CN 43-1258/TP
ISSN 1007-130X

计算机工程与科学
Computer Engineering & Science

第 45 卷第 5 期 2023 年 5 月
Vol. 45, No. 5, May. 2023

文章编号:1007-130X(2023)05-0820-10

套索边界条件:一种可指导目标冲突修复的分歧描述*

罗炜麟,万　海,杨滨好,李骁达,曹鉴恩,宋晓彤
(中山大学计算机学院,广东 广州 510006)

摘　要:在需求工程中,分歧的目标冲突分析旨在识别、评估和修复分歧。分歧是由于域属性和目标不适配,使得系统在边界条件下无法同时满足所有目标。边界条件以线性时态逻辑的形式描述分歧。由于任意形式的边界条件缺乏可解释性,且评估和修复分歧需要大量人工评估和设计,所以目前边界条件的定义不利于高效地、自动化地评估和修复分歧。为此,首先提出一个可解释的边界条件——套索边界条件,直观地描述了系统由于一些特定的前提条件而产生分歧的情况;然后,设计了一个基于逐步弱化的套索边界条件识别算法 LBCI,通过弱化线性时态逻辑公式逐步满足边界条件;最后,在基准数据集上评估了套索边界条件和 LBCI 的有效性。实验结果显示,套索边界条件增强了分歧的可解释性和对修复分歧有一定的指导作用。

关键词:目标冲突分析;边界条件;分歧
中图分类号:TP393　　**文献标志码**:A
doi:10.3969/j.issn.1007-130X.2023.05.007

Lasso boundary condition: A divergence description guiding goal-conflict resolution

LUO Wei-lin, WAN Hai, YANG Bin-hao, LI Xiao-da, CAO Jian-en, SONG Xiao-tong
(School of Computer Science and Engineering, Sun Yat-Sen University, Guangzhou 510006, China)

Abstract: Goal-conflict analysis of divergences in requirement engineering aims to identify, assess, and resolve divergences. The divergence is caused, because the mismatch between domain attributes and objectives makes the system unable to satisfy all objectives at the same time under boundary conditions. Boundary conditions describe disagreements in the form of linear temporal logic. Due to the lack of interpretability of arbitrary boundary conditions and the extensive manual evaluation and design required to evaluate and repair divergences, the current definition of boundary conditions is not conducive to efficient and automated evaluation and repair of divergences. Therefore, in this paper, an explainable boundary condition, called lasso boundary condition, is proposed. Lasso boundary condition intuitively describes the situation where the system diverges due to some specific preconditions. Then, a lasso boundary condition identification algorithm (LBC identifier, LBCI) based on gradual weakening is designed. LBCI gradually satisfies the boundary conditions by weakening the linear temporal logic formula. The effectiveness of lasso boundary condition and LBCI are evaluated on a baseline data set. The experimental results show that the lasso boundary condition can enhance the interpretability of the divergences and the guiding role in repairing the divergences.

Key words: goal-conflict analysis; boundary condition; divergence

* **收稿日期**:2022-02-14;**修回日期**:2022-04-11
基金项目:国家自然科学基金(61976232);广东省自然科学基金(2023A1515011470)
通信作者:万海(wanhai@mail.sysu.edu.cn)
通信地址:510006 广东省广州市中山大学计算机学院
Address: School of Computer Science and Engineering, Sun Yat-Sen University, Guangzhou 510006, Guangdong, P. R. China

1 引言

面向目标的需求工程 GORE(Goal-Oriented Requirement Engineering)[1]的目标是获得正确的软件需求,这对软件开发生命周期十分重要。在 GORE 中,域属性(Domain)和目标(Goal)以线性时态逻辑 LTL(Linear Temporal Logic)表征系统行为。LTL 是一种被广泛认为适用于抽象类需求、假设和领域属性的公式,因为它可以表达时间属性且表示简洁[1]。由于需求是多方面综合的结果,在需求工程过程中可能会出现各种各样的分歧(Divergence)[2]。分歧是一种弱不一致性,即某些目标的满足会抑制其他目标的满足。边界条件 BC(Boundary Condition)用 LTL 公式表示,通常用于描述分歧。

在 GORE 中,分歧的目标冲突分析[1,3]通过识别、评估和修复 3 个阶段来完成。整个过程需要多次迭代和大量的人机交互,因此提供易于解释分歧机理的 BC 具有重要意义。目前,研究人员已经提出了各种方法[2,4-6]来自动识别并评估 BC,但大多是基于任意形式的 LTL 公式表示的 BC。

本文主要关注 BC 的可解释性对修复分歧的作用。分歧的发生是由于不适配的域属性和目标,系统在 BC 下会出现一些不能同时满足所有目标的情况。本文观察到,用任意形式的 LTL 公式表示的 BC 由于嵌套了复杂的时态运算符而缺乏可解释性。并且,难以解释的 BC 使得其难以为解决分歧提供指导。预定义的模板 BC[2]可以提高可解释性,然而因为它们只适用特定的领域属性和目标,在实践中缺乏可扩展性。

为了解决可解释 BC 的建模问题,本文引入套索边界条件 LBC(Lasso Boundary Condition)的概念(定义 4)。LBC 是对前缀-后缀公式的析取(定义 3)。前缀-后缀公式对应 1 个前缀集合和 1 个后缀集合。直观地,前缀集合表示特殊的状态迁移,本文称之为前提条件;后缀集合表示一组特殊的状态集合,是系统通过由前缀集合刻画的迁移后可能到达的状态集合。因此,LBC 描述了系统由于前缀集合描述的前提条件出现分歧的情况。本文认为 LBC 可以解释分歧产生的机制,并可用于指导解决分歧。虽然 LBC 也是一种模板,但它不依赖于固定的域属性和目标,与现有模板相比,具有更强的可扩展性。基于 LBC,本文设计了基于逐步弱化的 LBC 识别算法 LBCI(LBC Identifier)。LBCI 的核心思想是通过弱化 LTL 公式逐步满足 BC 成立条件。

本文在基准数据集上评估了 LBC 和 LBCI,实验结果显示了 LBC 的优越性,表明 LBC 可以直观地解释分歧产生的机制,且能够表达大多数分歧。实验结果还证实了 LBCI 识别 LBC 的有效性,可以高效地识别绝大多数数据集的分歧。

2 相关工作

在需求工程领域中,使用形式化规约能够精确刻画软件的预期功能和运行场景,尤其对于安全性要求较高的场景具有较大的价值[7,8]。使用 LTL 对面向目标的需求工程建模后进行目标冲突分析能够获得正确的软件需求。目标冲突分析通常由识别-评估-修复循环驱动,旨在识别、评估和修复可能阻碍预期目标实现的不一致性。在这项工作中,本文关注弱不一致性,即 BC。

早期,van Lamsweerde 等人[2]提出了一种基于模板的方法,该方法仅以预定义的有限形式返回 BC,但依赖于人类知识总结的模板。Degiovanni 等人[6]开发了一种基于 tableaux 的方法来生成 BC,但因为 tableaux 难以构建,只适用于小规模的需求。近年来,提出了许多基于搜索的方法,极大地提高了识别 BC 的性能。Degiovanni 等人[5]提出了一种识别 BC 的遗传算法,可以在大型基准测试中找到数十甚至数百个 BC。Zhong 等人[9]发现 BC 之间的语法存在相似性,设计了一种局部搜索算法来识别更多的 BC。

此外,Degiovanni 等人[5]意识到识别出的 BC 数量的增加给评估和修复带来了负担,提出了通用性(General)的概念,过滤掉了不太通用的 BC 以缩小 BC 的集合。最近,Luo 等人[10]引入了一个细粒度的度量,即对比度(Contrasty),用来评估 2 个 BC 是否表示相同的分歧。本文使用这 2 个评价指标评估 LBC 的有效性。

随着识别出的 BC 数量的增加,评估阶段和修复阶段代价变大,甚至失去了可行性。近期,研究人员广泛讨论了 GORE 的评估阶段,以优先考虑要解决的 BC,并建议关注哪些目标以进行改进。然而,现有的工作[11-15]只考虑评估阻止满足单个目标的障碍,并假设提供了该领域的某些概率信息。为了能自动评估 BC,Degiovanni 等人[4]最近提出了一种基于模型计数的自动化方法来评估冲突的可能性。

目前还没有修复分歧的自动化方法,因为难以使用任意 LTL 表示的 BC 指导修复,并且需要工程师的大量经验来设计和评估修复。Felfernig 等人[16]基于模型诊断来计算需求冲突的个性化修复。Murukannaiah 等人[17]基于对竞争性假设和论证模式的分析,修复系统的利益相关者目标之间的冲突。

有一些工作侧重于研究 LTL 子集,即 Generalized Reactivity(1)公式(GR(1))。改进不可实现的 GR(1)公式的一种常见方法是在反策略(Counter-Strategy)指导下对环境添加新假设,这解释了原始公式的不可实现性。因此,需要通过额外的假设来排除。Cavezza 等人[18]通过对不可实现的核心(Unrealizable Core)进行插值分析,计算出了一组与反策略不一致的候选假设。Brizzio 等人[19]则将候选解确定为那些总体上运行反策略的公式的补充。相似地,Li 等人[20]提出了一种基于模板的方法来挖掘满足反策略的公式,然后可以通过将这种公式的否定添加到原始假设来移除反策略。Maoz 等人[21]提出了 2 种新颖的符号算法 JVTS-Repair 和 GLASS,用于修复无法实现的 GR(1)公式。

但是,上述方法仅限于以 GR(1)形式表示的那些公式。为了获得更好的灵活性,适用于任意 LTL 公式的方法都值得考虑。Chatterjee 等人[22]通过应用两步算法来构建假设ϕ,使得$\phi \rightarrow \psi$是可实现的,从而更正最初的不可实现公式ψ。Brizzio 等人[19]利用遗传算法生成可实现的公式,在语法和语义上尽可能多地产生与原始不可实现的 LTL 公式较为相似的修复。

由于 BC 定义的独特性,上述方法不能直接用于 BC 分析。但是,这些方法还是值得借鉴的。比如,定义新的概念用于修复不可实现性,直接启发了本文从修复分歧的角度提出 LBC。

3 背景

3.1 线性时态逻辑

线性时态逻辑 LTL[23]被广泛用于描述离散系统的无限行为。本文使用小写字母(例如 p、h)来表示命题。有限命题 $\mathbb{P}$的 LTL 的最小语法集合包括标准逻辑运算符或($\bigcirc$)和否定($\neg$)、$B=\{\bot,\top\}$(其中$\bot$表示逻辑假,$\top$表示逻辑真),以及时态运算符 next($\bigcirc$)和 until($\mathcal{U}$),如式(1)所示:

$$\phi := p \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \bigcirc\phi \mid \phi_1 \mathcal{U} \phi_2 \tag{1}$$

其中,$p \in \mathbb{P} \cup \{\bot,\top\}$。and($\wedge$)、release($\mathcal{R}$)、future($\Diamond$)、always($\Box$)和 weak-until($\mathcal{W}$)是常用的运算符,可以定义如式(2)~式(6)所示:

$$\phi_1 \wedge \phi_2 := \neg(\neg\phi_1 \vee \neg\phi_2) \tag{2}$$

$$\phi_1 \mathcal{R} \phi_2 := \neg(\neg\phi_1 \mathcal{U} \neg\phi_2) \tag{3}$$

$$\Diamond\varphi := \top \mathcal{U} \phi \tag{4}$$

$$\Box\phi := \neg(\top \mathcal{U} \neg\phi) \tag{5}$$

$$\phi_1 \mathcal{W} \phi_2 := \phi_1 \mathcal{U} (\phi_2 \vee \Box\phi_1) \tag{6}$$

如果非($\neg$)仅仅作用于原子命题,则 LTL 公式 ϕ 可称为否定范式 NNF(Negation Normal Form)。将 LTL 公式转换成否定范式的方式是通过等价转换使得否定只作用于原子命题前。本文使用 $|\phi|$ 表示公式ϕ的大小,即ϕ中时态运算符、逻辑连接词和原子命题的数量。

定义 1(子公式) LTL 公式ϕ的子公式 $sub(\phi)$ 的集合递归定义如下所示:

(1)如果 $\phi=p$,则 $sub(\phi)=\{p\}$;

(2)如果$\phi=o_1\phi'$,则 $sub(\phi)=\{\phi\} \cup sub(\phi')$;

(3)如果$\phi=\phi' o_2 \phi''$,则 $sub(\phi)=\{\phi\} \cup sub(\phi') \cup sub(\phi'')$。

其中,$o_1 \in \{\neg,\bigcirc,\Diamond,\Box\}$;$o_2 = \in \{\vee,\wedge,\mathcal{U},\mathcal{R},\mathcal{W}\}$;$\phi'$和$\phi''$是 LTL 公式。

LTL 公式可以用来表示无限命题状态的无限长轨迹。无限轨迹可以用套索的方式表示,表示方式为 $\pi=s_0\cdots s_k(s_{k+1}\cdots s_m)^\omega$,其中 $s_t \in 2^{\mathbb{P}}$是在时刻 t 时的状态集合,$s_0\cdots s_k$ 是前缀,$(s_{k+1}\cdots s_m)^\omega$ 是循环,$s_{k+1}\cdots s_m$ 按顺序无限出现。本文将 π_t 记为轨迹 π 从状态 s_t 开始的子轨迹,$t \in \mathbb{N}$($\mathbb{N}$表示整数),ϕ、ϕ_1 和ϕ_2为 LTL 公式,关系$\vDash$定义如下:

(1)$\pi_t \vDash p$ 当且仅当 $p \in s_t$,其中 $p \in \mathbb{P} \cup \{\top,\bot\}$;

(2)$\pi_t \vDash \neg\phi$当且仅当 $\pi_t \nvDash \phi$;

(3)$\pi_t \vDash \phi_1 \vee \phi_2$ 当且仅当 $\pi_t \vDash \phi_1$ 或 $\pi_t \vDash \phi_2$;

(4)$\pi_t \vDash \phi_1 \wedge \phi_2$ 当且仅当 $\pi_t \vDash \phi_1$ 且 $\pi_t \vDash \phi_2$;

(5)$\pi_t \vDash \bigcirc\phi$当且仅当 $\pi_{t+1} \vDash \phi$;

(6)$\pi_t \vDash \phi_1 \mathcal{U} \phi_2$ 当且仅当 $\exists k \geqslant t$,使得 $\pi_k \vDash \phi_2$ 且 $\forall t \leqslant j < k, \pi_j \vDash \phi_1$。

LTL 公式ϕ是可满足的,当且仅当存在轨迹 π 使得 $\pi_0 \vDash \phi$。如果ϕ的模型也都是ϕ'的模型,则 LTL 公式ϕ蕴含 LTL 公式ϕ',记为$\phi \rightarrow \phi'$。LTL 可满足性问题是检查一个 LTL 公式是否可满足,是 PSPACE-complete 问题[24]。目前研究人员已经开发了基于不同技术的 LTL 可满足性求解器,例

如 nuXmv[25]和 Aalta[26]。

Fisman 等人[27]介绍了 LTL 公式的强化/弱化性质。

性质 1[27] 设ϕ是 NNF 的 LTL 公式,ψ是ϕ的子公式,ψ'是一个 LTL 公式,满足 $\psi'\rightarrow\psi$(或 $\psi\rightarrow\psi'$)。如果$\phi'=\phi[\psi'/\psi]$,则 $\phi'\rightarrow\phi$(或$\phi\rightarrow\phi'$),其中$\phi[\psi'/\psi]$是将ϕ中的每个子公式 ψ 替换为ψ'得到的公式。另有,$\phi[\bot/\psi]\rightarrow\phi$及$\phi\rightarrow\phi[\top/\psi]$。

性质 1 保证了本文可以通过增强或减弱其子公式来强化或弱化 LTL 公式。该性质是本文识别 BC 算法的基础。

3.2 目标冲突分析

在 GORE[1]中,目标是系统必须实现的规定性陈述,领域属性是描述问题领域的描述性陈述。在实践中,要求需求公式完整或所有目标都可满足是不现实的。目标冲突分析[1,3]通过以下识别-评估-修复的过程来解决不一致性:

(1)识别阶段:识别导致不一致情况发生的一些条件;

(2)评估阶段:根据识别出的不一致情况的可能性和严重程度,对条件进行评估和排序;

(3)修复阶段:通过提供适当的对策来解决识别出的不一致情况。

3.2.1 目标冲突识别

本文关注弱不一致性,即分歧。分歧本质上代表了一个边界条件 BC,它的出现会导致目标的可满足性失效,即目标出现分歧[2]。

定义 2(边界条件) 假设$\{g_1,\cdots,g_n\}$是目标集合,$\{d_1,\cdots,d_m\}$是域属性集合。如果在目标集合和域属性集合下存在边界条件ϕ使得以下条件全部成立,则会发生分歧:

(1)$Dom\wedge G\wedge\phi\vDash\bot$(逻辑不相容性);

(2)$Dom\wedge G_{-i}\wedge\phi\nvDash\bot,1\leqslant i\leqslant n$(极小性);

(3)$\neg G\not\equiv\phi$(非平凡性)。

其中,$G=\wedge_{1\leqslant i\leqslant n}g_i$,$Dom=\wedge_{1\leqslant i\leqslant m}d_i$,$G_{-i}=\wedge_{1\leqslant j\leqslant n,j\neq i}g_j$。

直观地,BC 描述了目标无法被整体满足的特定情况。由于有极小性的限定,BC 不是$\bot$。本文可以通过调用 LTL 可满足性求解器来自动检查生成的 LTL 公式是否是 BC。

3.2.2 目标冲突评估

在评估阶段,为了给工程师推荐 BC,通常将其发生的概率作为一个重要指标[12]。对于没有概率信息的系统,可以用一种基于模型计数的方法[4]来分析 BC 的可能性和目标的严重程度。直观地,BC 的可能性越大,BC 表示的分歧更越可能发生。目标的重要性越小,违反目标的可能性就越大。因此,优先考虑可能性较大的 BC,优先分析严重性较小的目标。

3.2.3 目标冲突修复

在修复阶段,当系统达到 BC 出现的情况时,由于 BC 使系统发生故障,工程师需要一些策略来解决 BC 出现的分歧。直观地,解决分歧后,BC 导致的情况不会在由新的域属性和目标表示的系统下再发生。van Lamsweerde 等人[2]认为合理更新域属性和目标是一个开放问题,因为它需要大量的经验。

4 动机例子

在本节中,本文将以矿井泵[21]为例讨论 LBC 的特点。

例 1(矿井泵) 设有一个控制矿井内泵的系统,系统内有 2 个传感器:一个检测水位是否处于高点(h),另一个检测环境中是否存在甲烷(m)。如果水位高,水泵将被打开(p),以避免洪水。如果环境中出现甲烷,水泵将被关闭($\neg p$),以避免爆炸。域属性和目标通过以下 LTL 公式表示:

(1)域属性:

水泵作用(d_1)如式(7)所示:

$$\begin{aligned}&\square((p\wedge\bigcirc p)\rightarrow\bigcirc(\bigcirc\neg h))\\&\equiv\square((\neg p\vee\bigcirc\neg p)\vee\bigcirc(\bigcirc\neg h))\end{aligned}\tag{7}$$

当泵连续开启 2 个时刻,在接下来的 1 个时刻,水位将下降。

(2)目标:

防洪(g_1)如式(8)所示:

$$\square(h\rightarrow\bigcirc(p))\equiv\square(\neg h\vee\bigcirc(p))\tag{8}$$

每当水位高时,应在下一个时刻打开泵。

防爆炸(g_2)如式(9)所示:

$$\square(m\rightarrow\bigcirc(\neg p))\equiv\square(\neg m\vee\bigcirc(\neg p))\tag{9}$$

当出现甲烷,应在下一个时刻关闭泵。

在这种情况下,$\phi=\Diamond(p\wedge h\wedge\neg m\wedge\bigcirc(p\wedge h\wedge m\bigcirc(\square(m))))$是一个 LBC,其前缀集合为$\{p\wedge h\wedge\neg m,p\wedge h\wedge m\}$,后缀集合为$\{m\}$。$\phi$直观地描述了以下情况:未来,系统会先到达水位高、未检测到甲烷、打开泵的状态;然后,进入水位高、检测到甲烷、打开泵的状态;最后,进入检测到甲烷

的状态。直观地,前缀集合描述了特殊的前提条件。在这一前提条件下,系统进入水泵必须打开(g_1)和水泵必须关闭(g_2)的情况,即发生了分歧。

通过上面的例子,可以清楚地观察到,LBC 的第 1 个特点是可以解释分歧。通过 LBC,工程师可以清楚地知道分歧出现的原因是系统被前缀集合刻画的特殊前提条件所诱导,导致系统到达由后缀集合建模的状态。此外,LBC 的第 2 个特点是检查速度更快,因为其不需要检查非平凡性。

在识别阶段,LBCI 通过逐渐弱化 LTL 公式得到 LBC。具体过程如下文例 3 和例 4 所示。总体而言,LBCI 从$\Diamond(p\wedge\neg h\wedge m\wedge\bigcirc(p\wedge h\wedge m\wedge\bigcirc\Box((p\wedge h\wedge\neg m)\vee(\neg h\wedge\neg m))))$弱化得到$\Diamond(h\wedge m)$。$\Diamond(h\wedge m)$直观地描述了分歧的产生是由于未来的一个特殊状态:水位高并且检测到甲烷,导致了 g_1 和 g_2 无法在下一时刻同时被满足。本文称 $h\wedge m$ 为导致分歧的特殊前提条件。在评估阶段中,本文使用自动评估方法[4]来推荐 LBC 和解决冲突目标。假设评估后建议使用 LBC $\phi=\Diamond(h\wedge m)$和 $g_1=\Box(\neg h\vee\bigcirc(p))$来解决分歧。在修复阶段,由于前提条件 $h\wedge m$,一个简单的修复是加强 g_1 中下一时刻水泵打开($\bigcirc(p)$)的前提条件(h),即修复$\Box((\neg h\vee(h\wedge m))\vee\bigcirc(p))\equiv\Box(((h\wedge\neg(h\wedge m)))\rightarrow\bigcirc(p))$。

从修复分歧的过程可以看出,LBC 的第 3 个特点是指导修复,这缩短了合理修复的搜索时间,提高了修复质量。

5 套索边界条件

本节首先定义 LBC 的概念,然后提出 LBCI 以识别 LBC。

5.1 方法概述

为更具解释性地描述分歧,本文首先定义套索边界条件 LBC。

定义 3(前缀-后缀公式) 前缀-后缀公式是形如$\Diamond(t_0\wedge\bigcirc\mathcal{L})$的 LTL 公式,$\mathcal{L}$ 定义如式(10)所示:

$$\mathcal{L}::=\top\mid G(t_k\vee\cdots\vee t_n)\mid t_j\wedge\bigcirc\mathcal{L}\tag{10}$$

其中,t_i 是文字(Literal)的合取。本文称 $G(t_k\vee\cdots\vee t_n)$是后缀公式。令ϕ为前缀-后缀公式。ϕ的前缀集合由 $PREFIX(\phi)$递归计算得到,其定义如式(11)所示:

$$PREFIX(\phi)=\begin{cases}t_0\cup PREFIX(\phi_0),\phi=\Diamond(t_0\wedge\bigcirc\phi_0)\\\{t_i\}\cup prefix(\phi_i),\phi=t_i\wedge\bigcirc\phi_i\\\phi,\text{其他}\end{cases}\tag{11}$$

ϕ的后缀集合由后缀 $SUFFIX(\phi)$递归计算得到,其定义如式(12)所示:

$$SUFFIX(\phi)=\begin{cases}\{t_i,\cdots,t_n\},\phi=G(t_i\vee\cdots\vee t_n)\\SUFFIX(\phi_0),\phi=\Diamond(t_0\wedge\bigcirc\phi_0)\text{或}\\\qquad\phi=t_i\wedge\bigcirc\phi_i\\\phi,\text{其他}\end{cases}\tag{12}$$

本文用例 2 来说明定义 5。

例 2(续例 1) 前缀-后缀公式$\phi=\Diamond(p\wedge\neg h\wedge m\wedge\bigcirc(p\wedge h\wedge m\wedge\bigcirc\Box((p\wedge h\wedge\neg m)\vee(\neg h\wedge\neg m))))$的前缀集合是$\{p\wedge\neg h\wedge m,p\wedge h\wedge m\}$,后缀集合是$\{p\wedge h\wedge\neg m,\neg h\wedge\neg m\}$。

定义 4(套索边界条件) $\{g_1,\cdots,g_n\}$是目标集合,$\{d_1,\cdots,d_m\}$是域属性集合,目标集合和域属性集合下的套索边界条件ϕ是前缀-后缀公式的析取,并满足以下条件:

(1)$Dom\wedge G\wedge\phi\vDash\perp$(逻辑不相容性);

(2)$Dom\wedge G_{-i}\wedge\phi\nvDash\perp$,$1\leqslant i\leqslant n$(极小性)。

其中,$G=\wedge_{1\leqslant i\leqslant n}g_i$,$Dom=\wedge_{1\leqslant i\leqslant m}d_i$,$G_{-i}=\wedge_{1\leqslant j\leqslant n,j\neq i}g_j$。

由于 LBC 的特定结构,其必满足 BC 性质中的非平凡性。如果ϕ是 LBC,本文称ϕ的前缀-后缀公式的前缀集合为前提条件。直观地,前缀集合表示了特殊的前提条件,即系统可能发生的一些特殊的状态迁移;后缀集合表示系统被前缀集合刻画的迁移诱导后到达的状态集合。因此,LBC 描述了系统在前缀集合的前提条件下使系统陷入后缀集合描述的状态集合所产生的分歧。显然,LBC 直观地解释了分歧产生的机理。接下来,本文将介绍 LBC 的识别算法。

5.2 基于逐步弱化的 LBC 识别算法

在识别阶段,本文设计了 LBCI 来识别 LBC。首先,为前缀-后缀公式定义前缀弱化算子和后缀弱化算子。

定义 5(前缀弱化算子) ϕ为前缀-后缀公式,前缀-后缀公式的前缀弱化算子 $\rho^{p}_{\leq}$ 定义如式(13)所示:

$$\rho^{p}_{\leq}(\phi)=\begin{cases}\Diamond(t_1\wedge\phi_1),\phi=\Diamond(t_0\wedge\bigcirc\phi_0)\text{且}\\\qquad\phi_0=t_1\wedge\phi 1\\\phi,\text{其他}\end{cases}\tag{13}$$

其中,t_i 是文字的合取,ϕ_i 是一个 LTL 公式。

定理 1 设ϕ是一个前缀-后缀公式。如果$\phi'=\rho^{p}_{\leq}(\phi)$,则$\phi\rightarrow\phi'$。

定理 1 易证,因为$\Diamond(\phi\wedge\bigcirc\varphi)\rightarrow\Diamond(\varphi)$对于任何 LTL 公式$\phi$和 φ 成立且性质 1 成立。直观地,$\rho^{p}_{\leq}$通过依次减少前缀集合内的公式来弱化前缀-后缀公式。$\rho^{p}_{\leq}$需要保证前缀集合至少有一个元素,这就避免了前缀集合是空集。

定义 6(后缀弱化算子) 设ϕ为前缀-后缀公式。前缀-后缀公式的后缀弱化算子 $\rho^{s}_{\leq}$ 定义如式(14)所示:

$$\rho^{s}_{\leq}(\phi)=\begin{cases}\Diamond(t_0\wedge\bigcirc\rho^{s}_{\leq}(\phi_0)),\phi=\Diamond(t_0\wedge\bigcirc\phi_0)\\ t_i\wedge\bigcirc\rho^{s}_{\leq}(\phi_i),\phi=t_i\wedge\bigcirc\phi_i\ \text{且}\\ \qquad\phi_i\neq\Box\phi_j\\ \bigcirc\Box(t_i\vee\phi_i),\phi=t_i\wedge\bigcirc\Box\phi_i\\ \phi,\text{其他}\end{cases}\tag{14}$$

其中,t_i 是文字的合取,ϕ_i 是一个 LTL 公式。

定理 2 设ϕ是一个前缀-后缀公式,如果 $\phi'=\rho^{s}_{\leq}(\phi)$,则$\phi\rightarrow\phi'$。

定理 2 易证,因为$\phi\wedge\Box\bigcirc\varphi\rightarrow\bigcirc\Box(\phi\vee\varphi)$对于任何 LTL 公式$\phi$和 φ 成立且性质 1 成立。直观地,$\rho^{s}_{\leq}$通过依次减少前缀集合和添加后缀集合来弱化前缀-后缀公式。$\rho^{s}_{\leq}$还保证了前缀集合至少有一个元素。

本文使用弱化算子来设计 LBCI 算法。算法 1~算法 4 给出了 LBCI 的伪代码,其主要思想是迭代搜索满足 BC 逻辑不相容性的前缀-后缀公式(算法 1 的第 3 行),然后逐步弱化它们,促使它们满足 BC 的极小性(算法 1 的第 4 行)。在初始化和弱化过程中,LBC 记录了调用布尔可满足性问题 SAT(boolean SATisfiability problem)求解器的次数。如果在达到 SAT 求解器调用的某个阈值 η 后仍然无法获得 LBC,本文增加 k(算法 1 的第 8 行)。

算法 1 LBC 识别算法(LBCI)

输入:域属性 Dom,目标 G,阈值 η。

输出:LBC 集合 B_L。

```
1  B_L ← ∅, k ← case_k;
2  while time < TIMEOUT do
3     φ ← INITIALIZE(Dom, G, k);
4     φ* ← WEAKEN(Dom, G, φ, k);
5     if φ* is an LBC then
6        B_L ← B_L ∪ {φ*};
7     if B_L is not updated during η SAT calls then
8        k ← k + 1;
9  end while
10 return B_L
```

算法 2 初始化算法(INITIALIZE)

输入:域属性 Dom,目标 G,前缀-后缀公式个数 k。

输出:候选公式 ϕ。

```
1  while there are k models of ¬(Dom ∧ G) do
2     l ← K-RANDOMWALK(¬(Dom ∧ G));
3     φ ← build the disjunction of k prefix-suffix formulae based on l;
4     if φ does not satisfy the logical inconsistency under Dom and G then
5        continue;
6     else
7        return φ;
8  end while
9  return
```

算法 3 弱化算法(WEAKEN)

输入:域属性 Dom,目标 G,候选公式 ϕ。

输出:更新后的公式 ϕ^*。

```
1  for each prefix-sufix formula φ_i of φ do
2     φ'_i ← replace the suffix formula in φ_i with ⊤;
3     φ' ← replace φ_i in φ with φ'_i;
4     φ, φ* ← UPDATE(Dom, G, φ, φ*, φ');
5     while ⊤ do
6        if |SUFFIX(φ'_i)| = 0 then
7           break;
8        φ'_i ← ρ^s_≤(φ_i);
9        φ' ← replace φ_i in φ with φ'_i;
10       φ, φ* ← UPDATE(Dom, G, φ, φ*, φ');
11       if φ is not updated or |PREFIX(φ'_i)| = 1 then
12          break;
13    end while
14    while ⊤ do
15       φ'_i ← ρ^p_≤(φ_i);
16       φ' ← replace φ_i in φ with φ'_i;
17       φ, φ* ← UPDATE(Dom, G, φ, φ*, φ');
18       if φ' is not updated or |PREFIX(φ'_i)| = 1 then
19          break;
20    end while
21    for each term t_j in the prefix and suffix set of φ_i do
22       for each literal l_k in t_j do
23          t'_j ← replace the l_k in t_j with ⊤;
24          if the t'_j is not ⊤ then
25             φ'_i ← replace the t_j in φ_i with t'_j;
26             φ' ← replace φ_i in φ with φ'_i;
27             φ, φ* ← UPDATE(Dom, G, φ, φ*, φ');
28          end if
```

29 **end for**
30 **end for**
31 **end for**
32 **return** ϕ^*

算法 4 更新算法(UPDATE)

输入:域属性 *Dom*,目标 G,公式 ϕ,ϕ^*,φ。

输出:更新后的公式 ϕ,ϕ^*。

1 **if** ϕ does not satisfy the logical inconsistency under *Dom* and G **then**
2 **return** ϕ,ϕ^*;
3 **else if** φ does not satisfy the minimality under *Dom* and G **then**
4 **return** φ,ϕ^*;
5 **else**
6 **return** φ,φ;
7 **end if**

初始化的目的是生成满足 BC 逻辑不相容性的公式。为此,LBC 首先将 $\neg(Dom \wedge G)$ 作为 SPOT[28]求解器的输入并得到相应的自动机。LBC 在自动机上随机游走,采样 k 个不同的被自动机接受的轨迹,即满足 $\neg(Dom \wedge G)$ 的模型(算法 2 的第 1 行)。由于模型是套索的形式,因此很容易根据以下规则构建基于套索的前缀-后缀公式。假设 $\pi=s_0\cdots s_k(s_{k+1}\cdots s_m)^\omega$ 是一个套索,则其对应的前缀-后缀公式是 $\phi=\Diamond(t_{s_0}\wedge\bigcirc(\cdots\bigcirc(t_{s_k}\wedge\bigcirc\Box(t_{s_{k+1}}\vee\cdots\vee t_{s_m}))\cdots))$。显然,$\pi\models\phi$ 成立。LBC 基于 k 个不同模型形成公式 ϕ(算法 2 的第 3 行)。模型构建的公式(算法 2 的第 3 行)可能满 BC 的逻辑不相容性,因为它是 $\neg(Dom \wedge G)$ 模型的抽象。如果公式不满足 BC 的逻辑不相容性则继续搜索(算法 2 的第 5 行),否则 LBC 将其返回。本文用例 3 来说明初始化。

例 3(续例 1) $\neg(Dom \wedge G)$ 的一个模型为:$\pi=\{p,\neg h,m\}\{p,h,m\}(\{p,h,\neg m\}\{p,h,\neg m\}\{p,\neg h,\neg m\}\{\neg p,\neg h,\neg m\}\{p,h,\neg m\})^\omega$。

π 的前缀后缀公式是 $\phi=\Diamond(p\wedge\neg h\wedge m\wedge\bigcirc(p\wedge h\wedge m\wedge\bigcirc\Box((p\wedge h\wedge\neg m)\vee(\neg h\wedge\neg m))))$。$\phi$ 满足 BC 的逻辑不相容性。

在弱化步骤中,LBC 从初始化公式开始,经过如下 4 个步骤,逐渐满足极小性,从而找到 LBC:

(1)检查后缀集合的必要性(算法 3 的第 2~4 行)。

(2)弱化后缀集合(算法 3 的第 5~13 行)。

(3)弱化前缀集合(算法 3 的第 14~20 行)

(4)弱化前缀集合和后缀集合的状态(算法 3 的第 21~31 行)。

本文用例 4 来说明弱化过程。

例 4(续例 3) 令 $\phi=\Diamond(p\wedge\neg h\wedge m\wedge\bigcirc(p\wedge h\wedge m\wedge\bigcirc\Box((p\wedge h\wedge\neg m)\vee(\neg h\wedge\neg m))))$ 为初始化的结果,当检查后缀集合的必要性时,ϕ 和 ϕ^* 都更新为 $\Diamond(p\wedge\neg h\wedge m\wedge\bigcirc(p\wedge h\wedge m))$,因为 $\Diamond(p\wedge\neg h\wedge m\wedge\bigcirc(p\wedge h\wedge m))$ 也是一个 LBC。当弱化后缀集合时,LBC 不会更新 ϕ 和 ϕ^*,因为 $|SUFFIX(\varphi)|=0$。当弱化前缀集合时,ϕ 更新为 $\Diamond(p\wedge h\wedge m)$,因为 $\Diamond(p\wedge h\wedge m)$ 是一个 LBC。当弱化状态时,LBC 首先将 ϕ 更新为 $\Diamond(\top\wedge h\wedge m)$,因为 $\Diamond(\top\wedge h\wedge m)$ 也是一个 LBC。之后,LBC 依次检查 $\Diamond(\top\wedge m)$ 和 $\Diamond(\top\wedge h)$ 是否为 BC。由于它们都不是 BC,因此 ϕ 没有更新。显然,ϕ 从 $\Diamond(p\wedge\neg h\wedge m\wedge\bigcirc(p\wedge h\wedge m\wedge\bigcirc\Box((p\wedge h\wedge\neg m)\vee(\neg h\wedge\neg m))))$ 到 $\Diamond(\top\wedge h\wedge m)$ 的更新是一个逐渐弱化的过程。

定理 3 在算法 3 中,$\phi\rightarrow\phi'$ 是不变的。

通过更新算法 3 和算法 4 中的 ϕ 和 ϕ^*,以及性质 1,可以直接证明定理 3。定理 3 保证了 LBCI 的正确性:从满足 BC 逻辑不相容性的 LTL 公式出发,公式逐渐弱化,直到满足 BC 的极小性。

6 实验评估

本节在如表 1 所示的 16 个基准数据集[5]上进行了实验,以评估 LBCI 的有效性。本文提出的研究问题如下:

(1)LBC 的优势是什么?

(2)与 GA(Genetic Algorithm)[5]和 LOGION[9]相比,LBCI 效果如何?

本文将 LBCI 与 2 种最先进的方法 GA[5]和 LOGION[9]在识别 BC 方面进行了比较。GA 基于遗传算法识别任意 LTL 形式的 BC。LOGION 是一种针对任意 LTL 形式的 BC 的局部搜索算法。

本文 LBCI 实验设置如下:

(1)遵循了文献[5]中 GA 的配置,包括生成的初始种群大小和 50 代的限制,即遗传算法种群的 50 次进化。

(2)将 LBC 中的 η 值设置为 800,并调用 Aalta[33]作为 LTL 可满足性检查器,这与 GA 和 LOGION 相同。

(3)将模型计数设置为 1 000,文献[4]以该值获得了良好的准确性。

(4)将识别 BC 的时间限制为 1 h。

(5)运行10次取平均数据。

所有实验环境均为2.13 GHz Intel E7-4830,GNU/Linux (Ubuntu 16.04),128 GB内存。

Table 1 Overview of benchmark datasets

表1 基准数据集介绍

基准数据集	域属性个数	目标个数	变量个数	公式最大长度
RetractionPattern1 (RP1)	0	2	2	9
RetractionPattern2 (RP2)	0	2	4	10
Elevator (Ele)	1	1	3	10
TCP	0	2	3	14
AchieveAvoidPattern (AAP)	1	2	4	15
MinePump (MP)	1	2	3	21
ATM	1	2	3	22
Rail Road Crossing System(RRCS)	2	2	5	22
Telephone (Tel)	3	2	4	31
London Ambulance Service(LAS)	0	5	7	32
Prioritized Arbiter (PA)	6	1	6	57
Round Robin Arbiter (RRA)	6	3	4	77
Simple Arbiter (SA)	4	3	6	84
Load Balancer (LB)	3	7	5	85
LiftController (LC)	7	8	6	124
ARM's Advanced Microcontroller Bus Architecture (AMBA)	6	21	16	415

表2为LBCI识别BC的结果,其中,#BC表示BC的个数,#CBC表示contrasty BC[10]的个数。"v. s. GA"表示GA识别的BC中被LBCI识别的BC所覆盖的比例(>),反之亦然。"v. s. LOGION"的表示类似。对比结果中">"数值越高,"<"值越低,说明LBCI越好。如果求解器在10次实验中均失败,则结果将标记为"N/A"。

表2的结果表明,LBCI比GA更具可扩展性,LBCI可以解决14种情况,其中包括GA可以解决的10种情况。尽管LBCI识别出的BC数目比GA的略少,但在大多数情况下,LBCI获得的contrasty BC比GA的多。Luo等人[10]提出contrasty BC更值得推荐给工程师,因为contrasty BC描述了不同的分歧。综上,LBCI比GA能更好地搜索不同的分歧。

LOGION的可扩展性最好,识别的BC最多。但是,它获得的contrasty BC比LBCI的少,这说明了LBCI在识别质量上的优势。而且,通过分析LOGION识别的BC,发现它们都是BC的一种特殊形式,可以通过以下过程获得:$G=g_1,\cdots,g_n$是目标,首先本文设置$\phi=\neg g_1\vee\cdots\vee\neg g_n$,然后强化一些$\neg g_i$,这样本文就可以获得一种特殊形式的BC。而且,这种特殊形式的BC是没有意义的,因为其不能提供发生分歧的原因。

从表2的"v. s. GA"和"v. s. LOGION"可知,LBCI可以识别大多数情况下GA和LOGION可识别的分歧。在某些情况下,LBCI还可以识别GA和LOGION无法找到的分歧。这表明,LBCI是有效的,有表达大多数分歧的能力。

对于RQ 1,结果表明,LBC比任意形式的BC

Table 2 Results of identifying BC of LBCI

表2 LBCI识别BC实验结果

基准数据集	GA		LOGION		LBCI		v. s. GA		v. s. LOGION	
	#BC	#CBC	#BC	#CBC	#BC	#CBC	>	<	>	<
RP1	51.3	1.0	2 153.0	1	20.4	1.0	**1.00**	1.00	**1**	1.00
RP2	75.0	1.0	1 872.4	1	68.8	**5.3**	0	1.00	0	1.00
Ele	34.5	2.1	2 467.1	**7**	7.4	1.0	**1.00**	1.00	**1**	0.65
TCP	65.3	1.3	2 000.6	1	69.4	**4.5**	**0.95**	0.25	0	0
AAP	54.4	1.4	1 772.5	1	91.4	**7.1**	0.88	0.67	1	1.00
MP	113.7	1.0	2 078.2	1	1.2	1.0	**1.00**	1.00	0	1.00
ATM	99.3	**1.3**	2 309.0	1	12.4	1.0	0.86	1.00	**1**	1.00
RRCS	67.7	1.0	789.0	1	6.6	1.0	**1.00**	1.00	**1**	1.00
Tel	112.1	1.0	1 867.3	1	76.5	1.0	0	1.00	0	1.00
LAS	35.7	1.0	1 065.0	1	9.2	**3.5**	**1.00**	0.63	**1**	1.00
PA	N/A	N/A	947.8	1	1.4	**1.4**	0	0	**1**	0
RRA	N/A	N/A	1 400.8	1	7.4	**4.3**	0	0	**1**	0
SA	N/A	N/A	1 148.5	1	4.2	**1.1**	0	0	**1**	0
LB	N/A	N/A	789.0	1	0	0	N/A	N/A	N/A	N/A
LC	N/A	N/A	648.1	1	1.0	1.0	0	0	**1**	0
AMBA	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

具有更规则的模式来描述分歧,使工程师能够容易理解和分析。此外,LBC 可以表达大多数分歧。对于 RQ 2,LBCI 的输出数量明显小于其他求解器的输出数量,但仍然可以表示更多的分歧。这种简洁性大大降低了以后工作的复杂性。

7 讨论

7.1 有效性的威胁

LBCI 识别的 LBC 在某些情况下可以是几个前缀-后缀公式的析取,其形式较为复杂。当 BC 需要用多个前缀-后缀公式的析取才可以表示的时候,其可解释性会降低,需要分别考虑几个前缀-后缀公式的前缀集合情况。然而,结果(见表 2)表明,LBCI 在这种情况下仍然可以表现良好。

此外,本文的评估基于 16 种不同基准数据集[5],这些基准数据集包括各种来源的独立案例研究,并且规模大小不同,最大程度上代表了工程师在实际使用中的情况,减轻了有效性威胁。

7.2 正确性与完备性

本文使用求解器来确定 LBC,所以能够保证本文算法可以产生正确的分歧描述。对于完备性,由于本文算法采用了启发式搜索,加之任意的 LTL 公式可以表达空间不可穷举,因此本文算法的识别结果不是完备的。

8 结束语

在面向目标的需求工程中,分歧的目标冲突分析对软件质量和经济代价的影响都具有重要意义。本文提出了一种新的分歧描述——LBC,是 BC 的一种特殊形式,直观地解释了分歧产生的机制,并且设计出了高效的 LBC 的搜索算法——LBCI。最后,实验验证了 LBC 的优势,即可以解释分歧、求解高效、有利于指导修复。

在未来的工作中,将基于 LBC 研究其自动修复分歧的可能性,进一步优化 LBCI 以及将其扩展到一些工业数据集。

参考文献:

[1] van Lamsweerde A. Requirements engineering: From system goals to UML models to software specifications[M]. Hoboken: Wiley, 2009.

[2] van Lamsweerde A, Darimont R, Letier E. Managing conflicts in goal-driven requirements engineering[J]. IEEE Transactions on Software Engineering, 1998, 24(11): 908-926.

[3] van Lamsweerde A, Letier E. Integrating obstacles in goal-driven requirements engineering[C]//Proc of the 1998 IEEE/ACM International Conference on Software Engineering, 1998: 53-62.

[4] Degiovanni R, Castro P, Arroyo M, et al. Goal-conflict likelihood assessment based on model counting[C]//Proc of the 40th IEEE/ACM International Conference on Software Engineering, 2018: 1125-1135.

[5] Degiovanni R, Molina F, Regis G, et al. A genetic algorithm for goal-conflict identification[C]//Proc of the 33rd ACM/IEEE International Conference on Automated Software Engineering, 2018: 520-531.

[6] Degiovanni R, Ricci N, Alrajeh D, et al. Goal-conflict detection based on temporal satisfiability checking[C]//Proc of the 31st IEEE/ACM International Conference on Automated Software Engineering, 2016: 507-518.

[7] Huang Yi-hao, Feng Jin-cao, Zheng Han-yue, et al. A formal engineering method for requirement modeling of airborne embedded control software[J]. Computer Engineering & Science, 2019, 41(6): 1016-1025. (in Chinese)

[8] Hu Jun, Zhang Wei-jun, Li Wan-qian. A requirement oriented formal modeling and verification method for safety critical systems[J]. Computer Engineering & Science, 2019, 41(8): 1426-1433. (in Chinese)

[9] Zhong H, Wan H, Luo W, et al. Structural similarity of boundary conditions and an efficient local search algorithm for goal conflict identification[C]//Proc of the 27th Asia-Pacific Software Engineering Conference, 2020: 286-295.

[10] Luo W, Wan H, Song X, et al. How to identify boundary conditions with contrasty metric? [C]//Proc of the 43rd IEEE/ACM International Conference on Software Engineering, 2021: 1473-1484.

[11] Cailliau A, van Lamsweerde A. A probabilistic framework for goal-oriented risk analysis[C]//Proc of the 20th IEEE International Requirements Engineering Conference, 2012: 201-210.

[12] Alrajeh D, Kramer J, van Lamsweerde A, et al. Generating obstacle conditions for requirements completeness[C]//Proc of the 34th International Conference on Software Engineering, 2012: 705-715.

[13] Cailliau A, van Lamsweerde A. Integrating exception handling in goal models[C]//Proc of the 22nd IEEE International Requirements Engineering Conference, 2014: 43-52.

[14] Cailliau A, van Lamsweerde A. Handling knowledge uncertainty in risk-based requirements engineering[C]//Proc of the 23rd IEEE International Requirements Engineering Conference, 2015: 106-115.

[15] van Lamsweerde A, Letier E. Handling obstacles in goal-oriented requirements engineering[J]. IEEE Transactions on Software Engineering, 2000, 26(10): 978-1005.

[16] Felfernig A, Friedrich G, Schubert M, et al. Plausible repairs for inconsistent requirements[C]//Proc of the 21st Interna-

tional Joint Conference on Artificial Intelligence,2009:791-796.

[17] Murukannaiah P K,Kalia A K,Telangy P R,et al. Resolving goal conflicts via argumentation-based analysis of competing hypotheses[C]//Proc of the 23rd IEEE International Requirements Engineering Conference,2015:156-165.

[18] Cavezza D G,Alrajeh D. Interpolation-based GR (1) assumptions refinement[C]//Proc of International Conference on Tools and Algorithms for the Construction and Analysis of Systems,2017:281-297.

[19] Brizzio M,Degiovanni R,Cordy M,et al. Automated repair of unrealisable LTL specifications guided by model counting [J]. arXiv: 2105.12595, 2021.

[20] Li W,Dworkin L,Seshia S A. Mining assumptions for synthesis[C]//Proc of the 9th ACM/IEEE International Conference on Formal Methods and Models for Codesign,2011:43-50.

[21] Maoz S,Ringert J O,Shalom R. Symbolic repairs for GR (1) specifications[C]//Proc of the 41st IEEE/ACM International Conference on Software Engineering, 2019:1016-1026.

[22] Chatterjee K,Henzinger T A,Jobstmann B. Environment assumptions for synthesis[C]//Proc of International Conference on Concurrency Theory,2008:147-161.

[23] Pnueli A. The temporal logic in programs[C]//Proc of the 18th Annual IEEE Symposium on Foundations of Computer Science,1977:46-57.

[24] Sistla A P,Clarke E. The complexity of propositional linear temporal logics[J]. Journal of the ACM,1985,32(3):733-749.

[25] Cavada R,Cimatti A,Dorigatti M,et al. The nuXmv symbolic model checker[C] //Proc of the 26th International Conference on Computer Aided Verification, 2014:334-342.

[26] Li J,Zhu S,Pu G,et al. SAT-based explicit LTL reasoning [C]//Proc of the 11st International Haifa Verification Conference, 2015:209-224.

[27] Fisman D, Kupferman O, Sheinvald-Faragy S, et al. A framework for Inherent vacuity[C]//Proc of the 4th International Haifa Verification Conference, 2008:7-22.

[28] Duret-Lutz A,Lewkowicz A,Fauchille A,et al. Spot 2.0—A framework for LTL and ω-automata manipulation[C]//Proc of the 14th International Symposium on Automated Technology for Verification and Analysis,2016:122-129.

附中文参考文献:

[7] 黄怿豪,冯劲草,郑寒月,等.航空机载嵌入式控制软件需求建模的形式化工程方法[J].计算机工程与科学,2019,41(6):1016-1025.

[8] 胡军,张维珺,李宛倩.面向需求的安全关键系统形式化建模与验证方法研究[J].计算机工程与科学,2019,41(8):1426-1433.

作者简介:

罗炜麟(1992-),男,福建南平人,博士生,研究方向为需求工程和人工智能。**E-mail**:luowlin3@mail2.sysu.edu.cn

LUO Wei-lin,born in 1992,PhD candidate,his research interests include requirements engineering and artificial intelligence.

万海(1976-),男,江西南昌人,博士,副教授,研究方向为人工智能和形式化方法。**E-mail**:wanhai@mail.sysu.edu.cn

WAN Hai,born in 1976,PhD,associate professor,his research interests include artificial intelligence and formal method.

杨滨好(1997-),女,广东湛江人,硕士生,研究方向为需求工程。**E-mail**:yangbh7@mail2.sysu.edu.cn

YANG Bin-hao,born in 1997,MS candidate, her research interest includes requirements engineering.

李骁达(2000-),男,广东湛江人,研究方向为需求工程和人工智能。**E-mail**:lixd36@mail2.sysu.edu.cn

LI Xiao-da,born in 2000,his research interests include requirements engineering and artificial intelligence.

曹鉴恩(2000-),男,广东佛山人,研究方向为人工智能。**E-mail**:caojen@mail2.sysu.edu.cn

CAO Jian-en, born in 2000, his research interest includes artificial intelligence.

宋晓彤(1998-),女,黑龙江大庆人,硕士生,研究方向为需求工程。**E-mail**:songxt5@mail2.sysu.edu.cn

SONG Xiao-tong, born in 1998, MS candidate,her research interest includes requirements engineering.