化的, 因此, 变化性是软件的基本属性 [1]. 如今软件所处的环境已从静态、封闭和可控的环境演化为动态、开放和不可控的互联网环境. 这种开放性和动态性使得客户需求与计算环境更加频繁地变化, 导致软件的变化性和复杂性进一步增加 [2,3]. 在开放环境中, 环境的改变和系统本身的改变都可能会使得软件系统的实现不再满足原来规约, 从而最终导致软件失效 (software failure, SF) 的发生. 一般而言, 软件失效是指软件运行时产生的一种不期望或不可接受的外部行为. 所以目前软件系统所面临亟待解决的问题之一是在开放和动态演化环境下如何避免软件失效的发生. 设计阶段的验证技术 (如测试和模型检验技术 [4]) 能够保证在封闭或特定环境下软件行为的正确性, 但未完全考虑其不断演化的运行时环境和实际的执行状态, 不能检测开放环境下软件的失效.

监控技术是一种轻量级的形式化动态验证技术, 已成为开放环境下检测软件失效的基本方法 [5,6]. 其通过运行时对系统进行适当插桩 (instrument) 收集运行时行为数据, 并不断地检测软件系统的实际行为与原来规约行为是否一致, 当检测到失效时能够采取适当措施, 从而使得系统从失效中恢复. 为了能够正确地使用监控技术, 必须精确定义现有规约语言的监控语义. 一些研究工作已定义传统逻辑:

引用格式: 张鹏程, 李宣东, 李雯睿. 基于博弈论的开放环境下场景规约监控语义. 中国科学: 信息科学, 2014, 44: 263–283, doi: 10.1360/112012-562

如线性时序逻辑 (linear temporal logic, LTL) 的监控语义 [7,8]. 基于场景的规约用直观、可视的形式描述系统各部件之间的交互活动, 在软件开发过程中扮演了越来越重要的角色. 本文探讨如何定义基于场景规约的监控语义.

当前的场景规约包括消息序列图 (message sequence chart, MSC)[9]、UML 序列图 (sequence diagram, SD)[10]、属性序列图 (property sequence chart, PSC)[11,12]、活性序列图 (live sequence chart, LSC)[13] 和模态序列图 (modal sequence diagram, MSD)[14,15] 等. 其中 MSC 是由国际电信联盟提倡的, 已在业界得到广泛应用, 但其有限的表达能力只能描述可能发生的场景, 不能描述系统必须满足的场景. 于是, 有研究者提出了 LSC, 它是 MSC 的模态扩展, 能够表示强制场景的发生，指出活性是指好的事情最终会发生. 此后, UML 2.0 序列图和 MSD 吸收了 MSC 和 LSC 的众多优点, 如引入结构化操作符, 并借鉴 LSC 中 universal 模态的思想, 引入新操作符 assert 和 negate 来分别表示活性和安全性.

本文拟采用的 PSC 是由 UML 2.0 序列图的子集扩展而来的, 以表示并发执行的构件之间交互行为应满足的时序属性. 与传统的时序逻辑相比, PSC 为用户提供了完全图形化的前端, 并使用规约模式 [16] 度量其表达能力, 在理解简单和表达力之间找到了一种平衡. 先前工作已经定义了 PSC 基于 Büchi 自动机的操作语义和基于无穷轨迹的指称语义 [11,12], 但只能用于设计阶段验证完全模型是否满足用户期望的属性, 不能保证在开放环境下属性仍然正确. 在开放环境下对软件系统行为进行监控, 将面临如下两个重要问题: ①运行时 PSC 接受的是有穷轨迹, 如何针对有穷轨迹重新定义其接受条件 ②采用 PSC 进行监控的目的是能够检测到失效, 那么如何根据当前轨迹尽可能早地检测到失效, 并期望提供足够信息为失效的预防和恢复服务.

针对以上两个问题, 本文主要贡献在于提出一种开放环境下基于场景规约 PSC^{Mon}(property sequence chart for monitoring) 的监控语义. 为了解决问题①, 定义了运行时接受有限轨迹的 PSC^{Mon} 形式语义. 为了解决问题②, 将博弈论思想成功应用到场景规约 PSC^{Mon} 的监控语义中, 基于博弈论定义了 PSC^{Mon} 的多值语义, 从而提供足够的信息帮助系统失效的预防和恢复措施.

本文第 2 节简要介绍用到的预备知识: 监控技术的基本思路和场景规约 PSC; 第 3 节定义一种场景规约 PSC^{Mon} 的形式语法; 第 4 节基于博弈论定义了 PSC^{Mon} 的操作语义和指称语义; 第 5 节进行实例研究; 第 6 节是与相关工作的比较; 第 7 节是结论和进一步工作的设想.

2 预备知识

2.1 监控技术简介

监控技术是运行时分析系统行为是否满足或背离给定的属性. 一般的监控器 (monitor) 是由观察器 (observer) 和分析器 (analyzer) 组成 [5]. 观察器是一个程序片段集合, 被插入到目标系统中适当位置, 以记录待监控系统的状态改变, 并收集与属性相关的数据. 分析器是检测收集的数据与给定属性是否一致. 监控器的一般过程描述如下: 首先, 从需求中抽取出待监控的属性, 用某种规约表示, 并由规约自动生成监控代码. 其次, 在实现一个观察器之后, 需确定初始化监控代码的插桩点, 利用工具将监控代码自动插入到执行的目标系统中. 接着, 观察器收集与给定属性相关的数据, 并将数据传递给一个分析器, 由分析器检测收集的数据是否与给定属性一致. 最后, 如果分析结果显示运行时数据与待验证的属性相背离时, 监控器就会调用事件处理器 (event handler) 来诊断失效, 提供诊断信息给用户, 以辅助用户理解失效发生的原因, 并帮助系统从失效中恢复, 将系统直接引入到一个正确状态, 或

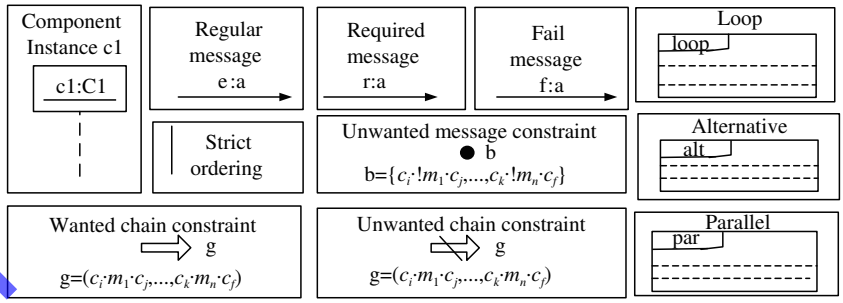

图 1 PSC 的图形化元素

Figure 1 PSC elements

仅以日志形式记录失效信息.

2.2 PSC规约简介

本节简单介绍 PSC 的图形化元素, 如图 1 所示. 首先, PSC 定义两类消息, 分别是箭头消息和限制消息, 并区分三种不同类型的箭头消息: a) 正则消息 (regular message): 由符号 “e:a” 标识. 可将正则消息视为下一条消息发生的前置条件, 系统没有强制其必须发生. 然而, 一旦该正则消息发生, 则意味着下一条消息的前置条件成立; b) 强制消息 (required message): 由符号 “r:a” 标识. 如果强制消息的前置条件满足, 则系统强制交换该类型消息, 由此验证系统的活性; c) 错误消息 (fail message): 由符号 “f:a” 标识. 如果系统交换了此类型消息, 则将会发生错误, 用来表示系统不能够交换的消息, 由此验证系统的安全性.

其次, 在 PSC 中, 还定义了箭头消息的限制, 是由强加在单个箭头消息上的限制消息组成. 箭头消息的限制分为 unwanted 消息限制和链 (chain) 限制. unwanted 消息限制表示不期望系统发生的消息集合. 链限制表示一个消息序列, 用来描述单个箭头消息和一个消息序列之间的关系. 与箭头消息不同, 链限制是将消息序列作为一个整体进行交换的. 链限制可进一步划分为 wanted 和 unwanted 链限制, wanted 链限制表示消息序列必须按照链中规定的顺序完全发生, 而 unwanted 链限制表示不期望消息序列完全交换. 另外, 根据限制和箭头消息在时序上的关系, 还可进一步将限制分为 past 限制和 future 限制, 分别表示限制在箭头消息之前或之后发生. 这样, unwanted 消息限制又可分为 past unwanted 消息限制和 future unwanted 消息限制, 而链限制则分为 past wanted, past unwanted, future wanted 和 future unwanted 链限制.

PSC 生命线有两种类型严格 (strict) 和松散 (loose). 严格类型表示一对消息之间具有严格的顺序, 即它们之间不允许其他消息发生, 反之则默认为一对消息之间是松散的顺序, 允许它们之间有其他消息发生. 此外, PSC 定义了额外三种操作符: 并发 (par)、循环 (loop) 和选择 (alt). 并发表示多条消息以不同的顺序交换; 循环表示一条或多条消息重复发生; 选择表示从多条消息中选择一条或多条进行交换.

3 $\mathrm{PSC}^{\mathrm{Mon}}$ 规约的形式语法

定义 1 (property sequence chart for monitoring, $\mathrm{PSC}^{\mathrm{Mon}}$) 一个 $\mathrm{PSC}^{\mathrm{Mon}}$ 规约是一个八元组 $\mathrm{PSC}^{\mathrm{Mon}}$=⟨ Scope, T, $\prec$, C, M, Con, timeline, Op ⟩, 其中:

• Scope={Globally, Before ..., After ..., Between ... and ..., After ... until ...}[16] 是指从规约模式的角度来说, 该属性属于何种模式;

• $T = \{t_0, t_1, \ldots, t_{n+1}\}$ 是一个有限的时间线集合;

• $\prec$ 是关于时间线中元素的一个严格偏序关系, $t_0 \prec t_1 \prec \ldots \prec t_{n+1}$ 表示时间线集合中的元素之间具有严格的先后时间顺序;

• $C = C_s \cup C_e$ 表示将有限的构件集合 C 分为系统构件集合 C_s 和环境构件集合 C_e;

• M 是系统和环境交互的消息集合, 对 M 中的消息做两种分类. 首先, 根据消息类型, $M = \text{AM} \cup \text{CM}$ 分为箭头消息 AM 和限制消息 CM, 允许 $\text{AM} \bigcap \text{CM} \neq \varnothing$ 表示某个消息既可以是箭头消息, 也可以是限制消息. 箭头消息有三种类型 type $\in \{e:, r:, f:\}$: 正则消息 (e:)、强制消息 (r:) 和错误消息 (f:). 其次, $M = \text{Input} \cup \text{Inner} \cup \text{Output}$ 且 $\text{Input} \cap \text{Inner} \cap \text{Output} = \varnothing$, Input 表示系统构件从环境构件中接收的消息集合, Inner 表示系统内部构件之间交互的消息集合, Output 表示系统构件发送给环境构件的消息集合;

• Con=$\{\lambda, \bullet\text{b}, \Rightarrow \text{g}, \nRightarrow \text{g}\}$ 是箭头消息交换时满足的限制集合. 其中, λ 则表示对应的限制为空; $\bullet$b 表示 past unwanted 消息限制或 future unwanted 消息限制; $\Rightarrow$ g 表示 past wanted 链限制或 future wanted 链限制; $\nRightarrow$ g 表示 past unwanted 链限制或 future unwanted 链限制. 对于 unwanted 消息限制和链限制, 它们基本定义如下: $\bullet\text{b} = \{l_1, l_2, \ldots, l_n\}$, 其中, $l_i = c_j.!m_i.c_k, m_i \in \text{CM}(i = 1, \ldots, n)$, "$!m_i$" 表示 $c_j \in C$ 和 $c_k \in C$ 之间不允许交换 m_i, 即 m_i 没有发生. g=$(l_1, l_2, \ldots, l_n)$, 其中, $l_i = c_j.m_i.c_k (i = 1, \ldots, n)$, $c_j, c_k \in C$, $m_i \in \text{CM}$;

• 函数 timeline : AM $\leftrightarrow T\backslash\{t_0, t_{n+1}\}$, 表示箭头消息集合中的元素和时间线集合中的元素 (去除 t_0 和 t_{n+1}) 之间存在一一对应关系, 即对于任意箭头消息 $\text{msg}_i \in \text{AM}$, $\text{timeline}(\text{msg}_i) = t_i, 1 \leqslant i \leqslant n$;

• Op = {loose, strict, alt, par, loop} 是一个有限的操作符集合. 对于 $\forall$ op $\in$ Op, op $\rightarrow \text{timeline}(\text{msg}_i) \times \text{timeline}(\text{msg}_j)$, $1 \leqslant i < j \leqslant n$, 指时间线 t_i 和 t_j 之间的消息可能存在 loose 操作符、strict 操作符、alt 操作符、par 操作符或 loop 操作符.

为了进行运行时监控, 新的 PSC^{Mon} 的形式语法增加了属性模式的定义, 并将构件分为系统构件和环境构件, 消息分为环境输入、系统内部和系统输出消息, 并修改了相应的一致性定义.

4 基于博弈论的 PSC^{Mon} 规约多值监控语义

4.1 博弈论基本知识

定义 2 (博弈结构 (game structure))[17] 环境和系统两个参与者的博弈结构是一个八元组 $G = \langle S_s, S_e, \Sigma_s, \Sigma_e, S_0, T_s, T_e, c\rangle$, 其中:

• S_s 是系统状态的集合, S_e 是环境状态的集合;

• Σ_s 是系统控制变量集合, Σ_e 是环境控制变量集合;

• S_0 是初始状态的集合;

• T_s 和 T_e 分别表示系统和环境的转换关系. 形式上来说, $T_s : S_s \times S_e \times \Sigma_s \times S_s \times S_e$, 对于 $\forall t_s = \langle s_s, s_e, a_s, s'_s, s'_e\rangle \in T_s$ 表示系统的转换关系 t_s 在系统可控的变量 $a_s \in \Sigma_s$ 的作用下, 系统状态从 s_s 迁移到 s'_s, 环境状态从 s_e 迁移到 s'_e. 形式上来说, $T_e : S_s \times S_e \times \Sigma_e \times S_s \times S_e$, 对于任意 $t_e = \langle s_s, s_e, a_e, s'_s, s'_e\rangle \in T_e$ 表示环境的转换关系 t_e 在环境可控的变量 $a_e \in \Sigma_e$ 的作用下, 系统状态从 s_s 迁移到 s'_s, 环境状态从 s_e 迁移到 s'_e.

• $c: S_s \cup S_e \to N$, 其中 N 是接受条件的集合, c 是一个着色函数将每个状态映射为系统或环境的某个接受条件.

定义 3 (轨迹 (trace))　Σ_e 和 Σ_s 分别是环境和系统变量集合, 则轨迹的形式定义为 $((\Sigma_e)^+(\Sigma_s)^*)^\omega$ 或 $((\Sigma_e)^+(\Sigma_s)^*)^*$, 表明对于环境的每一组动作, 系统会完成一组动作完毕后再返回控制给环境, 其中 $((\Sigma_e)^+(\Sigma_s)^*)^\omega$ 表示无限轨迹, 而 $((\Sigma_e)^+(\Sigma_s)^*)^*$ 表示有限轨迹.

定义 4 (策略 (strategy))　策略是指在所有可能发生情况下参与者的一套完整行动计划. 针对系统和环境两个参与者来说, 分为系统赢的策略 S_{sys} 和环境赢的策略 S_{env}. 针对无限轨迹 ω 来说系统赢的策略 $S_{\text{sys}}| = \omega$ 指如果无限轨迹能够满足系统接受条件无限次, 环境赢的策略 $S_{\text{env}}| = \omega$ 指如果无限轨迹能够满足环境接受条件无限次. 针对有限轨迹 μ 来说, 系统赢的策略 $S_{\text{sys}}| = \mu$ 指如果有限轨迹满足系统接受条件, 则以该有限轨迹作为前缀的无限轨迹都满足系统接受条件; 环境赢的策略 $S_{\text{env}}| = \mu$ 指如果有限轨迹满足环境接受条件, 则以该有限轨迹作为前缀的无限轨迹都满足环境接受条件.

4.2　PSC$^{\text{Mon}}$ 规约的操作语义

根据博弈论给出 PSC$^{\text{Mon}}$ 规约的多值监控语义. 首先给出基于博弈结构的 PSC$^{\text{Mon}}$ 规约形式定义, 接着定义操作语义规则将 PSC$^{\text{Mon}}$ 转化为对应的博弈结构. 其中, 规则分为基本语义规则和组合语义规则两类. 基本规则讨论如何将基本的 PSC$^{\text{Mon}}$ 规约转化为 G; 而组合规则讨论如何将基本 PSC$^{\text{Mon}}$ 生成的 G 粘合成一个复杂的 G, 或者当 PSC$^{\text{Mon}}$ 中存在复杂的结构化操作如选择 alt、并发 par、循环 loop 等的情况.

定义 5 (一个 PSC$^{\text{Mon}}$ 规约是一个博弈结构)　$G = \langle S_s, S_e, \Sigma_s, \Sigma_e, s_0, T_s, T_e, c, \text{Glue}\rangle$, 其中, S_s 是系统状态的集合, S_e 是环境状态的集合; Σ_s 是系统控制变量集合; Σ_e 是环境控制变量集合; s_0 是唯一的初始状态; T_s 和 T_e 分别表示系统和环境的转换关系; $\text{Glue} \subseteq S_s \cup S_e$ 指粘合状态的集合, 当前 G 的粘合状态 $s_{\text{glue}} \in \text{Glue}$ 用来和下一个 G 的初始状态 s_0 进行粘合而生成更复杂的 G; $c: S_s \cup S_e \to N$ 是一个着色函数表示接受条件, 为每个状态定义一个特定的着色标记, 分为以下 7 种, 即 $N = \{\surd, \Diamond, \square, \blacksquare, \circ, \bullet, \times\}$, 分别做如下解释:

• 满足 ($\surd$): 系统和环境之间的行为实际执行并满足该场景;

• 无限可控 ($\Diamond$): 当前处于环境和系统无限可控的状态, 即可无限停留在该状态而不发生违例;

• 系统有限可控 ($\square$): 当前处于系统有限可控状态, 但系统只能保证在该状态停留有限步骤, 有限步骤后走错会发生违例;

• 系统紧急可控 ($\blacksquare$): 当前处于系统紧急可控状态, 系统下一步走错即导致违例;

• 环境有限可控 ($\circ$): 当前处于环境有限可控状态, 但环境只能保证在该状态停留有限步骤, 有限步骤后走错会发生违例;

• 环境紧急可控 ($\bullet$): 当前处于环境紧急可控状态, 环境下一步走错即导致违例;

• 违例 ($\times$): 系统和环境之间的行为不满足该属性, 即违例发生了.

在这几种状态着色标记中, $\surd$, $\Diamond$表示满足或无限可控, $\square$ 和 $\blacksquare$ 表示系统可控的状态, $\circ$ 和 $\bullet$则表示环境可控状态, 而 $\times$ 表示违例即系统和环境都不可控的状态.

4.2.1　基于博弈结构的 PSC$^{\text{Mon}}$ 基本规则

下面定义三类箭头消息的基本语义规则. 对于每种类型消息, 只考虑单个箭头消息, 或带有 strict

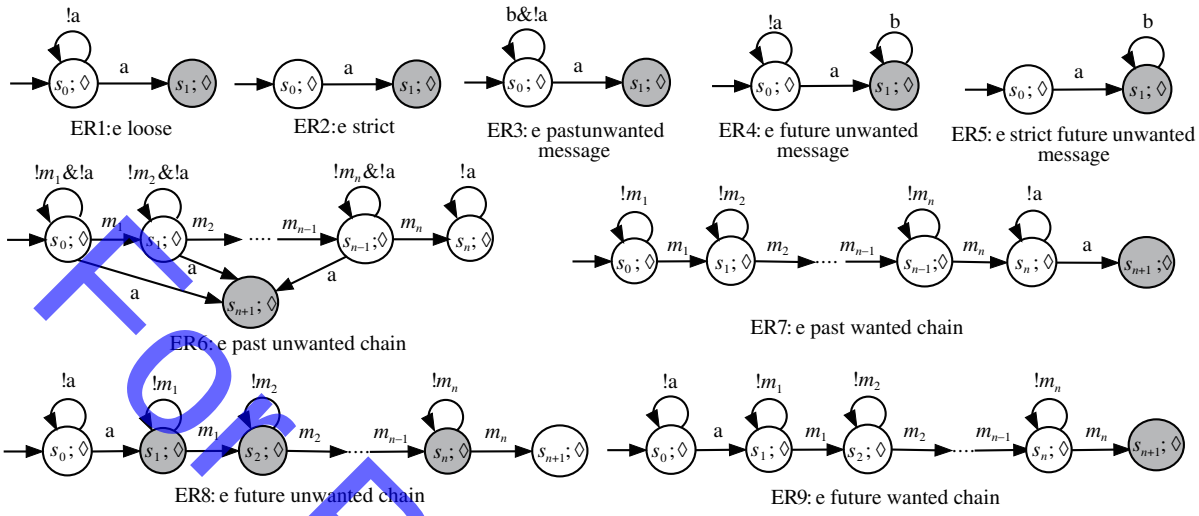

图 2　正则消息的博弈语义规则

Figure 2　Game-based rules for regular message

操作符, 或带有 unwanted 消息限制或 wanted、unwanted 链限制的情况. $\mathrm{a}^{1)} \in \Sigma_s \cup \Sigma_e$ 是交换的箭头消息, b 和 g 分别表示 unwanted 消息限制和链限制.

对于每个语义规则, 采用图形表示其生成的 G. 在生成的每个博弈结构中, 注意这里生成的 G 是一个同步结构表示, 即可用一个状态 s 来表示系统状态 s_s 和环境状态 s_e. 例如, 初始状态 s_0 既表示系统处于初始状态 s_{0s}, 又表示环境处于初始状态 s_{0e}.

(1) 正则消息的操作语义

正则消息的博弈语义规则 (regular rule, ER), 如图 2 所示. 图中灰色状态表示当前博弈结构的粘合状态. 由于正则消息可交换也可不交换, 故无论是系统或环境可交换的消息, 都处于系统或环境无限可控的状态 ($\Diamond$), 即系统或环境可无限停留在该状态而不发生违例. ER1 表示带有 loose 操作符的正则消息, 如果正则消息 e:a 不发生, 则停留在初始状态 s_0; 如果 e:a 发生了, 则从初始状态 s_0 迁移到无限可控粘合状态 s_1. ER2 表示带有 strict 操作符的正则消息, 在初始状态 s_0 时, 只有 e:a 发生且没有其他消息在 e:a 之前发生, 则迁移到无限可控粘合状态 s_1.

ER3~ER5 表示带有 unwanted 消息限制 $\mathrm{b}=\{m_1, m_2, \ldots, m_n\}$[2)]的正则消息. ER3 表示带有 past unwanted 消息限制的正则消息, 只有在 b 中消息未发生且 e:a 发生的前提下, 则从初始状态 s_0 迁移到无限可控粘合状态 s_1. ER4 表示带有 future unwanted 消息限制的正则消息, 故只有在 e:a 发生且消息限制 b 中消息未发生的前提下, 则到达无限可控粘合状态 s_1. ER5 表示带有 strict 操作符和 future unwanted 消息限制的正则消息, 故与 ER4 相类似, 只有在 e:a 发生之前不发生其他消息.

ER6~ER9 表示带有链限制 $\mathrm{g}=(m_1, m_2, \ldots, m_n)$ 的正则消息. ER6 表示带有 past unwanted 链限制的正则消息, 如果链限制 g 未完全发生且 e:a 发生, 则到达无限可控的粘合状态 s_{n+1}. ER7 表示带有 past wanted 链限制的正则消息, 如果链限制 g 完全发生且 e:a 发生, 则到达无限可控的粘合状态 s_{n+1}. ER8 表示带有 future unwanted 链限制的正则消息, 如果 e:a 发生且链限制 g 未完全发生, 则到

1) 是消息标签 $c_j.a.c_k$ 的缩写, 余文的定义都将采用这种形式.

2) 同样, m_i 是消息标签 $c_j.m_i.c_k$ 的缩写, $!m_i$ 是消息标签 $c_j.!m_i.c_k$ 的缩写, 余文的定义都将采用这种形式.

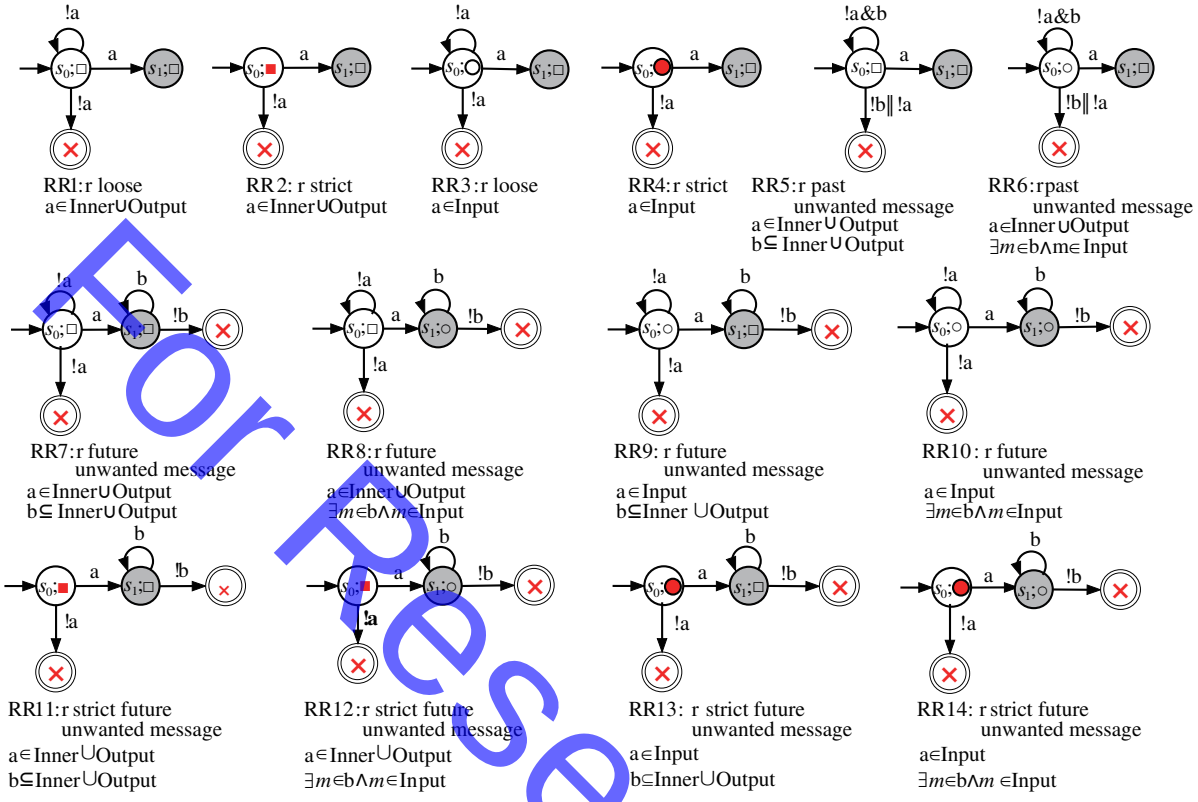

图 3 (网络版彩图) 强制消息的博弈语义规则: 部分 1

Figure 3 (Color online) Game-based rules for required message: Part 1

达无限可控的粘合状态 $s_1, s_2, \ldots, s_n$. ER9 表示带有 future wanted 链限制的正则消息, 如果 e:a 发生且链限制 g 完全发生, 则到达无限可控的粘合状态 s_{n+1}.

(2) 强制消息的操作语义

强制消息的博弈语义规则 (required rule, RR) 如图 3 和图 4 所示. RR1 表示带有 loose 操作符的强制消息, 当 r:a 是系统可控消息时 (a ∈ Inner ∪ Output), 初始状态 s_0 处于系统有限可控状态 (□), 如果强制消息 r:a 在有限步骤内发生, 则迁移到系统有限可控粘合状态 s_1, 反之到达违例状态 (×). 从技术上讲, 可使用一个足够大的常量来区分 s_0 的自转换和到达违例状态的转换. RR2 表示带有 strict 操作符的强制消息, 当 r:a 是系统可控消息时, 由于带有 strict 操作符, r:a 前不允许有其他消息发生, 所以初始状态 s_0 处于系统紧急可控状态 (■), 一旦系统下一步走错即导致违例发生, 如果强制消息 r:a 发生, 则迁移到系统有限可控粘合状态 s_1; 而如果 r:a 未发生而发生其他消息, 则到达违例状态. RR3 表示带有 loose 操作符的强制消息, 当 r:a 是环境可控消息时 (a ∈ Input), 初始状态 s_0 处于环境有限可控状态 (○), 如果期望的环境输入消息 r:a 发生, 则迁移到系统有限可控粘合状态 s_1, 反之到达违例状态. RR4 表示带有 strict 操作符的强制消息, 当 r:a 是环境可控消息时, 由于带有 strict 操作, 所以初始状态 s_0 处于环境紧急可控状态 (●), 如果强制消息 r:a 发生, 则迁移到系统有限可控粘合状态 s_1; 反之下一步期望的环境消息未发生即导致违例发生.

RR5~RR14 表示带有 unwanted 消息限制的强制消息. 其中, RR5 和 RR6 表示带有 past unwanted 消息限制的强制消息. 在 RR5 中, 当 r:a 和消息限制 b 中消息都是系统可控消息时, 初始状态 s_0 处于

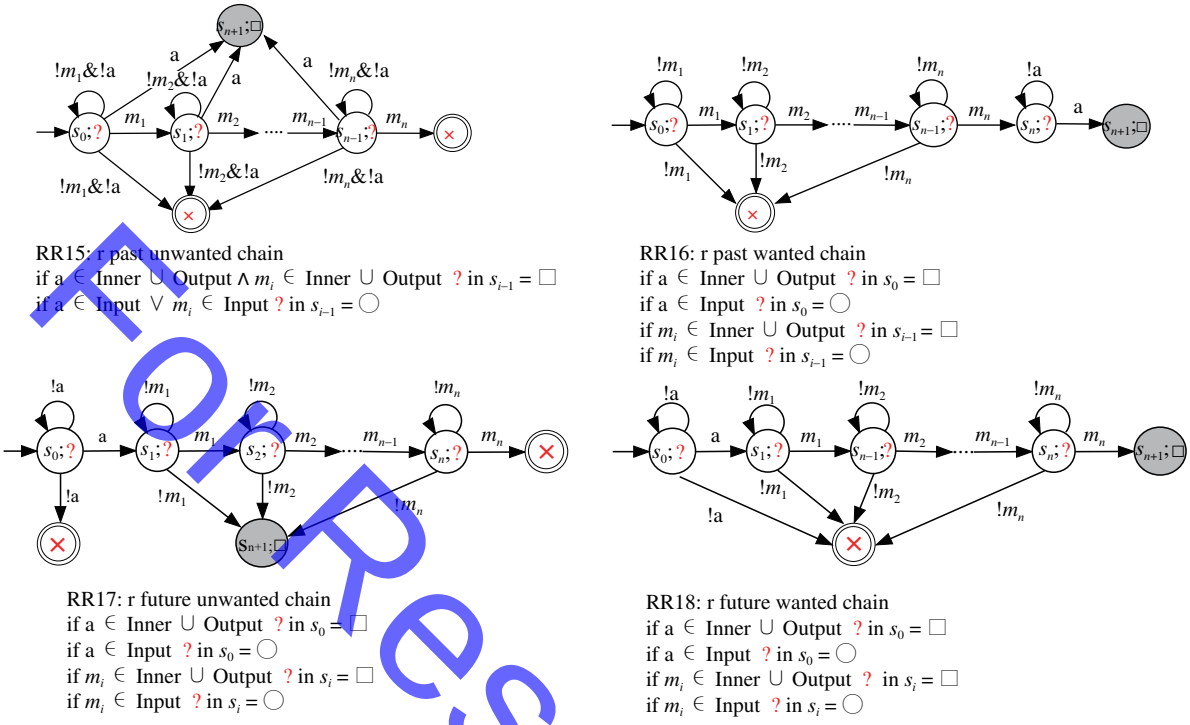

图 4 (网络版彩图) 强制消息的博弈语义规则: 部分 2

Figure 4 (Color online) Game-based rules for required message: Part 2

系统有限可控状态, 如果 b 中消息未发生且强制消息 r:a 发生, 则迁移到系统有限可控的粘合状态 s_1; 反之, 如果在有限步骤内 b 中消息发生 (由 !b 来表示) 或最终 r:a 未发生, 则到达违例状态. 在 RR6 中, 当 r:a 是系统可控消息和 b 中消息是环境可控消息时, 初始状态 s_0 处于环境可控状态 (○), 如果 b 中消息未发生且 r:a 发生, 则迁移到系统有限可控粘合状态 s_1; 反之, 如果在有限步骤内 b 中消息发生, 则到达违例状态.

RR7~RR10 表示带有 future unwanted 消息限制的强制消息. 在 RR7 中, 当 r:a 和 b 中消息都是系统可控消息时, 初始状态 s_0 处于系统可控状态, 如果 r:a 发生且 b 中消息未发生, 则迁移到系统有限可控的粘合状态 s_1; 反之, 如果在有限步骤内 r:a 未发生或在粘合状态时 b 中消息发生, 则到达违例状态. 在 RR8 中, 当 r:a 是系统可控消息和 b 中消息是环境可控消息时, 初始状态 s_0 处于系统可控状态, 如果强制消息 r:a 发生且 b 中消息未发生, 则迁移到环境有限可控粘合状态 s_1; 反之, 如果在有限步骤内 r:a 未发生或者在粘合状态 s_1 时 b 中消息发生, 则到达违例状态. 在 RR9 中, 当 r:a 是环境可控消息和 b 中消息是系统可控消息时, 初始状态 s_0 处于环境有限可控状态, 如果 r:a 发生且 b 中消息未发生, 则迁移到系统有限可控粘合状态 s_1; 反之, 如果在有限步骤内 r:a 未发生或者在粘合状态 s_1 时 b 中消息发生, 则到达违例状态. 在 RR10 中, 当 r:a 和 b 中消息都是环境可控消息时, 如果 r:a 发生且 b 中消息未发生, 则迁移到环境有限可控粘合状态 s_1, 反之, 到达违例状态.

RR11~RR14 表示带有 strict 操作符和 future unwanted 消息限制的强制消息. 在 RR11 中, 当 r:a 和 b 中消息都是系统可控消息时, 初始状态 s_0 处于系统紧急可控状态, 如果 r:a 发生且 b 中消息未发生, 则迁移到系统有限可控粘合状态 s_1, 反之则到达违例状态. 在 RR12 中, 当 r:a 是系统可控消息和 b 中消息是环境可控消息时, 初始状态 s_0 处于系统紧急可控状态, 如果 r:a 发生且 b 中消息未发生,

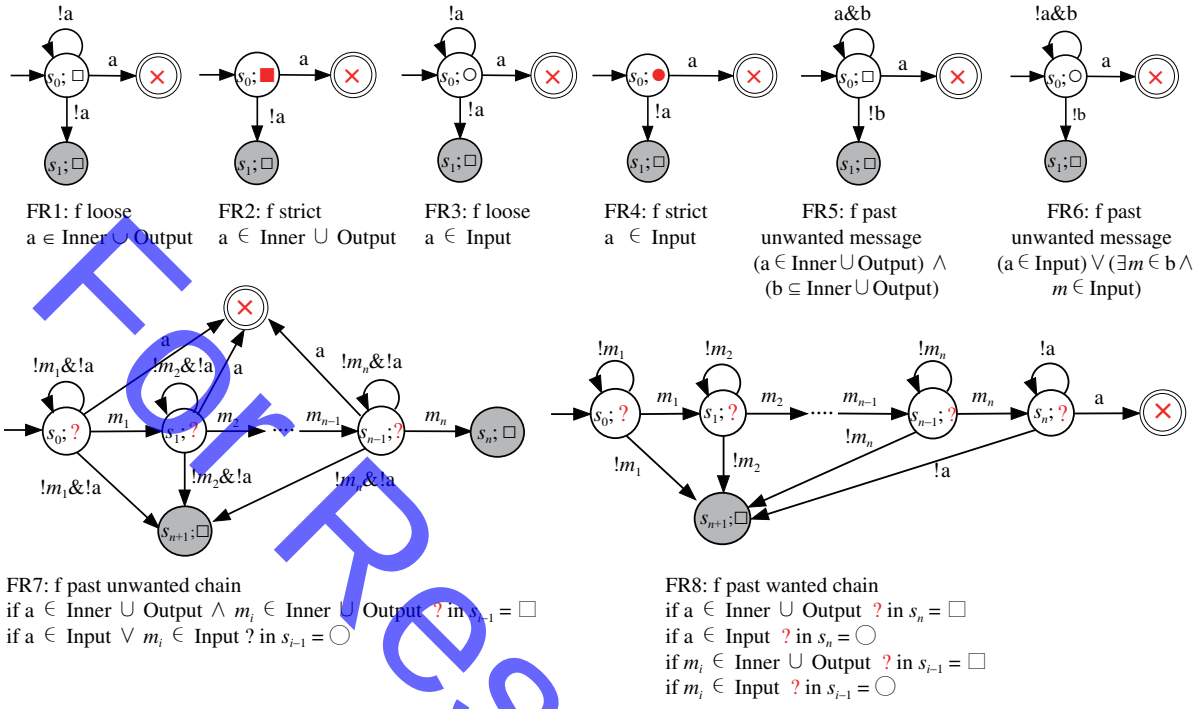

图 5　(网络版彩图) 错误消息的博弈语义规则

Figure 5　(Color online) Game-based rules for fail messages

则迁移到环境可控的粘合状态 s_1, 反之则到达违例状态. 在 RR13 中, 当 r:a 是环境可控消息和 b 中消息是系统可控消息时, 初始状态 s_0 处于环境紧急可控状态, 如果 r:a 发生且 b 中消息未发生, 则迁移到系统有限可控粘合状态 s_1, 反之则到达违例状态. 在 RR14 中, 当 r:a 和 b 中消息都是环境可控消息时, 初始状态 s_0 处于环境紧急可控状态, 如果 r:a 发生且 b 中消息未发生, 则迁移到环境有限可控粘合状态 s_1, 反之则到达违例状态.

RR15~RR18 表示带有链限制 g= $(m_1, m_2, \ldots, m_n)$ 的强制消息, 如图 4 所示. 图中 "?" 表示状态可能是系统有限可控 (□) 或环境有限可控 (○) 的, 在链限制中, 消息序列是作为一个整体交换的, 如果第一个消息 m_1 是系统可控消息, 则初始状态 s_0 处于系统有限可控状态; 如果 m_1 是环境可控消息, 则 s_0 处于环境有限可控状态, 链限制中其余消息 m_2 到 m_n 类似. RR15 表示带有 past unwanted 链限制的强制消息, 如果链限制 g 最终完全发生, 则到达违例状态; 反之, 如果链限制 g 未完全发生, 则到达状态 s_n, 并且这时 r:a 发生了, 则到达系统有限可控粘合状态 s_{n+1}, 反之到达违例状态. RR16 表示带有 past wanted 链限制的强制消息, 如果链限制 g 完全发生且 r:a 也发生, 则到达系统有限可控粘合状态 s_{n+1}, 反之到达违例状态. RR17 表示带有 future unwanted 链限制的强制消息, 当 r:a 是系统可控消息时, 初始状态 s_0 处于系统有限可控状态; 而当 r:a 是环境可控消息时, 初始状态 s_0 处于环境有限可控状态. 接着, 对于链限制中每个消息 m_i 情况类似, 当 m_i 是系统可控消息时, 相应 s_i 是系统有限可控状态; 当 m_i 是环境可控消息时, 相应 s_i 是环境有限可控状态. 如果链限制 g 最终未完全发生, 则到达系统有限可控粘合状态 s_{n+1}. RR18 表示带有 future wanted 链限制的强制消息, 与前面的情况相类似, 只是如果链限制 g 最终完全发生, 则到达系统有限可控粘合状态 s_{n+1}, 反之到达违例状态.

(3) 错误消息的操作语义

错误消息 (f:a) 的博弈语义规则 (fail rule, FR) 如图 5 所示. FR1 表示带有 loose 操作符的错误消息, 当 f:a 是系统可控消息时, 初始状态 s_0 处于系统有限可控状态, 如果 f:a 发生则到达违例状态, 反之则迁移到系统有限可控粘合状态 s_1. FR2 表示带有 strict 操作符的错误消息, 当 f:a 是系统可控消息时, 由于带有 strict 操作符, 所以初始状态 s_0 处于系统紧急可控状态, 如果 f:a 发生则到达违例状态, 反之则迁移到系统有限可控粘合状态 s_1. FR3 表示带有 loose 操作符的错误消息, 当 f:a 是环境可控消息时, 初始状态 s_0 处于环境有限可控状态, 如果 f:a 发生则到达违例状态; 反之, 如果 f:a 未发生则迁移到系统有限可控粘合状态 s_1. FR4 表示带有 strict 操作符的错误消息, 当 f:a 是环境可控消息时, 初始状态 s_0 处于环境紧急可控状态, 如果 f:a 发生则到达违例状态; 反之, 如果 f:a 未发生则迁移到系统有限可控粘合状态 s_1.

FR5 和 FR6 表示带有 past unwanted 消息限制的错误消息. 在 FR5 中, 当 f:a 和 b 中消息都是系统可控消息时, 初始状态 s_0 处于系统有限可控状态, 如果在 b 成立的前提下 f:a 发生则到达违例状态, 如果 b 不成立则到达系统有限可控粘合状态 s_1. 在 FR6 中, 当 f:a 是系统可控消息, 或者 b 中消息也是环境可控消息时, 初始状态 s_0 处于环境有限可控状态, 如果在 b 成立的前提下 f:a 发生则到达违例状态; 反之, 如果 b 不成立则到系统有限可控粘合状态.

FR7 和 FR8 表示带有 past unwanted 链限制的错误消息. 在 FR7 中, 当 f:a 是系统可控消息时, 则 s_n 处于系统有限可控状态; 而当 f:a 是环境可控消息时, s_n 处于环境有限可控状态. 而对于链限制中的任意 m_i, 当其是系统可控消息, s_{i-1} 处于系统有限可控状态, 否则 s_{i-1} 处于环境有限可控状态. 如果链限制完全发生而 f:a 未发生, 则到达系统可控的粘合状态 s_{n+1}. 在 FR8 中, 其他情况与 FR7 相类似, 只是在链限制完全发生后, f:a 消息发生了, 则到达违例状态, 反之, 到达系统可控的粘合状态 s_{n+1}.

4.2.2 基于博弈结构的 PSC$^{\text{Mon}}$ 组合规则

通过使用基本规则可以将基本消息转换为相应的基本博弈结构. 然而, 较复杂的属性是由带有操作符 Alt、Par 或 Loop 的消息构成, 需用组合规则将 PSC$^{\text{Mon}}$ 规约转换为相应的博弈结构.

通常情况下, 一个属性规约可由多个箭头消息以顺序的方式组成. 这时, 可通过 Merge 组合将这些基本箭头消息以顺序的方式组合起来, 下面给出 Merge 组合的形式操作语义.

定义 6 (Merge 组合) 给定两个博弈结构:$G_i = \langle S_s, S_e, \Sigma_s, \Sigma_e, s_0, T_s, T_e, c, \text{Glue}\rangle$, 其中 $i \in \{1,2\}$, 一个新生成的博弈结构 $G = \langle S_s, S_e, \Sigma_s, \Sigma_e, s_0, T_s, T_e, c, \text{Glue}\rangle$ 是 G_1 和 G_2 的顺序组合, 记为 $G = G_1 \cdot G_2$. 定义新生成 G 的各个元素如下:

- $G.S_s = G_1.S_s \cup G_2.S_s$;
- $G.S_e = G_1.S_e \cup G_2.S_e$;
- $G.\Sigma_s = G_1.\Sigma_s \cup G_2.\Sigma_s \cup \{\varepsilon\}$;
- $G.\Sigma_e = G_1.\Sigma_e \cup G_2.\Sigma_e$;
- $G.s_0 = G_1.s_0$;
- $G.\text{Glue} = G_2.\text{Glue}$;
- 对于转换 $G.T_s$ 和 $G.T_e$, 这里考虑系统和环境是一个同步结构, 故 $G.T_e$ 和 $G.T_s$ 的处理方式相似. 仅以 $G.T_s$ 为例. 对于 $\forall\, t = \langle s_s, s_e, a_s, s'_s, s'_e\rangle \in G.T_s$, 可定义如下: $G.T_s = G_1.T_s \cup G_2.T_s \cup T_{\text{glue}}$, 其中 $T_{\text{glue}} = \{t | t.s_s = t.s_e \in G_1.\text{Glue} \wedge t.s'_s = t.s'_e = G_2.s_0 \wedge t.a_s = \varepsilon\}$, 即 T_{glue} 表示从 G_1 的粘合状态到

G_2 的初始状态增加了一个空转换 (ε);

• 对于着色函数 c, 分以下几种情况进行讨论:

– $s \in G_1.S_s \cup G_1.S_e \wedge s\neg \in G_1.\text{Glue} \Rightarrow G.c(s) = G_1.c(s)$. 即对于属于 G_1 但不是 G_1 的粘合状态, 则这些状态在新生成 G 中状态的着色函数与 G_1 中状态的着色函数相同;

– $s \in G_1.\text{Glue} => G.c(s) = G_2.c(G_2.s_0)$. 对于 G_1 的粘合状态, 这些状态在新生成 G 中状态的着色函数与 G_2 中初始状态的着色函数相同;

– $s \in G_2.S_s \cup G_2.S_e \Rightarrow G.c(s) = G_2.c(s)$. 对于属于 G_2 的状态, 则这些状态在新生成 G 中状态的着色函数与 G_2 中状态的着色函数相同.

一个属性规约也可由多个箭头消息以选择组合的方式组成. 这时, 可通过 Alternative 组合将这些基本箭头消息以选择的方式组合起来, 下面给出 Alternative 组合的形式操作语义.

定义 7 (Alternative 组合) 给定两个博弈结构: $G_i = \langle S_s, S_e, \Sigma_s, \Sigma_e, s_0, T_s, T_e, c, \text{Glue}\rangle$, 其中 $i \in \{1, 2\}$, 一个新生成 $G = \langle S_s, S_e, \Sigma_s, \Sigma_e, s_0, T_s, T_e, c, \text{Glue}\rangle$ 是 G_1 和 G_2 的选择组合, 记为 $G = G_1 \otimes G_2$. 定义新生成 G 的各个元素如下:

• $G.S_s = G_1.S_s \cup G_2.S_s \cup \{s_0\}$;

• $G.S_e = G_1.S_e \cup G_2.S_e$;

• $G.\Sigma_s = G_1.\Sigma_s \cup G_2.\Sigma_s$;

• $G.\Sigma_e = G_1.\Sigma_e \cup G_2.\Sigma_e$;

• $G.s_0 = s_0$;

• $G.T_s = G_1.T_s \cup G_2.T_s \cup \{\langle s_{0s}, s_{0e}, \varepsilon, G_1.s_{0s}, G_1.s_{0e}\rangle\} \cup \{\langle s_{0s}, s_{0e}, \varepsilon, G_2.s_{0s}, G_2.s_{0e}\rangle\}$;

• $G.T_e = G_1.T_e \cup G_2.T_e \cup \{\langle s_{0s}, s_{0e}, \varepsilon, G_1.s_{0s}, G_1.s_{0e}\rangle\} \cup \{\langle s_{0s}, s_{0e}, \varepsilon, G_2.s_{0s}, G_2.s_{0e}\rangle\}$;

• $G.c$ 包括如两个方面:

– $s \in G_1 \Rightarrow G.c(s) = G_1.c(s)$. 对于作色函数属于 G_1, 它必属于 G;

– $s \in G_2 \Rightarrow G.c(s) = G_2.c(s)$. 对于作色函数属于 G_2, 它必属于 G.

• $G.\text{Glue} = G_1.\text{Glue} \cup G_2.\text{Glue}$.

Merge 和 Alternative 组合都产生了空转换, 可以利用现有的算法去除相应的空转换从而生成等价的博弈结构. 根据基本的 Merge 组合和 Alternative 组合, 可定义 PSC$^{\text{Mon}}$ 中 Alt、Par 和 Loop 操作符的博弈语义规则.

定义 8 (Alt 操作符) $\text{Alt}(\omega^{i_1,j_1}, \omega^{i_2,j_2}, \ldots, \omega^{i_r,j_r}, r)$ 指 PSC$^{\text{Mon}}$ 中的 Alt 操作符, 含有 r 个选择部分. 首先根据定义 7 的 Merge 组合生成 r 个基本序列的博弈结构, 再根据定义 8 的 Alternative 组合生成 r 个序列的博弈结构的 Alt 组合.

定义 9 (Par 操作符) $\text{Par}(\omega^{i_1,j_1}, \omega^{i_2,j_2}, \ldots, \omega^{i_r,j_r}, r)$ 指 PSC$^{\text{Mon}}$ 中的 Par 操作符, 含有 r 个并发部分. 首先计算可能生成的并发序列个数 $\text{num(Par)} = (k_1 + k_2 + \ldots + k_r)!/k_1!k_2! \ldots k_r!$, 其中 $k_1, k_2, \ldots, k_r$ 是相应的 r 部分消息. 再对每个并发序列根据定义 7 的 Merge 组合生成 num(Par) 个基本序列的博弈结构, 再根据定义 8 的 Alternative 组合生成 num(Par) 个基本序列的博弈结构.

定义 10 (Loop 操作符) $\text{Loop}(\omega^{i,j}, m, n)$, 指 PSC$^{\text{Mon}}$ 中的 Loop 操作符, 最小重复次数为 m, 最大重复次数为 n. 首先根据 Merge 组合生成序列 $\omega^{i,j}$ 的博弈结构, 可复用 Alternative 组合相应的 $(n - m + 1)$ 个博弈结构的组合, 其中第一个组合是循环体重复 m 次. 最后一个组合是循环体重复 n 次.

4.3 PSC$^{\text{Mon}}$ 的指称语义

上一节给出了基于博弈结构的 PSC$^{\text{Mon}}$ 操作语义, 本节给出基于轨迹的 PSC$^{\text{Mon}}$ 指称语义. 从系

统和环境之间的博弈角度, 指定各个接受条件所接受的不同轨迹集合, 这样, 指称语义不需要将 PSC^{Mon} 转化为博弈结构, 从而提供了另一个角度对 PSC^{Mon} 的理解.

下面首先介绍指称语义定义中用到的相关符号. 箭头消息 AM 的指称语义可定义如下: $[[(\langle \text{type}, a, c, c', p, f\rangle, \text{op})]]^{\text{trace}}$, 其中 type $\in \{e:, r:, f:\}$ 是消息类型. a 由发送构件 c 和接受构件 c' 交换的消息. $p, f \in \{\lambda, \bullet b, \Rightarrow \text{g}, \nRightarrow \text{g}\}$, p 指 past 限制, f 指 future 限制, λ 指空限制, $\bullet$ b 指 unwanted 消息限制, $\Rightarrow$ g 指 wanted 链限制和 $\nRightarrow$ g 指 unwanted 链限制. b= $\{m_1, m_2, ..., m_n\}$, g= $(m_1, m_2, ..., m_n)$. op $\in$ { Loose, Strict}, 其中 Loose 指松散操作符, 而 Strict 指严格操作符. L 是整个系统与环境之间交互的所有消息集合; $\alpha, \beta, \gamma \in L^*$ 系统与环境之间交互的有限轨迹; $\alpha, \beta, \gamma \in L^\omega$ 指系统与环境之间交互的无限轨迹, 由于本文是用来监控, 从技术上来讲, 当有限轨迹到达一定大的界限时称为无限轨迹. 指称语义由 $[[\text{PSC}]]^{\text{trace}}$ 来表示. 其中 PSC 可指单个消息或一系列消息表示的 PSC^{Mon} 规约; trace $\in$ { VT, IC, SFC, SUC, EFC, EUC, IVT}, 具体解释如下:

- $[[\ \text{PSC}\ ]]^{\text{VT}}$ 指该 PSC^{Mon} 接受的正确轨迹集合 (validated traces, VT);
- $[[\ \text{PSC}\ ]]^{\text{IC}}$ 指该 PSC^{Mon} 接受的无限可控轨迹集合 (infinite controllable traces, IC);
- $[[\ \text{PSC}\ ]]^{\text{SFC}}$ 指该 PSC^{Mon} 接受的系统有限可控轨迹集合 (system finite controllable traces, SFC);
- $[[\ \text{PSC}\ ]]^{\text{SUC}}$ 指该 PSC^{Mon} 接受的系统紧急可控轨迹集合 (system urgent controllable traces, SUC);
- $[[\ \text{PSC}\ ]]^{\text{EFC}}$ 指该 PSC^{Mon} 接受的环境有限可控轨迹集合 (environment finite controllable traces, EFC);
- $[[\ \text{PSC}\ ]]^{\text{EUC}}$ 指该 PSC^{Mon} 接受的环境紧急可控轨迹集合 (environment urgent controllable traces, EUC);
- $[[\ \text{PSC}\ ]]^{\text{IVT}}$ 指该 PSC^{Mon} 接受的错误轨迹集合 (invalidated traces, IVT).

4.3.1 基本轨迹语义

a) 正则消息的轨迹语义: 由于正则消息是系统和环境之间可能交换的消息, 故正则消息只能接受的正确轨迹集合和无限可控轨迹集合, 即 $[[\text{PSC}]]^{\text{VT}}$ 和 $[[\text{PSC}]]^{\text{IC}}$, 其他所有轨迹语义都为空.

a.1 带有 Loose 操作符的正则消息:

$[[(\langle e:, a, c, c', \lambda, \lambda\rangle, \text{Loose})]]^{\text{VT}} = \{\alpha \cdot a | \alpha \in (L\backslash\{a\})^*\}$;

$[[(\langle e:, a, c, c', \lambda, \lambda\rangle, \text{Loose})]]^{\text{IC}} = \{\alpha | \alpha \in (L\backslash\{a\})^\omega\}$.

a.2 带有 Strict 操作符的正则消息:

$[[(\langle e:, a, c, c', \lambda, \lambda\rangle, \text{Strict})]]^{\text{VT}} = \{a\}$;

$[[(\langle e:, a, c, c', \lambda, \lambda\rangle, \text{Strict})]]^{\text{IC}} = \{\beta \cdot \alpha | \beta \in (L\backslash\{a\}), \alpha \in L^\omega\}$.

a.3 带有 past unwanted 消息限制的正则消息:

$[[(\langle e:, a, c, c', \bullet b, \lambda\rangle, \text{Loose})]]^{\text{VT}} = \{\alpha \cdot a | \alpha \in (L\backslash b)^*\}$;

$[[(\langle e:, a, c, c', \bullet b, \lambda\rangle, \text{Loose})]]^{\text{IC}} = \{\alpha | \alpha \in ((L\backslash b) \cap (L\backslash\{a\}))^\omega\}$.

a.4 带有 future unwanted 消息限制的正则消息:

$[[(\langle e:, a, c, c', \lambda, \bullet b\rangle, \text{Loose})]]^{\text{VT}} = \{\alpha \cdot a \cdot \beta | \alpha \in (L\backslash\{a\})^*, \beta \in (L\backslash b)^*\}$;

$[[(\langle e:, a, c, c', \lambda, \bullet b\rangle, \text{Loose})]]^{\text{IC}} = \{\alpha\} \cup \{\beta \cdot a \cdot \gamma\}$, 其中 $\alpha \in (L\backslash\{a\})^\omega, \beta \in (L\backslash\{a\})^*, \gamma \in b^{\omega 3)}$.

a.5 带有 Strict 操作符和 future unwanted 消息限制的正则消息:

3) 该集合等价于 $\{\alpha | \alpha \in (L\backslash\{a\})^\omega\} \cup \{\beta \cdot a \cdot \gamma | \beta \in (L\backslash\{a\})^*, \gamma \in b^\omega\}$, 下文中我们都将如此表示.

$[[(\langle e:, a, c, c', \lambda, \bullet b\rangle, \text{Strict})]]^{\text{VT}} = \{a \cdot \beta | \beta \in (L\backslash b)^*\}$;

$[[(\langle e:, a, c, c', \lambda, \bullet b\rangle, \text{Strict})]]^{\text{IC}} = \{\beta \cdot \alpha\} \cup \{a \cdot \gamma\}$, 其中 $\beta \in (L\backslash\{a\}), \alpha \in L^\omega, \gamma \in b^\omega$.

a.6 带有 past unwanted 链限制的正则消息:

$[[(\langle e:, a, c, c', \nRightarrow g, \lambda\rangle, \text{Loose})]]^{\text{VT}} = \{\beta_1 \cdot a\} \cup \{\beta_1 \cdot m_1 \cdot \beta_2 \cdot a\} \cup \cdots \cup \{\beta_1 \cdot m_1 \cdot \beta_2 \cdot m_2 \cdots \beta_{n-1} \cdot m_{n-1} \cdot \beta_n \cdot a\}$, 其中 $\beta_i \in (L\backslash(\{m_i\} \cup \{a\}))^*, 1 \leqslant i \leqslant n$;

$[[(\langle e:, a, c, c', \nRightarrow g, \lambda\rangle, \text{Loose})]]^{\text{IC}} = \{\beta_1\} \cup \{\beta_1' \cdot m_1 \cdot \beta_2\} \cup \cdots \cup \{\beta_1' \cdot m_1 \cdot \beta_2' \cdots \beta_n' \cdot m_n \cdot \gamma\}$, 其中 $\beta_i' \in (L\backslash(\{m_i\} \cup \{a\}))^*, \beta_i \in (L\backslash(\{m_i\} \cup \{a\}))^\omega, 1 \leqslant i \leqslant n, \gamma \in (L\backslash\{a\})^\omega$.

a.7 带有 past wanted 链限制的正则消息:

$[[(\langle e:, a, c, c', \Rightarrow g, \lambda\rangle, \text{Loose})]]^{\text{VT}} = \{\beta_1 \cdot m_1 \cdot \beta_2 \cdot m_2 \cdots \beta_n \cdot m_n \cdot \gamma \cdot a | \beta_i \in (L\backslash\{m_i\})^*, 1 \leqslant i \leqslant n, \gamma \in (L\backslash\{a\})^*\}$;

$[[(\langle e:, a, c, c', \Rightarrow g, \lambda\rangle, \text{Loose})]]^{\text{IC}} = \{\beta_1\} \cup \{\beta_1' \cdot m_1 \cdot \beta_2\} \cup \cdots \cup \{\beta_1' \cdot m_1 \cdot \beta_2' \cdots \beta_n\} \cup \{\beta_1' \cdot m_1 \cdot \beta_2' \cdots \beta_n' \cdot m_n \cdot \gamma\}$, 其中 $\beta_i' \in (L\backslash\{m_i\})^*, \beta_i \in (L\backslash\{m_i\})^\omega, 1 \leqslant i \leqslant n, \gamma \in (L\backslash\{a\})^\omega$.

a.8 带有 future unwanted 链限制的正则消息:

$[[(\langle e:, a, c, c', \lambda, \nRightarrow g\rangle, \text{Loose})]]^{\text{VT}} = \{\alpha \cdot a\} \cup \{\alpha \cdot a \cdot \beta_1 \cdot m_1\} \cup \{\alpha \cdot a \cdot m_1 \cdot \beta_2 \cdot m_2\} \cup \cdots \cup \{\alpha \cdot a \cdot \beta_1 \cdot m_1 \cdot \beta_2 \cdot m_2 \cdots \beta_{n-1} \cdot m_{n-1}\}$, 其中 $\alpha \in (L\backslash\{a\})^*, \beta_i \in (L\backslash\{m_i\})^*, 1 \leqslant i \leqslant n-1$;

$[[(\langle e:, a, c, c', \lambda, \nRightarrow g\rangle, \text{Loose})]]^{\text{IC}} = \{\alpha\} \cup \{\alpha' \cdot a \cdot \beta_1 \cdot m_1 \beta_2 \cdots \beta_n \cdot m_n \cdot \gamma\}$, 其中 $\alpha \in (L\backslash\{a\})^\omega, \alpha' \in (L\backslash\{a\})^*, \beta_i \in (L\backslash\{m_i\})^*, 1 \leqslant i \leqslant n, \gamma \in L^\omega$.

a.9 带有 future wanted 链限制的正则消息:

$[[(\langle e:, a, c, c', \lambda, \Rightarrow g\rangle, \text{Loose})]]^{\text{VT}} = \{\alpha \cdot a \cdot \beta_1 \cdot m_1 \cdot \beta_2 \cdot m_2 \cdots \beta_n \cdot m_n | \alpha \in (L\backslash\{a\})^*, \beta_i \in (L\backslash\{m_i\})^*, 1 \leqslant i \leqslant n\}$;

$[[\langle e:, a, c, c', \lambda, \Rightarrow g\rangle, \text{Loose})]]^{\text{IC}} = \{\alpha\} \cup \{\alpha' \cdot a \cdot \beta_1\} \cup \{\alpha' \cdot a \cdot \beta_1' \cdot m_1 \cdot \beta_2\} \cup \cdots \cup \{\alpha' \cdot a \cdot \beta_1' \cdot m_1 \cdot \beta_2' \cdots \beta_{n-3} \cdot m_{n-3} \cdot \beta_{n-2}\} \cup \{\alpha' \cdot a \cdot \beta_1' \cdot m_1 \cdot \beta_2' \cdots \beta_{n-2}' \cdot m_{n-2} \cdot \beta_{n-1}\}$, 其中 $\alpha \in (L\backslash\{a\})^\omega, \alpha' \in (L\backslash\{a\})^*, \beta_i \in (L\backslash\{m_i\})^\omega, \beta_i' \in (L\backslash\{m_i\})^*, 1 \leqslant i \leqslant n-1$.

b) 强制消息的轨迹语义: 由于强制消息是系统和环境之间必须交换的消息, 故当不带有 Strict 操作符时强制消息只有接受的正确轨迹集合 ($[[\text{PSC}]^{\text{VT}}]$)、系统和环境接受的有限可控轨迹集合 ($[[\text{PSC}]^{\text{SFC}}]$) 和 ($[[\text{PSC}]^{\text{EFC}}]$) 和接受的错误轨迹集合 ($[[\text{PSC}]^{\text{IVT}}]$), 其他的语义轨迹为空. 当带有 Strict 操作符时, 系统和环境接受的有限可控轨迹集合替换为紧急可控轨迹 ($[[\text{PSC}]]^{\text{SUC}}$) 和 ($[[\text{PSC}]]^{\text{EUC}}$).

b.1 带有 Loose 操作符的强制消息:

$[[(\langle r:, a, c, c', \lambda, \lambda\rangle, \text{Loose})]]^{\text{VT}} = \{\alpha \cdot a | \alpha \in (L\backslash\{a\})^*\}$;

$[[(\langle r:, a, c, c', \lambda, \lambda\rangle, \text{Loose})]]^{\text{SFC}} = \{\alpha | \alpha \in (L\backslash\{a\})^*, a \in \text{Inner} \cup \text{output }\}$;

$[[(\langle r:, a, c, c', \lambda, \lambda\rangle, \text{Loose})]]^{\text{EFC}} = \{\alpha | \alpha \in (L\backslash\{a\})^*, a \in \text{Input }\}$;

$[[(\langle r:, a, c, c', \lambda, \lambda\rangle, \text{Loose})]]^{\text{IVT}} = \{\alpha | \alpha \in (L\backslash\{a\})^\omega\}$.

b.2 带有 Strict 操作符的强制消息:

$[[(\langle r:, a, c, c', \lambda, \lambda\rangle, \text{Strict})]]^{\text{VT}} = \{a\}$;

$[[(\langle r:, a, c, c', \lambda, \lambda\rangle, \text{Strict})]]^{\text{SUC}} = \{\varepsilon | a \in \text{Inner} \cup \text{output }\}$;

$[[(\langle r:, a, c, c', \lambda, \lambda\rangle, \text{Strict})]]^{\text{EUC}} = \{\varepsilon | a \in \text{Input }\}$;

$[[(\langle r:, a, c, c', \lambda, \lambda\rangle, \text{Strict})]]^{\text{IVT}} = \{\alpha | \alpha \in (L\backslash\{a\})^*\}$.

b.3 带有 past unwanted 消息限制的强制消息:

$[[(\langle r:, a, c, c', \bullet b, \lambda\rangle, \text{Loose})]]^{\text{VT}} = \{\alpha \cdot a | \alpha \in (L\backslash b)^*\}$;

$[[(\langle r:, a, c, c', \bullet b, \lambda\rangle, \text{Loose})]]^{\text{SFC}} = \{\alpha | \alpha \in (L\backslash b)^*, \forall m \in b, m \in \text{Inner} \cup \text{Output}\ \};$

$[[(\langle r:, a, c, c', \bullet b, \lambda\rangle, \text{Loose})]]^{\text{EFC}} = \{\alpha | \alpha \in (L\backslash b)^*, \exists m \in b, m \in \text{Input}\ \};$

$[[(\langle r:, a, c, c', \bullet b, \lambda\rangle, \text{Loose})]]^{\text{IVT}} = \{\beta\} \cup \{\alpha \cdot \gamma\}$, 其中 $\beta \in b^\omega, \alpha \in (L\backslash b)^*, \gamma \in (L\backslash\{a\})^\omega$.

b.4 带有 future unwanted 消息限制的强制消息:

$[[(\langle r:, a, c, c', \lambda, \bullet b\rangle, \text{Loose})]]^{\text{VT}} = \{\alpha \cdot a \cdot \beta | \alpha \in (L\backslash\{a\})^*, \beta \in (L\backslash b)^*\};$

$[[(\langle r:, a, c, c', \lambda, \bullet b\rangle, \text{Loose})]]^{\text{SFC}} = \{\alpha | \alpha \in (L\backslash b)^*, \forall m \in b, m \in \text{Inner} \cup \text{Output}\ \};$

$[[(\langle r:, a, c, c', \lambda, \bullet b\rangle, \text{Loose})]]^{\text{EFC}} = \{\alpha | \alpha \in (L\backslash b)^*, \exists m \in b, m \in \text{Input}\ \};$

$[[(\langle r:, a, c, c', \lambda, \bullet b\rangle, \text{Loose})]]^{\text{IVT}} = \{\alpha \cdot a \cdot \beta | \alpha \in (L\backslash\{a\})^*, \beta \in b^*\}.$

b.5 带有 Strict 操作符和 future unwanted 消息限制的强制消息:

$[[(\langle r:, a, c, c', \lambda, \bullet b\rangle, \text{Strict})]]^{\text{VT}} = \{a \cdot \beta | \alpha \in (L\backslash b)^*\};$

$[[(\langle r:, a, c, c', \lambda, \bullet b\rangle, \text{Strict})]]^{\text{SUC}} = \{\alpha | \forall m \in b, m \in \text{Inner} \cup \text{Output}\ \};$

$[[(\langle r:, a, c, c', \lambda, \bullet b\rangle, \text{Strict})]]^{\text{EUC}} = \{\alpha | \exists m \in b, m \in \text{Input}\ \};$

$[[(\langle r:, a, c, c', \lambda, \bullet b\rangle, \text{Strict})]]^{\text{IVT}} = \{\alpha\} \cup \{a \cdot \beta\}$, 其中 $\alpha \in (L\backslash\{a\})^*, \beta \in b^*$.

b.6 带有 past unwanted 链限制的强制消息:

$[[(\langle r:, a, c, c', \nRightarrow g, \lambda\rangle, \text{Loose})]]^{\text{VT}} = \{\beta_1 \cdot a\} \cup \{\beta_1 \cdot m_1 \cdot \beta_2 \cdot a\} \cup \{\beta_1 \cdot m_1 \cdot \beta_2 \cdot m_2 \cdots \beta_{n-1} \cdot m_{n-1} \cdot \beta_n \cdot a\}$, 其中 $\beta_i \in (L\backslash(\{m_i\} \cup \{a\}))^*, 1 \leqslant i \leqslant n$;

$[[(\langle r:, a, c, c', \nRightarrow g, \lambda\rangle, \text{Loose})]]^{\text{SFC}} = \{\beta_1\} \cup \{\beta_1 \cdot m_1 \cdot \beta_2\} \cup \cdots \cup \{\beta_1 \cdot m_1 \cdot \beta_2 \cdots \beta_{n-1} \cdot m_{n-1} \cdot \beta_n\}$, 其中 $\beta_i \in (L\backslash(\{m_i\} \cup \{a\}))^*, 1 \leqslant i \leqslant n, \forall m \in \beta_i, m \in \text{Output} \cup \text{Inner}$;

$[[(\langle r:, a, c, c', \nRightarrow g, \lambda\rangle, \text{Loose})]]^{\text{EFC}} = \{\beta_1\} \cup \{\beta_1 \cdot m_1 \cdot \beta_2\} \cup \cdots \cup \{\beta_1 \cdot m_1 \cdot \beta_2 \cdots \beta_{n-1} \cdot m_{n-1} \cdot \beta_n\}$, 其中 $\beta_i \in (L\backslash(\{m_i\} \cup \{a\}))^*, 1 \leqslant i \leqslant n, \forall m \in \beta_i, m \in \text{Input}$;

$[[(\langle r:, a, c, c', \nRightarrow g, \lambda\rangle, \text{Loose})]]^{\text{IVT}} = \{\beta_1'\} \cup \{\beta_1 \cdot m_1 \cdot \beta_2'\} \cup \cdots \cup \{\beta_1 \cdot m_1 \cdot \beta_2 \cdots m_{n-1} \cdot \beta_n'\}$ 其中 $\beta_i \in (L\backslash\{m_i\} \cup \{a\})^*, \beta_i' \in (L\backslash(\{m_i\} \cup \{a\}))^\omega, 1 \leqslant i \leqslant n, \gamma \in (L\backslash\{a\})^\omega$.

b.7 带有 past wanted 链限制的强制消息:

$[[(\langle r:, a, c, c', \Rightarrow g, \lambda\rangle, \text{Loose})]]^{\text{VT}} = \{\beta_1 \cdot m_1 \cdot \beta_2 \cdot m_2 \cdots \beta_n \cdot m_n \cdot \gamma \cdot a | \beta_i \in (L\backslash\{m_i\})^*, 1 \leqslant i \leqslant n, \gamma \in (L\backslash\{a\})^*\};$

$[[(\langle r:, a, c, c', \Rightarrow g, \lambda\rangle, \text{Loose})]]^{\text{SFC}} = \{\beta_1 \cup \beta_1 \cdot m_1 \cdot \beta_2 \cup \cdots \cup \beta_1 \cdot m_1 \cdot \beta_2 \cdots \beta_{n-1} \cdot m_n \cdot \gamma\}$, 其中 $\beta_i \in (L\backslash\{m_i\})^*, 1 \leqslant i \leqslant n, \gamma \in (L\backslash\{a\})^*, \forall m \in \beta_i, m \in \text{Output} \cup \text{Inner}$;

$[[(\langle r:, a, c, c', \Rightarrow g, \lambda\rangle, \text{Loose})]]^{\text{EFC}} = \{\beta_1 \cup \beta_1 \cdot m_1 \cdot \beta_2 \cup \cdots \cup \beta_1 \cdot m_1 \cdot \beta_2 \cdots \beta_n \cdot m_n \cdot \gamma\}$, 其中 $\beta_i \in (L\backslash\{m_i\})^*, 1 \leqslant i \leqslant n, \gamma \in (L\backslash\{a\})^*, \forall m \in \beta_i, m \in \text{Input}$;

$[[(\langle r:, a, c, c', \Rightarrow g, \lambda\rangle, \text{Loose})]]^{\text{IVT}} = \{\beta_1 \cup \beta_1' \cdot m_1 \cdot \beta_2 \cup \cdots \cup \beta_1' \cdot m_1 \cdot \beta_2' \cdots \beta_n' \cdot m_n \cdot \gamma\}$, 其中 $\beta_i \in (L\backslash\{m_i\})^\omega, \beta_i' \in (L\backslash\{m_i\})^*, 1 \leqslant i \leqslant n, \gamma \in (L\backslash\{a\})^\omega$.

b.8 带有 future unwanted 链限制的强制消息:

$[[(\langle r:, a, c, c', \nRightarrow g, \lambda\rangle, \text{Loose})]]^{\text{VT}} = \{\alpha \cdot a\} \cup \{\alpha \cdot a \cdot \beta_1 \cdot m_1\} \cup \{\alpha \cdot a \cdot \beta_1 \cdot m_1 \cdot \beta_2 \cdot m_2\} \cup \{\alpha \cdot a \cdot \beta_1 \cdot m_1 \cdot \beta_2 \cdot m_2 \cdots \beta_{n-1} \cdot m_{n-1}\}$, 其中 $\alpha \in (L\backslash\{a\})^*, \beta_i \in (L\backslash\{m_i\})^*, 1 \leqslant i \leqslant n-1$;

$[[(\langle r:, a, c, c', \nRightarrow g, \lambda\rangle, \text{Loose})]]^{\text{SFC}} = \{\alpha\} \cup \{\alpha \cdot a \cdot \beta_1\} \cup \{\alpha \cdot a \cdot \beta_1 \cdot m_1 \cdot \beta_2\} \cup \cdots \cup \{\alpha \cdot a \cdot \beta_1 \cdot m_1 \cdot \beta_2 \cdot m_2 \cdot \ldots \cdot \beta_{n-1} \cdot m_{n-1}\}$, 其中 $\alpha \in (L\backslash\{a\})^*, \beta_i \in (L\backslash\{m_i\})^*, 1 \leqslant i \leqslant n-1, \forall m \in \beta_i, m \in \text{Output} \cup \text{Inner}$;

$[[(\langle r:, a, c, c', \nRightarrow g, \lambda\rangle, \text{Loose})]]^{\text{EFC}} = \{\alpha\} \cup \{\alpha \cdot a \cdot \beta_1\} \cup \{\alpha \cdot a \cdot \beta_1 \cdot m_1 \cdot \beta_2\} \cup \cdots \cup \{\alpha \cdot a \cdot \beta_1 \cdot m_1 \cdot \beta_2 \cdot m_2 \cdot \ldots \cdot \beta_{n-1} \cdot m_{n-1}\}$, 其中 $\alpha \in (L\backslash\{a\})^*, \beta_i \in (L\backslash\{m_i\})^*, 1 \leqslant i \leqslant n-1, \forall m \in \beta_i, m \in$Input;

$[[(\langle r:, l, c, c', \nRightarrow g, \lambda\rangle, \text{Loose})]]^{\text{IVT}} = \{\alpha\} \cup \{\alpha' \cdot a \cdot \beta_1 \cdot m_1 \cdot \beta_2 \cdot m_2 \cdots \beta_n \cdot m_n\}$, 其中 $\alpha \in (L\backslash\{a\})^{\omega}, \alpha' \in (L\backslash\{a\})^{*}, \beta_i \in (L\backslash\{m_i\})^{*}, 1 \leqslant i \leqslant n$.

b.9 带有 future wanted 链限制的强制消息:

$[[(\langle r:, a, c, c', \lambda, \Rightarrow g\rangle, \text{Loose})]]^{\text{VT}} = \{\alpha \cdot a \cdot \beta_1 \cdot m_1 \cdot \beta_2 \cdot m_2 \cdots \beta_n \cdot m_n | \alpha \in (L\backslash\{a\})^{*}, \beta_i \in (L\backslash\{m_i\})^{*}, 1 \leqslant i \leqslant n\}$;

$[[(\langle r:, a, c, c', \lambda, \Rightarrow g\rangle, \text{Loose})]]^{\text{SFC}} = \{\alpha\} \cup \{\alpha \cdot a \cdot \beta_1\} \cup \{\alpha \cdot a \cdot \beta_1 \cdot m_1 \cdot \beta_2\} \cup \cdots \cup \{\alpha \cdot a \cdot \beta_1 \cdot m_1 \cdot \beta_2 \cdots \beta_{n-1} \cdot m_{n-1}\}$, 其中 $\alpha \in (L\backslash\{a\})^{*}, \beta_i \in (L\backslash\{m_i\})^{*}, 1 \leqslant i \leqslant n-1, \forall m \in \beta_i, m \in \text{Inner} \cup \text{Output}$;

$[[(\langle r:, a, c, c', \lambda, \Rightarrow g\rangle, \text{Loose})]]^{\text{EFC}} = \{\alpha\} \cup \{\alpha \cdot a \cdot \beta_1\} \cup \{\alpha \cdot a \cdot \beta_1 \cdot m_1 \cdot \beta_2\} \cup \cdots \cup \{\alpha \cdot a \cdot \beta_1 \cdot m_1 \cdot \beta_2 \cdots \beta_{n-1} \cdot m_{n-1}\}$, 其中 $\alpha \in (L\backslash\{a\})^{*}, \beta_i \in (L\backslash\{m_i\})^{*}, 1 \leqslant i \leqslant n-1, \forall m \in \beta_i, m \in \text{Input}$;

$[[(\langle r:, a, c, c', \lambda, \Rightarrow g\rangle, \text{Loose})]]^{\text{IVT}} = \{\alpha\} \cup \{\alpha' \cdot a \cdot \beta_1\} \cup \{\alpha' \cdot a \cdot \beta_1' \cdot m_1 \cdot \beta_2\} \cup \cdots \cup \{\alpha' \cdot a \cdot \beta_1' \cdot m_1 \cdot \beta_2' \cdots \beta_{n-1} \cdot m_{n-1}\}$, 其中 $\alpha \in (L\backslash\{a\})^{\omega}, \alpha' \in (L\backslash\{a\})^{*}, \beta_i \in (L\backslash\{m_i\})^{\omega}, \beta_i' \in (L\backslash\{m_i\})^{*}, 1 \leqslant i \leqslant n-1$.

c) 错误消息的轨迹语义: 错误消息的轨迹语义与强制消息相反, 也当不带有 Strict 操作符时, 有接受的正确轨迹集合 ($[[\text{PSC}]]^{\text{IVT}}$)、系统和环境接受的有限可控轨迹集合 ($[[\text{PSC}]]^{\text{SFC}}$) 和 ($[[\text{PSC}]]^{\text{EFC}}$) 和接受的错误轨迹集合 $[[\text{PSC}]]^{\text{IVT}}$, 其他的语义轨迹为空. 当含 Strict 操作符时, 系统和环境接受的有限可控轨迹集合替换为紧急可控轨迹 ($[[\text{PSC}]]^{\text{SUC}}$) 和 ($[[\text{PSC}]]^{\text{EUC}}$).

c.1 带有 Loose 操作符的错误消息:

$[[(\langle f:, a, c, c', \lambda, \lambda\rangle), \text{Loose})]]^{\text{VT}} = \{\alpha | \alpha \in (L\backslash\{a\})^{\omega}\}$;

$[[(\langle f:, a, c, c', \lambda, \lambda\rangle), \text{Loose})]]^{\text{SFC}} = \{\alpha | \alpha \in (L\backslash\{a\})^{*}, a \in \text{Inner} \cup \text{Input}\}$;

$[[(\langle f:, a, c, c', \lambda, \lambda\rangle), \text{Loose})]]^{\text{EFC}} = \{\alpha | \alpha \in (L\backslash\{a\})^{*}, a \in \text{Output }\}$;

$[[(\langle f:, a, c, c', \lambda, \lambda\rangle), \text{Loose})]]^{\text{IVT}} = \{\alpha \cdot a | \alpha \in (L\backslash\{a\})^{*}\}$.

c.2 带有 Strict 操作符的错误消息:

$[[(\langle f:, a, c, c', \lambda, \lambda\rangle), \text{Strict})]]^{\text{VT}} = \{\alpha | \alpha \in (L\backslash\{a\})^{*}\}$;

$[[(\langle f:, a, c, c', \lambda, \lambda\rangle), \text{Strict})]]^{\text{SUC}} = \{\varepsilon | a \in \text{Inner} \cup \text{Input }\}$;

$[[(\langle f:, a, c, c', \lambda, \lambda\rangle), \text{Strict})]]^{\text{EUC}} = \{\varepsilon | a \in \text{Output }\}$;

$[[(\langle f:, a, c, c', \lambda, \lambda\rangle), \text{Strict})]]^{\text{IVT}} = \{a\}$.

c.3 带有 past unwanted 消息限制的错误消息:

$[[(\langle f:, a, c, c', \bullet b, \lambda\rangle, \text{Loose})]]^{\text{VT}} = \{\beta \cup \alpha \cdot \gamma | \beta \in b^{\omega}, \alpha \in (L\backslash b)^{*}, \gamma \in (L\backslash\{a\})^{\omega}\}$;

$[[(\langle f:, a, c, c', \bullet b, \lambda\rangle, \text{Loose})]]^{\text{SFC}} = \{\alpha | \alpha \in (L\backslash b)^{*}, \forall m \in b, m \in \text{Inner} \cup \text{Output }\}$;

$[[(\langle f:, a, c, c', \bullet b, \lambda\rangle, \text{Loose})]]^{\text{EFC}} = \{\alpha | \alpha \in (L\backslash b)^{*}, \exists m \in b, m \in \text{Input }\}$;

$[[(\langle f:, a, c, c', \bullet b, \lambda\rangle, \text{Loose})]]^{\text{IVT}} = \{\alpha \cdot a | \alpha \in (L\backslash b)^{*}\}$.

c.4 带有 past unwanted 链限制的错误消息:

$[[(\langle f:, a, c, c', \nRightarrow g, \lambda\rangle, \text{Loose})]]^{\text{VT}} = \{\beta_1'\} \cup \{\beta_1 \cdot m_1 \cdot \beta_2'\} \cup \cdots \cup \{\beta_1 \cdot m_1 \cdot \beta_2 \cdots m_{n-1} \cdot \beta_n'\}$ 其中 $\beta_i \in (L\backslash\{m_i\} \cup \{a\})^{*}, \beta_i' \in (L\backslash\{m_i\} \cup \{a\})^{\omega}, 1 \leqslant i \leqslant n, \gamma \in (L\backslash\{a\})^{\omega}$;

$[[(\langle f:, a, c, c', \nRightarrow g, \lambda\rangle, \text{Loose})]]^{\text{SFC}} = \{\beta_1\} \cup \{\beta_1 \cdot m_1 \cdot \beta_2\} \cup \cdots \cup \{\beta_1 \cdot m_1 \cdot \beta_2 \cdots \beta_{n-1} \cdot m_{n-1} \cdot \gamma\}$, 其中 $\beta_i \in (L\backslash\{m_i\})^{*}, 1 \leqslant i \leqslant n-1, \gamma \in (L\backslash\{a\})^{*}, \forall m \in \beta_i, m \in \text{Output} \cup \text{Inner}$;

$[[(\langle f:, a, c, c', \nRightarrow g, \lambda\rangle, \text{Loose})]]^{\text{EFC}} = \{\beta_1\} \cup \{\beta_1 \cdot m_1 \cdot \beta_2\} \cup \cdots \cup \{\beta_1 \cdot m_1 \cdot \beta_2 \cdots \beta_{n-1} \cdot m_{n-1} \cdot \gamma\}$, 其中 $\beta_i \in (L\backslash\{m_i\})^{*}, 1 \leqslant i \leqslant n-1, \gamma \in (L\backslash\{a\})^{*}, \forall m \in \beta_i, m \in \text{Input}$;

$[[(\langle f:, a, c, c', \nRightarrow g, \lambda\rangle, \text{Loose})]]^{\text{IVT}} = \{\beta_1 \cdot a\} \cup \{\beta_1 \cdot m_1 \cdot \beta_2 \cdot a\} \cup \{\beta_1 \cdot m_1 \cdot \beta_2 \cdot m_2 \cdots \beta_{n-1} \cdot m_{n-1} \cdot \beta_n \cdot a\}$, 其中 $\beta_i \in (L\backslash\{m_i\} \cup \{a\})^{*}, 1 \leqslant i \leqslant n$.

c.5 带有 past wanted 链限制的错误消息:

$[[(\langle f :, a, c, c', \Rightarrow g, \lambda\rangle, \text{Loose})]]^{\text{VT}} = \{\beta_1\} \cup \{\beta_1' \cdot m_1 \cdot \beta_2\} \cup \cdots \cup \{\beta_1' \cdot m_1 \cdot \beta_2' \cdots \beta_n' \cdot m_n \cdot \gamma\}$, 其中 $\beta_i \in (L\backslash\{m_i\})^{\omega}, \beta_i' \in (L\backslash\{m_i\})^{*}, 1 \leqslant i \leqslant n, \gamma \in (L\backslash\{a\})^{\omega}$;

$[[(\langle f :, a, c, c', \Rightarrow g, \lambda\rangle, \text{Loose})]]^{\text{SFC}} = \{\beta_1\} \cup \{\beta_1 \cdot m_1 \cdot \beta_2\} \cup \cdots \cup \{\beta_1 \cdot m_1 \cdot \beta_2 \cdots \beta_n \cdot m_n \cdot \gamma\}$, 其中 $\beta_i \in (L\backslash\{m_i\})^{*}, 1 \leqslant i \leqslant n, \gamma \in (L\backslash\{a\})^{*}, \forall m \in \beta_i, m \in \text{Inner} \cup \text{Output}$;

$[[(\langle f :, a, c, c', \Rightarrow g, \lambda\rangle, \text{Loose})]]^{\text{EFC}} = \{\beta_1\} \cup \{\beta_1 \cdot m_1 \cdot \beta_2\} \cup \cdots \cup \{\beta_1 \cdot m_1 \cdot \beta_2 \cdots \beta_n \cdot m_n \cdot \gamma\}$, 其中 $\beta_i \in (L\backslash\{m_i\})^{*}, 1 \leqslant i \leqslant n, \gamma \in (L\backslash\{a\})^{*}, \forall m \in \beta_i, m \in \text{Input}$;

$[[(\langle f :, a, c, c', \Rightarrow g, \lambda\rangle, \text{Loose})]]^{\text{IVT}} = \{\beta_1 \cdot m_1 \cdot \beta_2 \cdot m_2 \cdots \beta_n \cdot m_n \cdot \gamma \cdot a | \beta_i \in (L\backslash\{m_i\})^{*}, 1 \leqslant i \leqslant n, \gamma \in (L\backslash\{a\})^{*}\}$.

4.3.2 组合轨迹语义

定义 11 (Merge 组合) Merge 组合指两个箭头消息通过基本轨迹语义顺序组合后生成的新轨迹集合, 各个轨迹分别定义如下:

$[[\text{msg}_1 \cdot \text{msg}_2]]^{\text{VT}} = \{\alpha_1 \cdot \alpha_2 | \alpha_1 \in [[\text{msg}_1]]^{\text{VT}}, \alpha_2 \in [[\text{msg}_2]]^{\text{VT}}\}$;

$[[\text{msg}_1 \cdot \text{msg}_2]]^{\text{IC}} = [[\text{msg}_1]]^{\text{IC}} \cup [[\text{msg}_2]]^{\text{IC}}$;

$[[\text{msg}_1 \cdot \text{msg}_2]]^{\text{SFC}} = [[\text{msg}_1]]^{\text{SFC}} \cup [[\text{msg}_2]]^{\text{SFC}}$;

$[[\text{msg}_1 \cdot \text{msg}_2]]^{\text{SUC}} = [[\text{msg}_1]]^{\text{SUC}} \cup [[\text{msg}_2]]^{\text{SUC}}$;

$[[\text{msg}_1 \cdot \text{msg}_2]]^{\text{EFC}} = [[\text{msg}_1]]^{\text{EFC}} \cup [[\text{msg}_2]]^{\text{EFC}}$;

$[[\text{msg}_1 \cdot \text{msg}_2]]^{\text{EUC}} = [[\text{msg}_1]]^{\text{EUC}} \cup [[\text{msg}_2]]^{\text{EUC}}$;

$[[\text{msg}_1 \cdot \text{msg}_2]]^{\text{IVT}} = [[\text{msg}_1]]^{\text{IVT}} \cup [[\text{msg}_2]]^{\text{IVT}}$.

定义 12 (Alternative 组合) Alternative 组合指两个箭头消息通过基本轨迹语义选择组合后生成的新轨迹集合, 各个轨迹分别定义如下: $[[\text{msg}_1 \otimes \text{msg}_2]]^{\text{VT}} = [[\text{msg}_1]]^{\text{VT}} \cup [[\text{msg}_2]]^{\text{VT}}$;

$[[\text{msg}_1 \otimes \text{msg}_2]]^{\text{IC}} = [[\text{msg}_1]]^{\text{IC}} \cup [[\text{msg}_2]]^{\text{IC}}$;

$[[\text{msg}_1 \otimes \text{msg}_2]]^{\text{SFC}} = [[\text{msg}_1]]^{\text{SFC}} \cup [[\text{msg}_2]]^{\text{SFC}}$;

$[[\text{msg}_1 \otimes \text{msg}_2]]^{\text{SUC}} = [[\text{msg}_1]]^{\text{SUC}} \cup [[\text{msg}_2]]^{\text{SUC}}$;

$[[\text{msg}_1 \otimes \text{msg}_2]]^{\text{EFC}} = [[\text{msg}_1]]^{\text{EFC}} \cup [[\text{msg}_2]]^{\text{EFC}}$;

$[[\text{msg}_1 \otimes \text{msg}_2]]^{\text{EUC}} = [[\text{msg}_1]]^{\text{EUC}} \cup [[\text{msg}_2]]^{\text{EUC}}$;

$[[\text{msg}_1 \otimes \text{msg}_2]]^{\text{IVT}} = [[\text{msg}_1]]^{\text{IVT}} \cup [[\text{msg}_2]]^{\text{IVT}}$.

根据 Merge 和 Alternative 组合, 可定义操作符 Alt, Par 和 Loop 的轨迹语义.

5 实例分析

本节结合开放环境下一个嵌入式系统 RailCab[18] 为例, 详细介绍如何根据 PSC^{Mon} 的多值监控语义生成运行时监控器.

RailCab 是一种新型的运输系统, 重要组成部分是微型的无人驾驶车辆, 使得人们能够在轨道上进行舒适的旅程, 同时满足个体机动性的愿望和满足本地或长途的交通要求. RailCab 不需要卸载或更换列车, 可直接载乘旅客或货物到达目的地. RailCab 不是按计划行驶和调度的, 而是根据需求, 用户可以通过电信服务随时随地订购 RailCab, 从而增加了系统的开放性和动态性. 这些 RailCab 在通常的路线上能自动护航, 从而增加运输能力, 节约巨额能源.

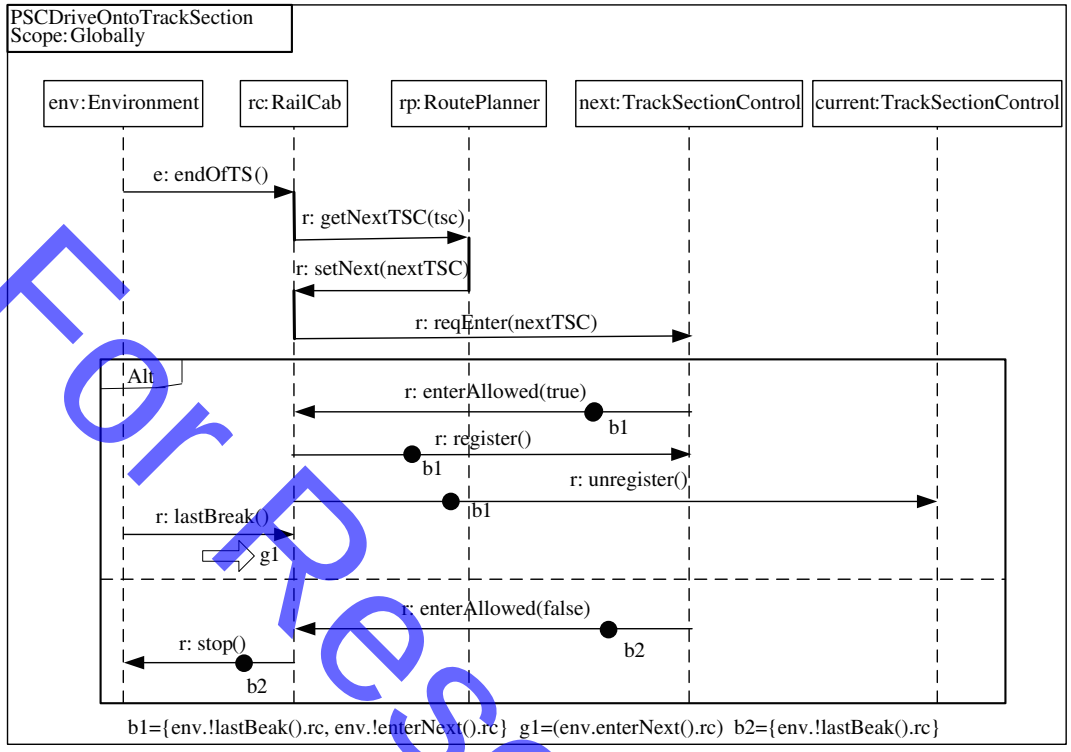

图 6 场景 (3)(DriveOntoTrackSection) 的 PSC^{Mon} 规约表示

Figure 6 PSC^{Mon} specifications for Scenario (3) (DriveOntoTrackSection)

Example 场景 (3)(driveontotracksection): 描述一辆 RailCab 请求安全进入下一段路. 当 RailCab 到达当前线路末尾时 (endOfTS), 必须要求线路规划构件 (route planner component) 为其登记进入下一段可能线路 (getNextTSC(tsc)). 航线规划构件必须马上做出回应, 将下一段路名称作为控制参数 (setNext(nextTSC)) 传递给 RailCab. 接着, RailCab 必须请求获准进入下一段路 (reqEnter(nextTSC)), 当被允许进入下一段路 (enter Allowed(true)) 时, 即如果绑定变量的值 isAllowed 是 true 时, 那么必须将 RailCab 注册到下一段路, 并把它从当前线路注销. 上述所有消息的发送过程必须是 RailCab 在最后一个刹车安全点 (lastBreak()) 和进入下一段路 (enterNext()) 之前完成. 但是, 如果检测到进入下一段路可能发生危险的情况, 将会被禁止进入下一段路 (enterAllowed(false)), 并且必须在最后一个刹车安全点 (lastBreak()) 信号发生之前停止.

场景 (3) 的 PSC^{Mon} 规约表示如图 6 所示. 环境发给 RailCab 的消息是正则消息. 接着, 三个严格生命线表示的强制消息, Alt 操作符描述了允许进入下一段路和不允许进入下一段路两种情况. 关于场景 (3) 生成的博弈结构如图 7 所示, s_0 处于系统和环境无限可控的状态, 接着遇到三个严格操作符的消息, 则处于系统紧急可控状态 ($s_1 - s_3$). 接着 Alt 操作中描述的消息需要环境不能发送 b_1 或 b_2 中的消息给系统, 故 ($s_4 - s_8$, s_9) 处于环境有限可控状态, 如果系统期望的消息不发生或消息限制中的消息发生, 则到达违例状态. 由于该属性的 Scope 是 Globally 类型, 最终当 Alt 中某个情况执行完毕后又回到系统和环境无限可控状态.

6 相关工作

同本文相关的研究工作有两个方面, 第一个方面是关于规约语言监控语义的定义. 在运行时监控

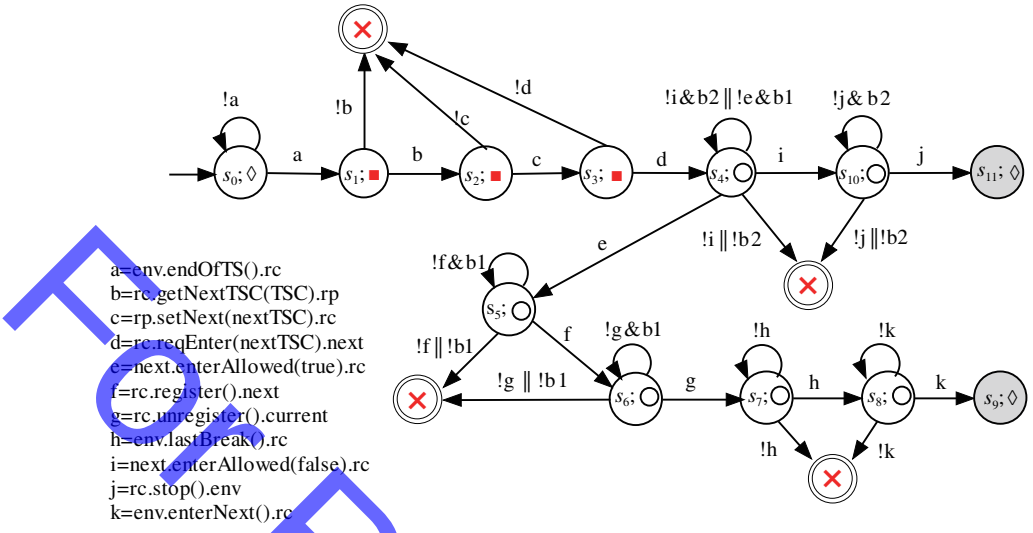

图 7 (网络版彩图) 场景 (3) 生成的博弈结构

Figure 7 (Color online) The game structure for Scenario (3)

研究领域中, 关于如何采用最佳方式将接受无限轨迹的规约转换为接受有限轨迹的监控器, 已存在较长时间的争论. 在早期工作中, Kupferman 等 [7] 使用 “信息前缀 (informative prefix)” 表示能够根据这种前缀检测出规约违例. 信息前缀的好处是监控器不需要分析其将来行为而能得出结论. 然而, 信息前缀的长度经常比需要的长, 比如说, 规约 False 的信息前缀长度是 1, 但是可不需要根据任何轨迹 (即长度为 0) 就可以得出违例. 为了尽可能早地识别出违例, 需要更有质量的前缀. “坏前缀” 是一个有限的前缀, 且该前缀不能是任意接受轨迹的前缀, 为了构建满足 “坏前缀” 的监控器, 可将 LTL 公式转化为等价的不确定 Büchi 自动机, 消除接受空语言的状态, 并采用子集法构建一个等价的接受有限坏前缀的确定自动机. Rosu 等 [8] 说明了如何识别 “坏前缀” 并从监控器中剪裁这类状态, 从而大大降低由监控带来的运行时负载. 文 [19] 介绍了一种基于自动机理论的参数化 LTL(parameterized LTL(linear temporal logic)) 运行时预测监控器构造方法, 在该方法中, 参数化监控器能够精确地识别被验证性质的最小好/坏前缀. 这些研究都基于传统的二值语义, 没有考虑到监控语义的多值情况, 然而这些研究为后续多值监控语义的研究奠定了理论基础.

Bauer 等 [20] 指出传统的二值语义并不能监控所有的属性, 故针对有限轨迹扩展 LTL 逻辑提出了一种 LTL_3 逻辑, 基本思想也是将有限轨迹 u 视为一个无限轨迹的前缀. LTL_3 语义定义的基本思想如下: 如果含有 u 为前缀的每个无限轨迹的值和 u 的值一样, 为满足或违例, 则在 LTL_3 中也是满足或违例; 如果含有 u 的无限轨迹不能得出结论, 则在 LTL_3 中也不能得出结论, 用 “?” 来表示. Bauer 等 [21] 在后续工作中, 建立了运行时验证 LTL 逻辑的四个假设. 由于现有的逻辑都不完全满足这些假设, 引入一个新的满足这四个假设的四值逻辑 RV-LTL, 并根据这些假设定义了描述系统行为的有限轨迹是否满足 RV-LTL 的运行时四值语义, 其中 (1) 满足, (2) 违例, (3) 属性可能违例, 或 (4) 一旦系统趋于稳定, 属性可能满足. 其中 (1) 和 (2) 符合 LTL 的经典语义, 而 (3) 和 (4) 表示检测系统当前行为尚未得出属性满足或违例的结论. 这些方法主要是定义 LTL 逻辑公式的多值监控语义, 而且未考虑到环境和系统之间的可控性.

第二个方面是关于博弈理论在规约语言的语义和综合算法中的应用. 博弈理论在规约语言的综合

算法中得到较为广泛应用. 文 [22,23] 考虑了如何从基于场景的规约 LSC 中自动综合生成反应系统, 考虑将系统规约看成系统和环境之间博弈关系, 综合问题相当于系统存在一个赢的策略, 并设计算法显示如何进行综合. 反应系统的综合指检测系统规约是否是从初始状态是可实现的, 而本文的方法从系统和环境之间的博弈关系分析, 并且为系统参与者寻找一个赢的策略. 故综合和本文的监控方法的关键区别在于综合仅仅是从初始状态检测是否存在这样的策略, 而监控是根据观察到的系统轨迹, 在运行时反复多遍做这样的判断. 文 [24] 提出了一种基于博弈论的 LTL 逻辑的多值监控语义, 将有限轨迹作为无限轨迹的前缀, 不仅要判断正确和错误的存在, 还可判断系统和环境是否有能力来避免违例. 将监控状态分为四种: (1) 违例: 没有执行满足规约; (2) 不可实现性: 意味着控制环境可以满足规约; (3) 可实现性: 意味着控制系统可以满足规约; (4) 满足: 表示所有可能延续满足规约. 本文的方法借鉴了该方法的思想, 然而本文方法可视为该方法三个方面的扩展: (1) 在其基础上将四值语义扩展为更多值的监控语义; (2) 将多值语义的方法扩展到基于场景规约 PSC$^{\text{Mon}}$ 中, 从而更具有实用性和应用前景; (3) 从操作语义和指称语义两种角度定义了其语义规则的一致性.

7 结束语

本文探讨了在开放环境下基于场景规约 PSC$^{\text{Mon}}$ 的多值监控语义, 以在开放环境下精确监控系统的行为. 通过多值监控语义的定义, 监控器不仅能够判断出满足和违例状态, 还能判断出无限可控、系统有限可控、系统紧急可控、环境有限可控、环境紧急可控等状态, 从而提供足够信息为失效的预防和恢复服务. 并结合一个开放环境下的嵌入式系统 RailCab 为例对 PSC$^{\text{Mon}}$ 进行了实例研究. 下一步的工作包括: 开发支持生成 PSC$^{\text{Mon}}$ 的多值监控语义完全自动化的工具 [25], 帮助一般的软件工程师来表示系统待监控的需求规约, 考虑将 PSC$^{\text{Mon}}$ 应用在真实环境进行行为监控, 并定义具体的预防和恢复策略.

参考文献

1 Yang F Q. Thinking on the development of software engineering technology. J Softw, 2005, 16: 1–7 [杨芙清. 软件工程技术发展思索. 软件学报, 2005, 16: 1–7]

2 Mei H, Cao D G. Software trustworthiness: the challenges of the Internet. Commun CCF, 2010, 6: 20–37 [梅宏, 曹东刚. 软件可信性: 互联网带来的挑战. 中国计算机学会通讯, 2010, 6: 20–37]

3 Shen C X, Zhang H G, Wang H M, et al. Research and development of trusted computing. Sci Sin Inform, 2010, 40: 139–166 [沈昌祥, 张焕国, 王怀民, 等. 可信计算的研究与发展. 中国科学: 信息科学, 2010, 40: 139–166]

4 Clarke E M, Emerson E A, Sistla A P. Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Trans Progr Lang Sys, 1986, 8: 244–263

5 Delgado N, Gates A, Roach S. A taxonomy and catalog of runtime software-fault monitoring tools. IEEE Trans Softw Eng, 2004, 30: 859–872

6 Leucker M, Schallhart C. A brief account of runtime verification. J Log Algebr Program, 2009, 78: 293–303

7 Kupferman O, Vardi M Y. Model checking of safety properties. Form Method Syst Des, 2001, 19: 291–314

8 d'Amorim M, Rosu G. Efficient monitoring of ω-languages. In: Etessami K, Rajamani S K, eds. CAV. Volume 3576 of LNCS, Berlin: Springer, 2005. 364–378

9 ITU. International Telecommunication Union Recommendation Z.120: Message Sequence Charts. Technical Report. 1996

10 Object Management Group (OMG). UML: Superstructure Version 2.2. 2012

11 Autili M, Inverardi P, Pelliccione P. Graphical scenarios for specifying temporal properties: an automated approach.

Autom Softw Eng, 2007, 14: 293–340
12 Zhang P C, Zhou Yu, Li B X, et al. Property sequence chart: formal syntax and semantics. J Comput Res Dev, 2008, 45: 318–328 [张鹏程, 周宇, 李必信, 等. 属性序列图: 形式语法和语义. 计算机研究与发展, 2008, 45: 318–328]
13 Damm W, Harel D. LSCs: breathing life into message sequence charts. Form Method in Syst Des, 2001, 19: 45–80
14 Harel D, Maoz S. Assert and negate revisited: modal semantics for UML sequence diagrams. Softw Syst Model, 2008, 7: 237–252
15 Li W R, Wang Z J, Zhang P C. Formal semantics of universal modal sequence diagram. J Softw, 2011, 22: 659–675 [李雯睿, 王志坚, 张鹏程. 模态顺序图 uMSD 的形式语义. 软件学报, 2011, 22: 659–675]
16 Dwyer M B, Avrunin G S, Corbett J C. Property specification patterns for finite-state verification. In: Proceedings of the 21st Int'l Conference on Software Engineering, Los Angeles, 1999. 411–420
17 Alfaro L de, Stoelinga M. Interfaces: a game-theoretic framework for reasoning about component-based systems. Electr Notes Theor Comput Sci, 2004, 97: 3–23
18 Greenyer J, Kindler E. Compositional Synthesis of Controllers from Scenario-Based Assume-Guarantee Specifications. Berlin Heidelberg: Springer, 2013. 774–789
19 Zhao C Z, Dong W, Sui P, et al. Construction techniques of anticipatory monitors for parameterized LTL. J Software, 2010, 21: 318–333 [赵常智, 董威, 隋平, 等. 面向参数化 LTL 的预测监控器构造技术. 软件学报, 2010, 21: 318–334]
20 Bauer A, Leucker M, Schallhart C. Runtime verification for LTL and TLTL. ACM Trans Softw Eng Meth, 2011, 20: 14
21 Bauer A, Leucker M, Schallhart C. Comparing LTL semantics for runtime verification. J Log Comput, 2010, 20: 651–674
22 Kugler H, Plock C, Pnueli A. Controller Synthesis from LSC Requirements. In: Proceedings of the 12th Fundamental Approaches to Software Engineering. Berlin: Springer-Verlag, 2009. 79–93
23 Harel D, Segall I. Synthesis from scenario-based specifications. J Comput Sys Sci, 2012, 78: 970–980
24 Ehlers R, Finkbeiner B. Monitoring Realizability. In: Proceedings of the 2nd International Conference on Runtime Verification. Berlin: Springer, 2012. 427–441
25 Zhang P C, Su Z Y, Zhu Y L, et al. WS-PSC Monitor: a tool chain for monitoring temporal and timing properties in composite service based on property sequence chart. In: Proceedings of the 1st International Conference on Runtime Verification. Berlin Heidelberg: Springer, 2010. 485–489

Game-based monitors for a scenario-based specification in open environments

ZHANG PengCheng[1,2*], LI XuanDong[1] & LI WenRui[3]

[1] *State Key Laboratory of Novel Software Technology, Nanjing University, Nanjing* 210093, *China;*
[2] *College of Computer and Information, Hohai University, Nanjing* 211100, *China;*
[3] *School of Mathematics and Information Technology, Nanjing Xiaozhuang University, Nanjing* 211171, *China*
*E-mail: pchzhang@nju.edu.cn

Abstract In open environments, unsafe run-time changes of systems and environments may compromise the correct execution of the entire systems and make the software systems do not meet the original specifications, which may eventually lead to the occurrence of software failures. Runtime monitor which is a lightweight formal dynamic verification technology has become the basic means of detecting software failures in open environments. For scenario-based specification property sequence charts, this paper defines the multi-valued monitoring semantics from the perspective of game theory: satisfied, infinitely controllable, the system is finitely controllable, the system

is emergency controllable, the environment is finitely controllable, the environment is emergency controllable, violated. Through the multi-valued semantics definition, the monitor can detect failures as early as possible and also provide sufficient information to help the system to take measures for failure prevention and recovery. Finally, the property sequence chart used in RailCab case study shows its extensive application prospect.

Keywords open environments, scenario-based specification, property sequence chart, multi-valued monitor semantics, game structure

ZHANG PengCheng was born in 1981. He received his Ph.D. degree in software engineering from Southeast University in 2010. He is now an associate Professor at Hohai University and also a post-doctor at Nanjing University. His research interests include software modeling, analysis and verification.

LI XuanDong was born in 1963. He received his Ph.D. degree in software engineering from Nanjing University. He is now a full professor in the Department of Computer Science and Technology at Nanjing University. His research interests include software modeling and analysis, software testing and verification.

LI WenRui was born in 1981. She received her Ph.D. degree in computer application technique from Hohai University in 2010. She is now a lecturer at Nanjing xiaozhuang University. Her research interests include software modeling, analysis and verification.