软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2021,32(12):3728−3750 [doi: 10.13328/j.cnki.jos.006098]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

SSRules:让智能家居自动化规则更易于编写和检查*

王 博[1,2], 张 昱[1,2], 耿佳宁[1,2], 李向阳[1,2]

[1](中国科学技术大学 计算机科学与技术学院 下一代移动计算与数据创新实验室,安徽 合肥 230027)

[2](中国科学院 无线光电通信重点实验室,安徽 合肥 230027)

通讯作者: 张昱, E-mail: yuzhang@ustc.edu.cn

摘 要: 智能家居赋予家庭设备以智能,受到用户的广泛欢迎.由于用户需求不同,服务提供商采用“触发-动作”编程(TAP)模式以支持用户定制规则.然而,现在 TAP 编程和智能家居执行引擎中流行的 Event-State 时序范式极易出错,且难以修改规则和追踪运行错误.对TAP缺陷的原因进行系统分析之后,提出一种编写和修改难度较低、且能够检测规则运行异常的方案,记为 SSRules.SSRules 允许用户以一种改进的 State-State 时序范式输入规则,并基于 Z3 定理证明器将其翻译为 Event-State 时序范式,且为开源智能家居系统 Home Assistant 所接受的规则输入.考虑到智能家居需要实时掌握设备的动态,SSRules 引入了运行时子系统获取实体状态信息,并对规则执行有效性检查.最后,基于 Unity3D 开发了智能家居模拟器 HA-Simulator.测试结果表明:SSRules 与传统方法相比表达简洁,规则数目平均减少 60%左右,且能够及时检测瞬时异常并记录原因,更易被用户理解和使用.

关键词: 智能家居;触发-动作编程;终端用户编程;运行时系统;缺陷检测

中图法分类号: TP311

中文引用格式: 王博,张昱,耿佳宁,李向阳.SSRules:让智能家居自动化规则更易于编写和检查.软件学报,2021,32(12):3728−3750. http://www.jos.org.cn/1000-9825/6098.htm

英文引用格式: Wang B, Zhang Y, Geng JN, Li XY. SSRules: Make it easier to write and check automation rules for smart home systems. Ruan Jian Xue Bao/Journal of Software, 2021,32(12):3728−3750 (in Chinese). http://www.jos.org.cn/1000-9825/6098.htm

SSRules: Make it Easier to Write and Check Automation Rules for Smart Home Systems

WANG Bo[1,2], ZHANG Yu[1,2], GENG Jia-Ning[1,2], LI Xiang-Yang[1,2]

[1](LINKE, School of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China)

[2](Key Laboratory of Wireless-Optical Communications, Chinese Academy of Sciences, Hefei 230027, China)

Abstract: Smart home systems make home devices smart and are widely welcomed by users. Due to different user needs, service providers use “trigger-action” programming (TAP) mode to support user-tailored rules. However, the Event-State paradigm, which is now popular in TAP programming and smart home rule engines, is highly error-prone, and the modification of the rules and the tracking of errors are difficult. After systematic analysis of the causes of TAP defects, a scheme with low difficulty in writing and modification and being able to detect abnormal rule operation is proposed, denoted as SSRules. SSRules allows users to enter rules written in improved State-State paradigm, and SSRules can translate them into rules written in Event-State paradigm and acceptable by the open-source smart home system Home Assistant based on the Z3 Theorem Prover. Considering that smart homes need to master the dynamics of the device in real-time, SSRules introduces a runtime subsystem to obtain state information and perform rule execution validity checks. Finally, a smart home simulator HA-Simulator is developed in Unity3D. Tests on it show that SSRules is more concise than traditional methods, the number of rules is reduced by around 60% on average. It can detect transient anomalies promptly and record the cause, which is easier for users to understand and use.

* 基金项目: 国家重点研发计划(2018YFB0803400); 国家自然科学基金(61772487); 安徽省自然科学基金(1808085MF198)

Foundation item: National Key Research and Development Program of China (2018YFB0803400); National Natural Science Foundation of China (61772487); Anhui Provincial Natural Science Foundation (1808085MF198)

收稿时间: 2020-02-21; 修改时间: 2020-04-28; 采用时间: 2020-06-16

Key words: smart home; trigger-action programming; end-user programming; runtime system; bug detection

随着物联网、人工智能与 5G 等创新技术驱动的万物互联时代的到来,智能家居作为物联网的重要应用场景之一,正快速进入千家万户.智能家居令家庭中的各种设备互联互通,使用户实现设备的自动控制、远程控制和可编程控制,甚至可以根据历史经验主动进行自动化服务.根据 MarketWatch 发布的报告[1],2016 年,全球智能家居市值 556.5 亿美元,预计 2025 年将达到 1 742.4 亿美元;同时,中国的智能家居市场也在快速发展,据艾瑞咨询发布的报告[2],2017 年,中国智能家居市场规模为 3 254.7 亿元,预计 2020 年将达到 5 819.3 亿元.

为管理种类和规模不断增加的智能设备,国内外主要领导厂商纷纷推出智能家居平台,如三星的 SmartThings、Amazon 的 Alexa、Google 的 Home、小米的米家、华为 HiLink、云端规则定制平台 IFTTT(“IF This Then That”)[3]等,但它们几乎不开源并且仅支持相关品牌的设备,限制了用户所能管控的设备类型及范围;在开源的智能家居平台中,基于 Python3 的 HomeAssistant[4]相对成熟,支持多种操作系统/平台,集成了 IFTTT、GoogleAssistant、MQTT 等产品/功能.这些智能家居平台将要管控的设备抽象化,通过建立通信标准以及 API 互联等方式连接设备和 app,采用“触发-动作编程”(trigger-action programming,简称 TAP)[5]支持用户定制规则以指定系统行为.但我们通过多次调研发现:现有 TAP 虽足以满足终端用户表达简单的自动化任务,但灵活性不足,难以支持更复杂的用例[5−8].

TAP 规则(如 IFTTT)典型地将单个触发器(trigger)与单个动作(action)关联起来,例如“如果开始下雨,则关窗”.然而,许多常见行为需要 TAP 提供更强的表达能力,如对空调等复杂设备的设定、对门锁窗户等安全攸关设备的可靠设定、对夜灯/咖啡机等设备的工作持续时间设定等.为支持用户的日常需求,不同的智能家居平台提供不同的支持,例如引入 and 连接多个触发器[5]或者连接多个触发器和多个动作(如 Stringify[9])、用更一般的 and/or 组合多个触发器或多个动作(如 HomeAssistant)、增加对触发器的条件过滤(如 Microsoft Flow[10],Zapier[11]和 HomeAssistant)、增加“状态保持时长”的表达能力(如 SmartRules[12])、支持自定义的事件/服务(如 HomeAssistant).各种 TAP 扩展丰富了可定制性,但是也存在语义不清晰、编写复杂等缺陷,不便于用户使用.

随着设备数量的增加,其交互也随之增多,TAP 编程出错的可能性也会增加.针对这类问题,一些研究如 BuildingRules[13],IRuler[14]等提供对某些 TAP 编程缺陷的检测;AutoTap[15]能由线性时序逻辑表达的设备性质合成某些 TAP 规则,从而避免用户直接编写 TAP 规则;AutoTap 和 Menshen[16]还提供有限的错误修复功能.Brackenbury 等人[17]首次提出和规范可用于 TAP 规则触发器的 3 种时序范式 Event-Event,State-State 和 Event-State,并总结了 TAP 可能存在的 10 种编程缺陷.

针对智能家居系统面临的挑战,本文研究易写易改、支持多触发器-多动作、支持状态保持描述等能力的 TAP 规则的表示,然后构建支持这种规则表示的智能家居自动化框架.在 TAP 规则的表示方面,触发器和动作均可以分为(纯)状态型和事件型.本文首先系统分析和总结了各种 TAP 编程范式以及它们引起的缺陷种类、原因和检测/修复现状,指出了改进 TAP 的可能渠道;然后,针对用户不易分清事件和状态、不易写全规则所要考虑的情况组合以及“Event-trigger Event-action”(EE)风格的规则冗长、更改琐碎等特征,本文提出以“State-trigger State-action”(SS)为基础构建面向终端用户易写易改的、具有丰富表达能力的具体 SS 规则范式.在自动化框架构建方面,鉴于 HomeAssistant(简称 HA)相对成熟和开源,支持多种平台和多种设备及服务集成方式,提供以 Event 为核心的宽泛 TAP 配置,本文以它为基础构建支持 SS 规则范式的智能家居自动化框架 SSRules.针对 Brackenbury 等人[17]指出的 10 种 TAP 缺陷,采用 SS 规则范式能天然排除时间窗错误和翻转触发器这 2 种缺陷,完全避免优先级冲突;结合 SSRules 的实现策略,能部分解决或消除安全默认偏差、非瞬时动作、重复触发这 3 种缺陷,并附带改善缺少反向动作的缺陷;对于无限循环和不确定时序这两种缺陷,可以通过模型检查等方法在规则转译成 HA 之前避免;SSRules 尚不能解决矛盾的动作这种缺陷.本文的主要贡献如下:

(1) 系统分析和总结了 TAP 编程范式及其编程缺陷,指出了改进 TAP 的几种可能渠道,为我国智能家居领域从业者和软件研发人员全面了解 TAP 编程、缺陷及改进提供了参考资料;

(2) 提出以“State-trigger State-action”为基础构建易写易改、有丰富表达力的 TAP 编程范式.SS 规则范式

以实体-能力抽象为基础,按动作实体将规则分组、组内按序优先,有效避免了优先级冲突;它区分只读和可控能力,提供 4 种表示历史状态的算符,并方便表达在指定状态/状态保持下对多种能力的期望;

(3) 引入能表达事件型智能家居执行引擎的独立“Event-trigger Event-action”(EE)中间表示,基于 HA 构建智能家居自动化框架 SSRules,它能将 SS 规则集合经 EE 规则表示转译成 HA 规则,利用 Z3 定理证明器[18]进行触发事件的筛选和规则化简,并提供运行时子系统动态感知设备变化和检查规则执行的有效性;

(4) 实现了智能家居模拟器和 SSRules 原型系统.对 10 组测试场景,用 SS 范式编写所需的规则条数比转译和合并后得到的 EE 范式规则数平均减少了 60%左右;在模拟实验环境下,最终生成的 HA 规则的运行时检查都未发现连续异常,并且所记录到的瞬时异常(小于总检查次数的 0.3%)经查阅日志均为网络延迟引起.

本文第 1 节对现有 TAP 编程范式及其引起的缺陷、易用性以及缺陷检查和修复现状进行总结.第 2 节提出智能家居自动化框架的设计目标,对以 HA 为底层基础、以 SS 规则范式为用户输入的智能家居自动化框架 SSRules 的总体设计进行总结.第 3 节详细讨论 SS 规则范式的定义及其到 HA 规则的转译方法.第 4 节介绍基于 HA 构建的智能家居模拟环境,并在该模拟环境上评估了 SSRules 的有效性.第 5 节和第 6 节分别对相关工作和本文的工作进行总结.

1 TAP 编程范式及缺陷分析

为了满足终端用户灵活多样的需求,大多数智能家居平台都提供特定的 TAP 编程接口用于对智能设备和/或服务进行编程,以定制智能家居系统的行为.本节首先对 TAP 编程范式进行了详细分析,然后系统总结分析了可能导致 TAP 缺陷的原因,最后简要总结改进 TAP 的可能渠道.

1.1 TAP编程范式概述

TAP 的表达能力及易写易改性,对智能家居的推广和普及至关重要.目前最流行的 TAP 接口是 IFTTT 在线服务[3],它允许用户创建程序、自动执行由单个事件触发的动作,尽管 IFTTT 应用广泛,但其表达能力非常受限.

- 多触发器、多动作

根据 Ur 等人的用户调查[5],22%的编程行为需要不止一个触发器或动作,并且触发器和动作在用户期望的行为中的组合也更为多样化.因此,Ur 等人引入“and”连接触发同一动作的多个事件,例如“IF 窗户是开的 and 在下雨”就触发“关窗”.Zapier[11]允许将多个动作绑定到一个触发器;Stringify[9]支持形如“If This and This, Then That and That”的规则.HA[4]还允许定制只要多个触发器中任意一个被触发就执行的规则.

- 条件过滤与工作流控制

Zapier 和 HA 提供过滤器,支持对触发器的进一步(and/or)条件过滤.例如:可以在触发器“接收邮件”后追加过滤器“主题包含 X and 来自 Y”;MicrosoftFlow[10]不仅提供多步骤的工作流,还提供 If-then-else 条件过滤、For-Each 和 Do-While 循环等;HA 允许定制包含“call service”动作、自定义事件和自定义触发器的规则,允许开发者编程实现各种服务.Flow 和 HA 提供的这些功能非常强大,但是对终端用户编程的能力要求高,且缺少对用户配置的有效性检查.

- 区分事件(event)与状态(state)

IFTTT 的“if this then that”易被理解为通用编程中的 if-then 语句,而其实际语义却等价于事件驱动编程.Huang 等人[6]认为:把 IFTTT 隐喻成 if-then 会有歧义,这种歧义会给用户带来困难.由此提出用事件和状态区分触发器:事件是瞬时信号(如“温度降到 10°C 以下”),而状态是可随时评价的布尔条件(如“在下雨”).Brackenbury 等人[17]进一步将动作分成事件(在特定时刻发生,如“关灯”)和状态(一段时间内设备的期望状态,如“灯应亮着”)两类.这种区分事件和状态的 TAP 看似消除了一些歧义,潜在简化了系统实现中对 TAP 规则的理解,但是研究表明:终端用户往往很难清晰区分事件和状态[6],且它们的组合又引出新问题.

- 事件和状态的组合

事件和状态在组合时有以下语义:多个状态 s_i 的合取是另一个状态 $\boldsymbol{s}$,仅当每个 s_i 为真时 $\boldsymbol{s}$ 为真;状态 $\boldsymbol{s}$ 与事件 $\boldsymbol{e}$ 的合取是在 s 为真时 $\boldsymbol{e}$ 同时发生的另一个事件 e';两个事件 e_1 和 e_2 的合取是与 e_1 和 e_2 同时发生的另一个

事件.然而在智能家居系统中,“事件同时发生”无论是理论还是实际都不太可能发生.Brackenbury 等人[17]提出了 3 种多触发器组合的时序范式:Event-Event,Event-State 和 State-State,并将它们与动作组合成表 1 所示的 4 类 TAP 规则.其中:Event-Event 时序范式引入 WITHIN 时间窗口子句来描述不能同时发生的事件,并提供 AFTERWARDS 指定事件次序;Event-State 时序范式将触发器限制为一个带若干状态的事件.这两类触发器均触发事件型动作,也是真实智能家居系统主要采取的方式.State-State 触发器范式组合多个状态并且状态会在一段时间内满足,故自然地触发状态动作;考虑到影响同一设备的多条规则的多个状态可能同时为真(如控制空调的模式),故 State-State→State 范式需要包含优先级来解决冲突.此外,对于少数不能直接表示为状态型的离散事件动作(如“记录日志”),State-State 时序范式会触发事件动作.

- 状态保持性描述

Huang 等人[6]将动作分成瞬时动作(如“发邮件”)、在固定时间完成的动作(如“冲泡咖啡”)、包含改变状态的持续性动作(如“开灯”),这给智能家居系统提出了描述和实现状态保持特性的需求.SmartRules 提供“状态保持”的表达,可以创建“如果门开着有 5 分钟就通知”之类的动作.HA 提供的 trigger 可以定时,虽然其 action 没有直接提供保持时长的表示方法,但是它提供的 callservice 动作潜在允许开发者编写处理状态保持时长等服务的能力,因此也可以实现类似状态保持的表达.

Table 1 Four kinds of TAP rules defined by Brackenbury *et al*. and examples

表 1 Brackenbury 等人定义的 4 类 TAP 规则范式以及规则举例

范式类别	范式定义	规则举例
Event-Event→**Event**	IF event (AND event|AND AFTERWARDS event)*(WITHIN timewindow) THEN event	IF 主人离家 AND AFTERWARDS 外卖送达 WITHIN 5 分钟 THEN 通知主人
Event-State→**Event**	IF event (WHILE state (AND state)*)* THEN event	IF 主人进入卧室 WHILE 晚上 THEN 打开卧室灯
State-State→**State**	IF state (AND state)* THEN state PRIORITY priority	IF 小明不在家 AND 小黄不在家 THEN 空调应处于关闭状态 PRIORITY 0 IF 室内温度高于 28°C THEN 空调应处于制冷模式 PRIORITY 1
State-State→**Event**	IF state (AND state)* THEN event	IF 室内温度高于 30°C THEN 记录温度到日志文件

1.2 TAP规则的缺陷分析

用户在编写 TAP 规则时很容易出错,原因在于智能家居设备众多、用户期望的行为复杂多样、智能家居设备之间存在大量的交互[19],单个设备会直接或间接影响到其他设备.人的行为也会影响设备状态,而人的即使看似简单的行为也可能会引起背后复杂的设备行为;且人的行为与需求并非一成不变,在不同的场景需求下,很可能会造成对已有规则的破坏[20].由于智能家居的用户通常缺乏编程相关的训练,为降低编程门槛,TAP 过分简化了操作界面[6],从而使程序的表达能力变差,这进一步恶化了 TAP 对复杂行为的可解释性.

Brackenbury 等人[17]通过用户调查,总结出可能出现在 TAP 的 10 种缺陷,包括:

(1) 4 种缺陷已被发现且详细讨论

① 优先级冲突(priority conflict),仅在采用 State-State→State 时需要对作用于同一设备的多条规则进行优先级排序,而用户往往难以按自己的意图正确设置规则优先级[21].这种缺陷已被 TAP 文献广泛讨论[7,13,22−24],可以通过静态分析来检测;

② 安全默认偏差(secure-default bias)发生在用户默认系统处在安全状态[21](例如夜间锁住窗口),但广泛部署的 Event–State 设备没有默认状态,Yarosh 等人详细讨论了门锁的这种缺陷[21];

③ 非瞬时动作(extended action)源于动作发生在一段时间而不是瞬时的(如冲泡咖啡);

④ 缺少反向动作(missing reversal)发生在用户创建了一条执行某动作的规则而没有写撤销该动作的规则时,例如写了“IF 进客厅 THEN 开灯”但忘写离开关灯的规则,但用户可能认为系统会自动关灯[6,21].

Huang 等人[6]详细讨论了缺陷③和缺陷④,并对系统接口提出改进建议.缺陷④可以通过静态分析检测,但

不总能被修复.

(2) 3 种缺陷被提及

⑤ 无限循环(infinite loop)的原因是规则相互触发,这种缺陷在机器人终端用户编程中常被遇到[6];

⑥ 重复触发(repeated triggering)指用户希望规则仅触发一次,但却触发多次.这种缺陷主要出现在 State-State 范式下因状态触发器持续为真导致规则触发多次.Cao 等人[25]在 mashup 编程工作中提及了这种缺陷;

⑦ 不确定的时序(nondeterministic timing)是由于系统处理同时发生的触发器的顺序不确定而导致.Huang 等人[6]简要讨论了 TAP 中存在的这种缺陷.

这 3 种缺陷均可以通过静态分析来检测,但是实际系统无法鉴别重复触发是否是用户的有意行为.缺陷⑤和缺陷⑦还可以通过动态分析检测.

(3) TAP 新增缺陷

⑧ 矛盾的动作(contradictory action)指的是长周期内在矛盾的动作间无限循环(例如加热和冷却之间不断交替);

⑨ 时间窗错误(time-window fallacy)是当用户在编写 Event-Event 范式的规则时忘指定时间窗时导致;

⑩ 翻转触发器(flipped triggers)是因用户难以指定触发器中哪部分是事件哪部分是状态而导致的.这种缺陷在静态或动态分析中都很难检测到.

表 2 总结了上述 10 种缺陷的原因、引起缺陷的 TAP 编程范式、缺陷检查或修复方法.

Table 2 Causes and detection of various bugs in TAP rules

表 2 各种 TAP 缺陷的起因与可检测性

缺陷类型	缺陷原因	引起缺陷的 TAP 范式	缺陷检测/修复	在 SSRules 中的情况	
① 优先级冲突	A 用户难以正确设置规则的优先级	State-State 触发器	静态检测	无	规则按动作实体分组,组内靠前的规则优先级高
② 安全默认偏差	A 用户认为设备默认在安全状态,但却不是	All	在特定领域嵌入系统默认	部分解决	系统提供可配的安全默认
③ 非瞬时动作	B 时序缺陷	State-State 触发器	系统提示、不提供混淆的选项、更多 TAP 结构等	部分解决	在 State-State→Event 时才有,SS 范式不提供用户选择动作事件而由 SSRules 根据指定的期望状态来判断执行前提
④ 缺少反向动作	A 用户认为会自动撤销某些动作	All	系统提示,可静态检测,但不总能修复	辅助改善	SSRules 处理②③的对策可以附带改善这种缺陷
⑤ 无限循环	C 控制流缺陷	All,规则相互触发	静态或动态分析	可检测	检测实体依赖关系是否有环
⑥ 重复触发	C 控制流缺陷	All, State 型触发器因状态持续为真而多次触发	可静态分析是否重复	部分消除	消除那些会出错的重复触发
⑦ 不确定时序	B 时序缺陷	All	静态或动态分析	提示	检测实体依赖关系
⑧ 矛盾的动作	C 控制流缺陷	All,在矛盾的动作间循环	难以检测	未解决	存在,无法检测
⑨ 时间窗错误	A 用户忽略时间窗的设置	Event-Event 触发器	可以静态检测	无	不存在
⑩ 翻转触发器	A 用户分不清事件和状态	Event-State/Event 触发器	检测标准不好确定	无	不存在

10 种缺陷中,第①种、第②种、第④种、第⑨种、第⑩种归结为与用户预期不一样,第③种、第⑦种为时序缺陷,剩余 3 种为控制流缺陷.表中第 3 列的“All”表示表 1 中的 4 类 TAP 规则都可能引起这种缺陷.从中可见:第①种和第③种这两种缺陷仅存在于纯状态触发器的 TAP 编程中,而第⑨种、第⑩种仅存在于含事件的触发

器的 TAP 编程中.Brackenbury 等人[17]对这 10 种缺陷的进一步用户调查表明:相比 Event-State,问卷参与者使用 State-State 能写得更正确,而使用 Event-Event 则正确性要差些.

1.3 改进TAP的可能渠道

现有的 TAP 平台尚未能充分满足用户需求,在实际运行中容易出现缺陷导致系统出错.综合分析现有智能家居系统及学术研究成果,以下给出了几种改进 TAP、减少编程缺陷的渠道.

(1) TAP 编程范式

实际智能家居系统大都基于 Event-State→Event 规则范式来实现.相比于使用事件型触发器进行 TAP 编程,使用纯状态触发器更容易让终端用户正确表达意图.即使使用相同的触发器范型和动作范型,不同的 TAP 规则结构仍会影响 TAP 编程的表达能力、易用性和编程缺陷.现有解决方案中,除了直接让终端用户写 TAP 规则,研究人员还提出了不同的实现方案,如:

- AutoTap[15]允许用户输入设备的安全性质(可转化为线性时序逻辑公式)而由系统合成 Event-State→Event 规则,从而减少最终规则的缺陷,但是存在合成效率较低、难以适用较多设备、安全性质不适合表达需求细节等不足;
- Zave 等人[24]提出让用户对设备按不同目的(如安全性、节能等)分别描述需求特征,然后手工为特征建立状态机,并根据它设计运行时的特征组合机制来实时控制执行器.如何由非形式的特征描述建立 FSM 以及解决特征冲突(或称特征交互)是其中的难点,该方法目前只支持对几种设备的人工建模且仅支持“值设置”动作.

(2) TAP 规则的检查与修复

改进 TAP 编程范式不可能完全消除编程缺陷,TAP 平台仍需要一些额外的工具或方法来检查 TAP 规则的有效性,并尝试修复规则.现有的方法大都针对 Event-State→Event 规则进行静态检查(如 IRuler[14],AutoTap[15],MenShen[16,22])、动态检查(IoTGuard[19])或规则修复(AutoTap,MenShen,TrigGen[26]),仅有极个别方案是基于状态触发的[21,24],这很大程度上是因为实际的智能家居系统是按事件驱动方式来运行的.表 3 从多种维度比较了已有方法,可以看出,Event-State→Event 规则的修复存在搜索修复策略低效以及在不清楚用户意图的情况下仍需用户决策等局限性.而随着设备数增多,这类规则及修复逻辑的可读/可理解性也在下降.此外,由于网络延迟/重放攻击等,也会使基于 EE 的智能家居系统执行不期望的动作.Wang 等人[27]提出,通过分析日志中事件间的依赖来事后追因.

Table 3 Existing work towards bugsin smart home rules

表 3 关于智能家居用户规则缺陷的相关工作

	AutoTap[15]	IRuler[14]	MenShen[16]	TrigGen[26]	IoTGuard[19]	FIResolution[24]
工具类型	修复和合成	静态检查	修复	修复	动态检查	运行时特征组合
支持的平台	自定义	自定义	自定义	openHAB	SmartThings	AT&T HA
用户输入	用户规则和安全性质	用户规则	用户规则	用户规则	运行时策略	特征描述
支持的范式	Event-State→Event	Event-State→Event	IFTTT	Event-State→Event	Event-State→Event	状态触发器
减少缺陷的做法	检查用户指定的安全性质,并添加规则来修复	用重写逻辑检查如绕过条件、动作循环或重复等 10 种通用逻辑缺陷	模型检查设备的安全性质,修改规则来修复	生成必要和充分的触发器修复,如触发器缺失和矛盾的动作等	运行时检查用户配置的 36 种安全策略	运行时特征组合实时控制执行器
局限性	算法复杂度高,难以适用较多设备	处理的缺陷类型有限,且模型只适用于 IFTTT	支持的输入范式受限	只能处理部分触发器缺失缺陷且规则脚本必须有触发器的引用	要在 Applet 中插桩,特殊情况下安全策略和阻塞会有不良后果	操作繁琐,需人工建模几种设备的特征

(3) 智能家居系统的实现

对 TAP 规则的静态检测和修复可以在规则实际运行前识别出部分规则缺陷,并尝试手工/自动修复.但是如表 2 第 4 列所说,还有许多缺陷是静态无法检测,或是由于难以判断用户意图从而无法确定缺陷是否存在.因此,智能家居系统在实现时需要提供:

1) 给用户更多的提示和警告:针对那些潜在有歧义、有冲突或已做动作不会被自动撤销等情况,系统要给出用户易于理解的提示或警告;
2) 提供不易让用户混淆的选项供用户选择;
3) 提供运行时日志的记录,需要权衡日志的存储量与事后推理的便利性;
4) 支持和提供含义更明确的顶层 trigger-action 结构,等等.

2 SSRules 智能家居系统总体设计

为使终端用户易于编程,本文重点探索状态触发器-状态动作的 TAP 编程范式(简称 SS 规则范式),并基于 HA 构建了支持 SS 规则范式输入的 SSRules 系统.SSRules 在不改变 HA 规则执行引擎的基础上,扩展提供更易被用户理解和掌握的规则表达方式以及对应的规则转译器.虽然 SS 规则范式尚不支持表达无状态或难以抽象出状态的事件动作,但由于这类动作所占比重小、用户需求低,因此,SSRules 可以支持用户的绝大多数需求.未来可以设计专门的机制,将这些无状态的事件动作加入到 SSRules 系统中.本节首先简要介绍 SSRules 的设计目标,然后分析 HA 的关键特征,最后给出 SSRules 总体架构以及所引入的 Entity-Capability 抽象.

2.1 设计目标

智能家居正进入寻常百姓家,为了让 SSRules 满足用户的不同需求,现阶段至少应围绕如下目标设计.

G1. 易写易改.针对用户调查[5,6,15,17]反映的 TAP 规则编写和修改的问题,SSRules 应提供让无编程经验的终端用户快速学习并正确使用的编程范式,方便表达所期望的智能家居系统性质,且在需求变化时容易修改;

G2. 可管可控.SSRules 系统应能提供按终端用户的有效需求对来自不同厂商的设备实时监测与控制,感知系统中动态加入和退出的管控对象,而不能只支持特定厂商的设备和仅提供极其有限的监测能力;

G3. 异常检测.智能家居系统的可靠性受限于网络、设备故障等不定因素,而用户希望系统提供实时反馈以及对系统更多的控制感.因此,系统需要能够帮助用户监视运行状况,并将异常信息反馈给用户;

G4. 协调需求.作用于同一设备的多种需求存在冲突,是智能家居中的常见现象.系统要提供简单的处理方式,便于用户协调好同一设备存在冲突的多种需求.例如希望灯在夜间关闭,但是有人经过则应以较低亮度打开一段时间,如果遇到紧急情况(如厨房着火)则应当完全打开;

G5. 多方信息.智能家居系统的功能边界应当不限于家庭的物理空间.系统不仅要能够处理家庭中存在的物理设备,也要能处理虚拟传感器和虚拟的设备,如情景模式、天气、时钟等.

2.2 HomeAssistant(HA)

本文的研究重点为探索 TAP 编程的表示及其在现有执行引擎的实现,同时提供一定的规则有效性检查功能.因此,为更有效地实现 G2 和 G5 涉及的目标,SSRules 需要选择成熟开源的智能家居执行引擎来实现与各种(虚拟)设备的连接、实时监控、动态感知以及自动化管理等.

在开源智能家居系统中,HA 是基于 Python 的成熟智能家居开源系统,能部署在任何能运行 Python 3 的机器上,从树莓派到网络存储,还可以使用 Docker 部署到其他系统上.它主要根据 automations.yaml 等配置文件实现集中化的智能家居设备管理,设备支持度高,具有自动化、群组化、UI 客制化等高度定制化设置;同时,其开发者数量、版本更新速度在所有开源项目中也都属于佼佼者.因此,我们最终选取 HA 作为 SSRules 的执行引擎.

- 灵活的集成方式

在 HA 中,物理设备和虚拟设备均被抽象为实体,带有状态的实体可以与状态对象关联.天气、太阳角度等

均为 HA 预置的虚拟实体;通过用户自定义或第三方集成,还可以实现如模式开关、空气质量、用户位置等虚拟传感器和动作器.实体上可以执行的动作被抽象为服务调用,除了 HA 自带的值变化事件、服务调用事件等事件类型,第三方集成也能够发布其他自定义的事件.采用 HA 可以自然地支持目标 G5.

- 设备的动态管理

HA 接入设备的方式之一是 MQTT Discovery,它提供对基于 MQTT(message queuing telemetry transport)协议通信的设备的自动发现、配置和移除.MQTT 协议是一种基于 TCP 的发布订阅协议,在物联网通信中使用较多.设备接入 HA 时,每个设备只需按照 HA 指定的格式配置其 MQTT 通信模块,HA 即可间接从 MQTT 代理获得配置信息并完成配置流程.若设备需要移除,则它可以更新状态信息将自己移除,也可以由 HA 根据超时设置自动移除.因此,HA 提供实现 G2 的基础,SSRules 需要主动与 HA 交互真正达到 G2.

- 高自由度的输入

HA 的规则输入语言为 Trigger-Condition-Action(即 Event-State→Event)范式,单规则中可以含有多触发器、多动作.动作不仅可以是与设备相关的服务调用,也可以是自定义的事件触发和状态值写入、HTTP 接口调用等,自由度很高.但是 HA 的自动化规则的编写非常繁琐,用户难以使用其描述有优先级的功能需求,并且用户自行推理编写极易遗漏或写错.此外,由于 HA 的实体状态和服务调用之间的松耦合,HA 提供运行时异常检查极其有限.HA 在 G1,G3 和 G4 方面较弱,因此,这也是 SSRules 要重点解决的问题.

2.3 SSRules系统总体架构

SSRules 系统总体架构如图 1 所示.整个系统通过 HA 提供的接口与智能家居设备连接,包括通过 MQTT 代理连接智能家居真实设备或模拟器.系统由离线的转译器子系统和在线的 SSRules 运行时子系统组成:前者提供终端用户输入的 SS 规则到 HA 规则的转译,后者提供 HA 抽象信息的获取和规则执行的有效性检查.SSRules 引入 Entity-Capability 抽象(见第 2.4 节)来描述智能家居系统所能管控的设备(实体)及可管控的内容(能力),用户输入的 SS 规则最终被 SSRules 转化为 HA 规则注入到 HA 的配置文件并得以执行.

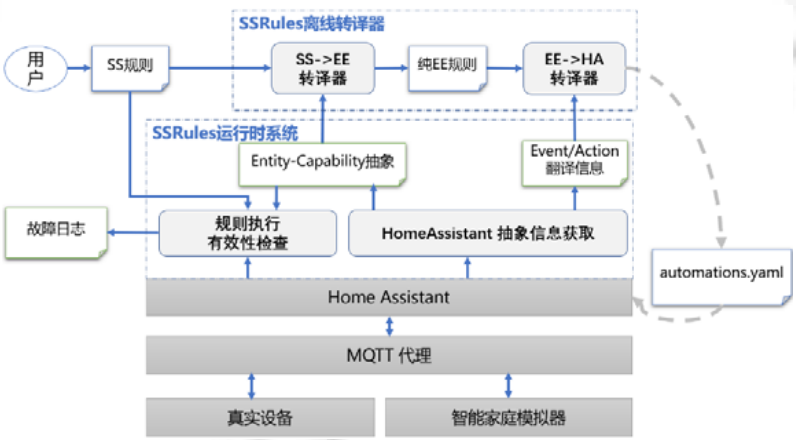


Fig.1 Overall structure of SSRules

图 1 SSRules 系统总体架构

- 抽象信息获取

SSRules 需要 HA 中的实体、状态、服务的抽象信息用于 SS 规则的转译和异常检测,这些信息由 SSRules 运行时子系统的抽象信息获取模块来提取和更新.该模块定期从 HA 更新实体信息并检测变化情况,获取设备的动态信息(加入或离开),生成 SSRules 所需的最新实体-能力抽象信息和动作翻译信息.

- 规则转换流程

SSRules 的用户交互模块根据实体-能力抽象信息将用户输入规则所需的信息以用户友好的方式呈现在前

端界面中.用户完成编辑后,用户交互模块根据用户的输入生成 SS 规则文件.接着,SSRules 运行时系统会调用可离线运行的转译器将 SS 规则转译成 HA 规则,再将其更新到 HA 的 automations.yml 中.

- 异常检测设计

当由 SS 规则翻译得到 HA 规则并开始执行之后,规则执行有效性检查模块会从 HA 获取所关心的状态变化事件,定期获取全部实体的状态,检查实体状态及其状态变化是否符合 SS 规则的描述,并将因网络/设备故障等问题导致的设备状态与 SS 规则描述不相符等情况进行告警以及写入日志.

2.4 Entity-Capability抽象

一个智能家居系统所管理的设备可以包括真实物理设备(如空调)、虚拟设备(如天气、时钟等),还可以是人.SSRules 系统使用实体-能力抽象(entity-capability abstraction)来描述智能家居系统中各种设备的可感知和/或可控制的能力,记为 $W=\{\mathcal{E}_1,\mathcal{E}_2,...,\mathcal{E}_{N_D}\}$,$N_D$ 为实体总数.每个设备抽象为一个实体 $\mathcal{E}$(entity),每个实体由其 ID $\boldsymbol{e}$、一组能力 $\mathbb{C}$(capability)以及描述各能力状态之间转移关系的状态机 M 组成,即 $\mathcal{E}=(\boldsymbol{e},\mathbb{C},M)$.

- 能力

每种能力表示单个可变化的只读或者可控属性.表 4 列出了几种常见智能家居设备及其能力.每种能力可表示为一个四元组 $(c,k,ro,\mathcal{J})$,其中:c 是该能力在实体内的唯一标识;k 表示该能力的状态值类型,可以是二值(binary)/多值(set)/实数(numeric)等类型之一(实数可以细分成子界类型或实数区间等,本文为简便起见不作区分);布尔量 ro 为真表示能力是只读的、为假表示是可控的;$\mathcal{J}$ 表示一组关联的命令序列,其中每个元素是要执行的命令代号.

- 状态机

每个实体的状态机由该实体每种能力 $\boldsymbol{c}$ 的状态机组成:$M_{\boldsymbol{c}}=(V_{\boldsymbol{c}},\mathbb{T}_{\boldsymbol{c}})$,其中,$V_{\boldsymbol{c}}$ 是能力 $\boldsymbol{c}$ 的状态值集合,$\mathbb{T}_{\boldsymbol{c}}$ 是状态值之间的转移函数.

- 若 $\boldsymbol{c}$ 为只读的,则转移函数为 $\mathbb{T}_{\boldsymbol{c}}:V_{\boldsymbol{c}}\xrightarrow{[C]}V_{\boldsymbol{c}}$,箭头上方的 C 为事件转移发生的必要条件,表示能力 $\boldsymbol{c}$ 仅在 C 为真时才可能从源状态值转移到目的状态值;
- 若 $\boldsymbol{c}$ 为可控的,则转移函数的类型为 $\mathbb{T}_{\boldsymbol{c}}:V_{\boldsymbol{c}}\xrightarrow{[C]\mathcal{J}}V_{\boldsymbol{c}}$,箭头上方的 C 为动作转移允许发生的充分必要条件;$\mathcal{J}$ 为要执行的命令代号,表示能力 $\boldsymbol{c}$ 仅在 C 为真时才能够接受代号为 $\mathcal{J}$ 的动作,或者在出现外部事件时(如用户通过遥控器、手机 APP、物理开关等控制),使得从源状态值转移到目的状态值.

Table 4 Examples of entities and their capabilities

表 4 实体及能力举例

	Capability	取值范围	性质		Capability	取值范围	性质
空调	模式	制冷/制热/抽湿/送风/关闭	可控	人体传感器	状态	激活/未激活	只读
	设定温度	16°C~30°C	可控	咖啡机	开关	打开/关闭	可控
	室温	–30°C~70°C	只读		咖啡状态	准备好/未准备好	只读
	风扇模式	开/关	可控	夜灯	开关	打开/关闭	可控
	扫风模式	开/关	可控		亮度	0~255	可控
电扇	开关	开/关	可控	时钟	时间	24 小时制的 HH:MM	只读
	风速	低/中/高	可控		状态	白天/夜间	只读
门锁	状态	锁定/未锁定	可控	模式	状态	回家模式/离家模式	只读

在缺省的 $M_{\boldsymbol{c}}$ 中,任意两个状态值之间存在条件为 True 的转移(即无条件状态转移);若 $\boldsymbol{c}$ 可控,则存在无条件动作,它可触发任意两个状态值之间的状态转移.SSRules 系统会提供初始的 $M_{\boldsymbol{c}}$ 配置文件,其中提供部分常用设备能力及其状态转移描述,用户也可以根据需要添加某个设备或者某一类设备的 $M_{\boldsymbol{c}}$ 补充信息.

根据 $\boldsymbol{c}$ 的 $M_{\boldsymbol{c}}$ 抽象,可以进一步检测动作的执行前提,或者判断事件的可能性.例如,假设电扇和咖啡机的能力状态机如图 2 所示:① 电扇的 $\boldsymbol{c}_2$(风速)只有在其 $\boldsymbol{c}_1$(开关)的取值为开时才可以接受改变风速的动作;② 咖啡机的咖啡状态仅在咖啡机的 $\boldsymbol{c}_1$(开关)取值为开时才能从未准备好进入准备好状态.

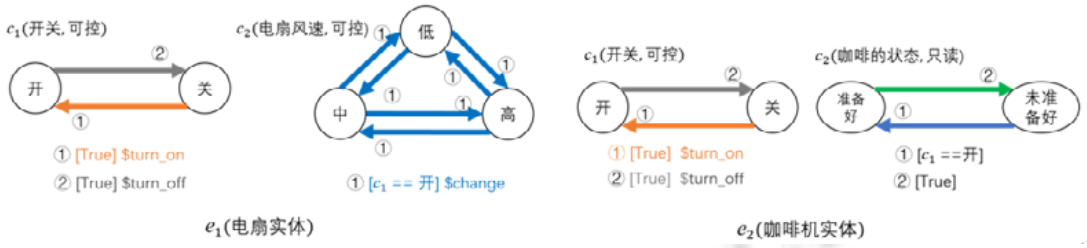


Fig.2 The state machines for each capability of the fan and coffee machine

图 2 电扇和咖啡机的能力状态机

3 SSRules 编程范式及其到 HA 规则的翻译

本节首先分析以状态触发器-状态动作(SS)风格设计 SSRules 编程范式的可行性;然后给出 SSRules 提供给终端用户使用的 SS 规则范式的定义,分析它在应对表 2 所列的 10 种 TAP 缺陷的能力;接着分析转译器面对的主要问题,并引入 EE 中间范式作为 SS 规则到 HA 规则转换的桥梁;最后介绍 SS 到 EE 转译的关键算法.

3.1 SS范式的引入

由第 1 节可知:终端用户不太能区分事件和状态,并且用户使用 State-State 时序范式会比使用含 Event 触发器的范式更易写对规则.为此,我们以状态为基础,通过规则分组、自动优先级等改进措施(详见第 3.2 节)来提供“State-trigger State-action”编程范式,简称 SS 范式.

- SS 范式的易写易改性

SS 范式可以较为简洁地表达用户日常常见需求;并且和 Event-State→Event 相比,SS 范式写出的规则容易随需求的变更和细化进行修改.为了更清晰地表述这两种范式的区别,表 5 给出了分别用两种范式描述夜灯开关的规则,其中,模式是 HA 提供的虚拟传感器,有回家模式和离家模式两种取值.在 SS 范式下,作用于夜灯的 3 条规则均作为子句按序置于“ FOR 夜灯”语句中,每个子句形如“EXPECT A [WHILE B]”,表示当 B 成立(或永真)时,期望 A 中各能力为指定的值;对于同一实体的多个 EXPECT,SSRules 按从前往后以第一个 WHILE 条件成立的 EXPECT 子句所指定的期望状态为准.然而,当用 Event-State→Event 表示时,为了正确表达所列出的几种情况的状态保持和它们之间的转移关系,需要写出 9 条规则来细分情况.假设用户用 SS 范式先写了后 2 条规则,运转一段时间后他又添加第 1 条保证离家模式下夜灯总是关闭的需求,在 SS 范式下,这种添加是简单的;但是若使用 Event-State→Event 达到相同目的,则需要在原先第 3 条~第 6 条、第 8 条、第 9 条这 6 条规则基础之上新增 3 条规则,并修改原先的规则细节(修改部分用粗体字表示),这样的编写和修改非常琐碎、易错.

Table 5 Example of comparing SS rules and Event-State→Event rules

表 5 SS 范式与 Event-State→Event 范式的比较举例

SS 范式	Event-State→Event 范式
FOR 夜灯 1. **EXPECT(开关,关) WHILE 模式.状态==离家模式** 2. EXPECT(开关,开) WHILE 人体传感器.状态==激活 AND 时钟.状态==夜间 3. EXPECT(开关,关)	1. **IF 进入离家模式 WHILE 灯开启 THEN 关闭夜灯** 2. **IF 进入回家模式 WHILE 人体传感器激活 AND 夜间 THEN 打开夜灯** 3. IF 变为白天 WHILE 灯开启 THEN 关闭夜灯 4. IF 变为夜间 WHILE 人体传感器激活 THEN 打开夜灯 5. IF 人体传感器信号变为激活 WHILE 晚上 **AND 回家模式** THEN 打开夜灯 6. IF 人体传感器变为未激活 WHILE 晚上 **AND 回家模式** THEN 关闭夜灯 7. **IF 夜灯变为打开 WHILE 离家模式 THEN 关闭夜灯** 8. IF 夜灯变为关闭 WHILE **回家模式 AND** 人体传感器激活 AND 夜间 THEN 打开夜灯 9. IF 夜灯变为打开 WHILE **回家模式 AND** 人体传感器激活 OR 白天 THEN 关闭夜灯

- SS 范式的局限

SS 范式能表达大多数智能家居的场景需求,但是无法表达无状态或者难以抽象出状态的事件和动作(如无

状态按钮、记录日志等).对于这类需求,目前仍需编写 Event-State→Event 规则.因此,SSRules 将 SS 范式转译为 Event-State→Event,并允许两种范式共存.未来可以扩展降低这类需求编程难度的具体范式.

3.2 SS范式的定义

表 6 的左部列出了 SSRules 系统支持的 SS 范式的抽象语法,其中:FOR 语句集结同一动作实体的所有规则,且约定组内靠前的规则优先级更高;每条规则(ssrule)包含在 WHILE 子句指定的条件成立时,对该实体全部或部分能力的期望状态;多条规则按从前到后的次序、以第 1 条满足 WHILE 条件的规则中的 EXPECT 子句来设置期望的实体能力状态;一般地,无 WHILE 子句的规则放在最后作为安全默认.Condition 中的“entityID.capName op value”,“historyOp timeperiod entityID.capName value”分别表示当前状态和历史状态型原子判断,其中,历史状态运算符 historyOp 的 4 种可能取值的含义举例如下.

- “WITHIN 5 分钟电扇.开关关”表示电扇的开关在 5 分钟内曾经处于关闭状态,则该原子判断为真;
- “!WITHIN 5 分钟电扇.开关关”表示电扇的开关在 5 分钟内不曾处于关闭状态,则该原子判断为真;
- “STAY 5 分钟电扇.开关开”表示电扇的开关在 5 分钟内持续处于开启状态,则该原子判断为真;
- “!STAY 5 分钟电扇.开关关”表示电扇的开关在 5 分钟内未一直处于开启状态,则该原子判断为真.

Table 6 Abstract syntax of paradigms of SS rules and EE rules used in SSRules

表 6 SSRules 系统使用的 SS 范式和 EE 范式的抽象语法

State-State(SS)范式	Event-Event(EE)范式
ssrulesGroup:=**FOR** entityID ssrule+ ssrule:=**EXPECT** capState+ **WHILE** condition capState:=(capName,Value) condition:=condition **OR** condition |condition **AND** condition |**NOT** condition |(condition) |entityID.capName op value |historyOp timePeriod entityID.capName value op:=‘==’|‘!=’|‘<’|‘>’|‘≤’|‘≥’ historyOp=**‘STAY’|‘WITHIN’|‘!STAY’|‘!WITHIN’** timePeriod:=时长	eerule:=**IF** event+ [**WHILE** condition] **THEN** eeAction+ event:=entityID.capName **FROM** range **TO** range |entityID.capName **STAYON** value **REACH** timeperiod eeAction:=entityID.capName.cmdName value condition:=condition **OR** condition |condition **AND**condition |**NOT** condition |(condition) |entityID.capName op value |historyOp timePeriod entityID.capName value op:=‘==’|‘!=’|‘<’|‘>’|‘≤’|‘≥’ historyOp=**‘STAY’|‘WITHIN’|‘!STAY’|‘!WITHIN’** timePeriod:=时长 range:=值集

3.3 SS范式的缺陷避免能力分析

SS 范式天然地排除了表 2 中所列的缺陷⑨、缺陷⑩这两种缺陷.对于在使用 State-State 时序范式时可能存在的缺陷①~缺陷⑧这 8 种缺陷,SS 范式的表示机制能完全避免缺陷①优先级冲突;SS 范式的表示及转译机制能部分解决或消除缺陷②安全默认偏差、缺陷③非瞬时动作、缺陷⑥重复触发,其附带效果可以辅助改善缺陷④缺少反向动作;对于缺陷⑤无限循环、缺陷⑦不确定时序等缺陷的检测,可以通过模型检查等方法在规则转译之前避免;由于难以提供通用的鉴别何谓矛盾的机制,因而 SSRules 不能解决缺陷⑧矛盾的动作.下面分别讨论 SS 范式应对缺陷①~缺陷③和缺陷⑥的措施.

(1) “按动作实体将规则分组、组内规则有序”可完全避免缺陷①优先级冲突

用户只需移动规则在组内的相对位置,免除显式设置优先级数值的困难.由于同一动作实体的各规则优先级不同,故无缺陷①(示例见 3.1 节).

(2) “将配置的安全默认规则自动作为动作实体的最低优先级规则”可辅助消除缺陷②安全默认偏差

SSRules 提供可配的安全默认设置文件,其中的条目形如“实体/实体类 ssrule”可描述指定实体或实体类的默认期望状态,SSRules 会自动将其追加到相应实体的规则集尾部,作为最低优先级规则.例如针对门锁可以配置默认安全规则“EXPECT (状态,锁定)”,则用户只需描述其他需求,如:

FOR 门锁 EXPECT (状态,未锁定) WHILE 人体传感器.状态==激活 [AND 其他安全条件]

而 SSRules 会自动在尾部追加“EXPECT (状态,锁定)”.只要人体传感器不再激活,SSRules 就会按最后一条

无条件子句的期望状态将门锁锁定.这种做法消除缺陷②的前提是要正确配置安全默认文件.

如果用户希望在人体传感器激活后 2 分钟内均保持门锁解锁,可使用 SS 范式提供的历史状态判断:

FOR 门锁 EXPECT (状态,未锁定) WHILE WITHIN 2 分钟人体传感器.状态激活 [AND 其他安全条件]

(3) “通过避免用户直接编写动作”可避免缺陷③非瞬时动作和缺陷⑥重复触发

在 SSRules 中,状态转移所需要的动作的执行前提被描述在实体-能力抽象信息中,用户只能选择期望的状态,而无法写出有缺陷的程序造成对非瞬时动作的多次触发动作而进入不期望的状态.例如:针对冲泡咖啡这样的非瞬时动作,如果主人希望每天 7:00 让咖啡机工作,且咖啡机工作完成后自动关闭,7:00~7:20 之外的时间不开启咖啡机,则用 SS 范式编写的规则为:

FOR 咖啡机

EXPECT (开关,开) WHILE 时钟.时间>7:00 AND 时钟.时间<7:20 AND 咖啡机.咖啡状态==未准备好

EXPECT (开关,关)

当咖啡机开启后,即使咖啡未准备好,SSRules 转译后的 HA 规则也保证不会再出现开启动作.而用户倘若用 State-State→Event 范式,容易写出如下有缺陷的规则,造成咖啡机多次启动:

IF 咖啡未准备好 AND 时间处于 7:00~7:20 之间 THEN 启动咖啡机

对于重复触发缺陷,能够用 SS 范式表达的规则可以用上述类似的机制来避免;而不能够用 SS 范式表达的规则,仍需要使用 Event-State→Event 或者 Event-Event→Event 范式来写(未来将对此提供更直观的编程方式).

3.4 转译器面对的问题及EE中间范式的引入

- 转译器面对的问题

如果要在一个仅支持 Event-State→Event 范式的智能家居系统上实现对 SS 范式规则的支持,需要将智能家居系统状态之间的转移和状态转移带来的新的状态设定与智能家居系统所支持的事件和动作关联起来,并且要尽可能避免重复触发、非瞬时动作等缺陷.这一翻译过程应该对用户不可见,是由系统自动完成的.此外,为了从翻译的主要算法中排除 HA 规则描述的琐碎细节,以及让算法一般化以支持到其他平台的移植,SSRules 引入了 EE 中间范式(基于 Event-State→Event).从而:转译器的第 1 阶段,SS→EE 的翻译可以避开 HA 的语法及实现细节,重点是基于同一套实体-能力抽象将 SS 范式表示的功能需求转化为 EE 范式所表示的具体做法;而第 2 阶段,EE→HA 的翻译则只需从较为抽象的 EE 规则根据 HA 的事件/条件描述方式、动作对应表,生成 HA 所需的规则细节即可.

- EE 中间范式

表 6 的右部列出了 SSRules 系统转译中使用的 EE 中间范式的抽象语法,其主要特点为:一条规则可以含有多个触发事件(event)、条件过滤(condition)和多个按顺序执行的动作(eeAction).event 包括状态值变化事件“entityID.capName FROM range TO range”和状态保持事件“entityID.capName STAYON value REACH timeperiod”:前者表示给定实体能力的取值从一个值集变迁到另一个值集,后者表示指定的实体能力保持给定值 value 的时间长达 timeperiod 时触发该事件.目前,SSRules 仅对二值类型的能力提供状态保持事件.EE 范式中条件过滤 condition 的定义与 SS 范式一样.

- EE 规则到 HA 规则的转译

EE 规则到 HA 规则的翻译是一一对应的,EE 规则的状态值变化事件可以对应到 HA 的 state_changed 事件和在 HA 规则的 condition 中对事件数据中的 old_state 和 new_state 的限制条件.EE 规则中的状态保持事件可以转译为在 HA 中的延时执行和自定义事件触发.EE 规则中的条件表达式可以对应到 HA 条件表达式中的相同树形结构.EE 规则中,实体能力的取值对应到 HA 规则中的状态对象 state 或者状态对象 attribute 中的附加状态;EE 规则中,对实体能力的历史状态判断被对应到 HA 规则中的自定义状态对象取值判断;EE 规则中的一个或多个动作被单个或组合对应到 HA 中无参或带参的服务调用.

3.5 SS规则集与EE规则集的符号表示与一些定义

为了方便后文的说明和理解,我们引入表 7 中的符号以及几个基本概念,这些符号和表 6 定义的语法结构一一对应.例如:G^S 对应于语法定义中的一个 ssrulesGroup,R^S 对应于一个 ssrule,R^S 中的 C 和$\mathcal{X}$分别对应于 WHILE 子句中的条件表达式和 EXPECT 子句中的期望状态组.wt 和 st 分别表示“WITHIN”和“STAY”.

Table 7 Symbolic representation of SS ruleset and EE ruleset

表 7 SS 规则集合和 EE 规则集合的符号表示

(SS 范式的程序)	$\mathbb{G}^S$	::=	$\{G_1^S,...,G_{N_S}^S\}$	(EE 范式的程序)	$\mathbb{R}^E$	::=	$\{R_1^E,...,R_{N_E}^E\}$
(实体的 SS 规则组)	G^S	::=	$(\boldsymbol{e},\mathbb{R}^S)$	(EE 规则)	R^E	::=	$(\mathbb{E},C,\mathbb{A})$
(SS 规则序列)	$\mathbb{R}^S$	::=	$(R_1^S,...,R_k^S)$	(事件集合)	$\mathbb{E}$	::=	$\{E_1,\ldots,E_r\}$
(SS 规则)	R^S	::=	$(C,\mathcal{X})$	(事件)	E	::=	$E^I\|E^H$
(期望状态组)	$\mathcal{X}$	::=	$\{x_1,\ldots,x_m\}$	(动作序列)	$\mathbb{A}$	::=	$(A_1,\ldots,A_n)$
(期望状态)	x	::=	$(\boldsymbol{c},v)$	(动作)	A	::=	$(\boldsymbol{e},\boldsymbol{c},\mathcal{J},v)$
(能力的状态值)	v	∈	V	(状态变化事件)	E^I	::=	$(\boldsymbol{e},\boldsymbol{c},V_s,V_d)$
(条件)	C	::=	C or $C\|C$ and $C\|!C\|(C)$	(状态保持事件)	E^H	::=	$(\boldsymbol{e},\boldsymbol{c},v,t)$
			$\|(\boldsymbol{e},\boldsymbol{c},O,v)\|(\boldsymbol{e},\boldsymbol{c},O_h,t,v)$	(运算符)	O	::=	$=\|\neq\|<\|>\|\leqslant\|\geqslant$
(时长)	t		带单位秒、分钟或小时等	(历史状态运算符)	O_h	::=	wt|!wt|st|!st

对于一个由 N_D 个实体组成的智能家居系统 $W=\{\mathbb{E}_1,\mathbb{E}_2,\mathbb{E}_{N_D}\}$,每个 $\dot{\mathbb{E}}_i$ 由三元组$(\boldsymbol{e},\mathbb{C},M)$描述.在某个时刻下, W 的系统状态记为σ,它是各实体的各能力到状态值的映射,即$\sigma(\boldsymbol{e},\boldsymbol{c})$表示实体 e 的能力 $\boldsymbol{c}$ 在该时刻的状态值.不论是 SS 范式的程序$\mathbb{G}^S$,还是 EE 范式的程序$\mathbb{R}^E$,都是对系统状态变迁的描述;而转译的目标是由$\mathbb{G}^S$得到$\mathbb{R}^E$,使得$\mathbb{G}^S$与$\mathbb{R}^E$满足一致性.在定义一致性之前,我们先定义系统抽象状态ω,然后定义 SS 规则与系统抽象状态ω的关系,包括规则在ω下是否激活、规则是否与ω兼容.

定义 1(系统抽象状态). 对于规则引擎来说,系统当前状态σ、当前时刻τ_{now}以及状态变化轨迹$\mathbb{T}$可抽象为一个三元组$\omega=(\sigma,\tau_{now},\mathbb{T})$,称之为系统抽象状态.其中,$\mathbb{T}=((\boldsymbol{e}_1,\boldsymbol{c}_1,v_1,\tau_1),\ldots,(\boldsymbol{e}_n,\boldsymbol{c}_n,v_n,\tau_n))$,$(\boldsymbol{e}_i,\boldsymbol{c}_i,v_i,\tau_i)$表示实体 e_i 的能力 $\boldsymbol{c}_i$ 在τ_i 时刻变为 $v_i(i\in\{1,\ldots,n\})$,它只记录状态值发生变化的特定实体的特定能力及其新状态.状态变化可能由外部事件引起(如用户交互、环境变化),也可能由$\mathbb{G}^S$或$\mathbb{R}^E$执行动作序列引起.每当发生状态变迁时,都相当于$\mathbb{T}$更新为$\mathbb{T}\uplus(\boldsymbol{e}_{n+1},\boldsymbol{c}_{n+1},v_{n+1},\tau_{n+1})$,其中,$\tau_{n+1}$为状态变迁发生时的$\tau_{now}$.

定义 2(规则在系统抽象状态ω下激活). 假设 $G^S=(\boldsymbol{e},\mathbb{R}^S)$为某动作实体 e 的 SS 规则组,$\mathbb{R}_{\mathcal{X}}^S$ 表示$\mathbb{R}^S$中期望状态组均为$\mathcal{X}$的规则子集,当前的系统抽象状态为ω.称期望状态组相同的一组规则 $\mathbb{R}_{\mathcal{X}}^S$ 在ω下激活,是指该组内任意一条规则 R^S 激活;而规则 R^S 在ω下激活,是指在该规则所在的规则组 G^S 内,其前面的规则的条件(对应 WHILE 子句,即 C)在ω下都未满足,而该规则的条件 $R^S.C$ 满足.

定义 3(规则与系统抽象状态ω兼容). 规则R^S与系统抽象状态ω不兼容,是指在ω下 R^S激活且$R^S.\mathcal{X}$尚不成立;而 R^S 与ω兼容,是指在ω下 R^S 未激活或者 $R^S.\mathcal{X}$已成立;一组规则与ω兼容,是指组内每条规则与ω都是兼容的;一组规则与ω不兼容,是指组内存在与ω不兼容的规则.

实际的规则引擎提供的时间戳的精度是有限的.假设系统抽象状态ω中的时间戳都有最小时间单位(例如 1 秒),τ_{now}以最小时间单位离散地更新.那么,对于规则集合执行方式的假设如下.

- SS 规则执行假设.假设无时序缺陷的规则组$\mathbb{G}^S$的执行方式为:每当发生状态变化,或者τ_{now}变为新值之后,判断$\mathbb{G}^S$中是否存在与最新的ω不兼容的规则.如果存在这样的规则 R^S,说明其对应实体 e 不满足 R^S 的期望状态组$\mathcal{X}$,于是计算 e 上的需要执行的动作序列$\mathbb{A}$,使得$\mathbb{A}$执行后实体 e 的状态符合$\mathcal{X}$;
- EE 规则执行假设.假设无时序缺陷的规则组$\mathbb{R}^E$ 的执行的方式为:设$\mathbb{R}^E$ 中的所有触发事件可分为状态

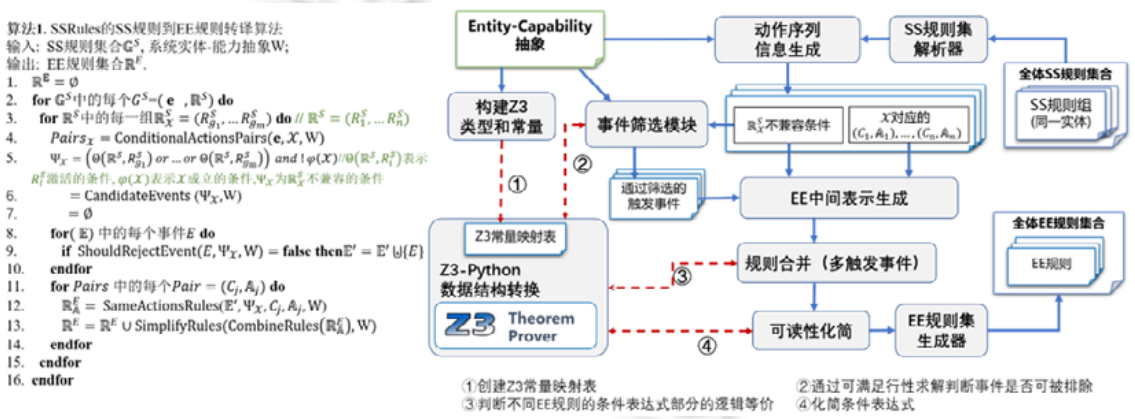
变化事件 E^I 和状态保持事件 E^H 这两种类型,每当出现与 E^I 相符的状态变迁,或者每当 τ_{now} 的更新触发了 E^H 中的某个状态保持事件后,检查与触发事件相符的各条规则.如果存在 R^E,$R^S.C$ 在 ω 下成立,则执行 $\mathbb{R}^E.\mathbb{A}$.

定义 4(SS 与 EE 的一致性). 称 $\mathbb{G}^S$ 与 $\mathbb{R}^E$ 一致是指:对于无时序缺陷的 $\mathbb{G}^S$,$\mathbb{R}^E$,在上述规则执行假设下(且不存在超时、执行失败、规则重叠执行的情形),从与 $\mathbb{G}^S$ 兼容的任意系统抽象状态 ω 开始,对于同样的、同时刻的、一次发生一个事件的外部事件序列,$\mathbb{G}^S$ 所产生的状态变迁序列和 $\mathbb{R}^E$ 所产生的状态变迁序列一致.

与 $\mathbb{G}^S$ 一致的 $\mathbb{R}^E$ 不唯一,原因在于:① $\mathbb{R}^E$ 中不会被执行的规则不影响一致性;② 一个条件较弱的 EE 规则(如一条规则 R_1^E,条件 $R_1^E.C$ 为时钟.状态==白天)可等价于多个条件更强的 EE 规则(如等价于规则 R_2^E 和 R_3^E,条件 $R_2^E.C$ 和 $R_3^E.C$ 分别为“时钟.状态==白天 AND 模式.状态==回家模式”和“时钟.状态==白天 AND 模式.状态==离家模式”);③ 一条 EE 规则 R^E 还可以等价于 m 条 EE 规则 $R_i^E(i=1,...,m)$,满足 $R^E.\mathbb{E}=\cup_i R_i^E.\mathbb{E}$;④ 逻辑表达式有多种等价表述.

3.6 SS规则到EE规则的转译

图 3 详细说明了 SS 规则到 EE 规则的转译算法实现,其中:图 3(a)中的算法 1 是总控算法;图 3(b)是相应的流程图,转译器的输入分别是左上角的实体-能力抽象信息 W 和右上角的 SS 规则集合 $\mathbb{G}^S$,输出是与输入 SS 规则集对应的 EE 规则集 $\mathbb{R}^E$(右下角).



(a) 转译算法 (b) 转译流程图

Fig.3 Algorithm and flowchart of translation from SS rules to EE rules

图 3 SS 规则到 EE 规则转译算法与流程图

转译的具体流程为:

- SS 规则集解析器首先会解析用户输入的 $\mathbb{G}^S$,并将解析结果传给动作序列信息生成模块;
- 然后,动作序列信息生成模块会将同一动作实体 e 的 SS 规则序列 $G^S=(e,\mathbb{R}^S)$ 按照 EXPECT 子句作用的期望状态组分类,期望状态组 $\mathcal{X}$ 相同的 SS 规则组 $\mathbb{R}_X^S$ 中的多条规则会放在一起处理:一方面,调用 *ConditionalActionsPairs*(e,$\mathcal{X}$,W)(见算法 2),计算实体 e 上所有可能的动作序列 $\mathbb{A}_j$ 和每种动作序列的执行前提 C_j,得到二元组集合 $Pairs_X=\{(C_j,\mathbb{A}_j)\mid j\in\{1,n\}\}$;另一方面,求出与规则组 $\mathbb{R}_X^S$ 不兼容的条件 Ψ_X(见算法 1 第 5 行),这是事件发生后、动作执行前系统抽象状态所应满足的条件;
- 接下来,事件筛选模块首先调用 *CandidateEvents*(Ψ_X,W)(见算法 3),根据 Ψ_X 生成一组候选事件 $\mathbb{E}$(这些事件可能对 Ψ_X 的成立与否有影响);然后,对 $\mathbb{E}$ 中的每个事件 E 调用 *ShouldRejectEvent*(E,Ψ_X,W)(见算法

4),依据缺省的事件筛选策略$\mathcal{P}_{default}$判断该事件是否要排除掉,即:若 E 发生后$\Psi_{\mathcal{X}}$不成立,则需将 E 排除;

- 筛选后的事件集$\dot{E}'$与$Pairs_{\mathcal{X}}$一并送至 EE 中间表示生成模块处理,该模块对$Pairs_{\mathcal{X}}$中的每个元组(C_j, $\mathbb{A}_j$)调用 $SameActionRules(\dot{E}',\Psi_{\mathcal{X}},C_j,\mathbb{A}_j,W)$(见算法 5),产生动作序列为$\mathbb{A}_j$的 EE 规则集;然后,经过规则合并、可读性化简后(见算法 1 第 13 行),这些 EE 规则集再由 EE 规则集生成器汇总输出精简的 EE 规则集,最终实现从 SS 规则得到对应的 EE 规则.

在转译过程中,有多个环节需要与 Z3 交互:在转译开始之前,转译器会根据实体-能力抽象构建 Z3 数据类型和 Z3 常量,并将对应关系填入 Z3-Python 数据结构转换模块中的 Z3 常量映射表;在转译过程中,事件筛选模块和 EE 中间表示生成模块通过 Z3 判定事件的可满足性;规则合并模块通过 Z3 判断等价的条件部分,合并触发事件;可读性化简模块通过 Z3 进行表达式化简以减少条件表达式长度.

下面结合算法 2~算法 5 详细介绍动作序列信息生成、事件筛选以及 EE 中间表示生成这 3 个关键模块.

算法 2. "执行前提-动作序列"对集合的生成(ConditionalActionsPairs).

输入:实体 e,期望状态组$\mathcal{X}$,系统实体-能力抽象 W;

输出:结构如$\{(C_1,\mathbb{A}_1),\ldots,(C_n,\mathbb{A}_n)\}$的 $Pairs$,表示实体 e 满足 C_i 时要达到$\mathcal{X}$需执行$\mathbb{A}_i$.

1. $Pairs=\varnothing$, $\mathbb{C}_{\mathcal{X}}=\mathcal{X}|_{\boldsymbol{c}}$ //$\mathcal{X}|_{\boldsymbol{c}}$表示对$\mathcal{X}$中的$(\boldsymbol{c},v)$二元组集合的 $\boldsymbol{c}$ 投影得到实体能力集合
2. **if** $\exists \boldsymbol{c}\in\mathbb{C}_{\mathcal{X}}$, c 不是可控能力 **then error**
3. $\mathbb{C}'_{\mathcal{X}}=Reorder(\mathbb{C}_{\mathcal{X}},\{M_{\boldsymbol{c}}\mid \boldsymbol{c}\in\mathbb{C}_{\mathcal{X}}\})$
4. **for each** $\boldsymbol{c}_i$ in $\mathbb{C}'_{\mathcal{X}}$, $(\boldsymbol{c}_i,v_i)\in\mathcal{X}$ (i=1,…,m) **do**
5. $Sets_i=SplitRangeByAction(M_{\boldsymbol{c}_i},v_i)$ //将 $\boldsymbol{c}_i$ 的状态值域$V_{\boldsymbol{c}_i}$根据$\mathbb{T}_{\boldsymbol{c}_i}$和目标值 v_i 划分为$\{V_1^i,\ldots,V_{k_i}^i\}$
6. **end for**
7. **for** $Sets_1\times Sets_2\times\ldots\times Sets_m$ 中的每一个组合 $(V_{j_1}^1,\ldots,V_{j_m}^m)$ **do** //考察每一种初始取值的组合
8. $\mathbb{A}=CheckedActionSteps((\boldsymbol{c}_1,V_{j_1}^1,v_1),\ldots,(\boldsymbol{c}_m,V_{j_m}^m,v_m))$ //返回动作序列或特殊值 IMPOSSIBLE
9. $C=AsCond(\boldsymbol{c}_1,V_{j_1}^1)$ and...and $AsCond(\boldsymbol{c}_m,V_{j_m}^m)$ //$\boldsymbol{c}_1$~$\boldsymbol{c}_m$的取值范围分别处于$V_{j_1}^1,\ldots,V_{j_m}^m$的条件
10. $Pairs=Pairs\cup\{(C,\mathbb{A})\}$
11. **end for**

- 动作序列信息生成模块

该模块会针对期望状态组$\mathcal{X}$相同的 SS 规则组$\mathbb{R}_{\mathcal{X}}^S$,调用算法 2 产生可能的动作序列及其执行前提集合$Pairs_{\mathcal{X}}=\{(C_j,\mathbb{A}_j)\mid j\in\{1,n\}\}$,并求出与$\mathbb{R}_{\mathcal{X}}^S$不兼容的条件$\Psi_{\mathcal{X}}$(见算法 1 第 5 行).算法 2 的第 2 行指出,$\mathcal{X}$投影得到的能力集合$\mathbb{C}_{\mathcal{X}}$中的每个能力都必须是可控的.如第 2.4 节所述,可控能力 $\boldsymbol{c}$ 的状态机 $M_{\boldsymbol{c}}=(V_{\boldsymbol{c}},\mathbb{T}_{\boldsymbol{c}})$中的转移函数$\mathbb{T}_{\boldsymbol{c}}:V_{\boldsymbol{c}}\xrightarrow{[C]\mathcal{J}}V_{\boldsymbol{c}}$,每个状态转移带有条件 C 以及要执行的命令$\mathcal{J}$,C 中可能会涉及其他实体能力.考虑到实际需求以及处理上的简便,假设$\mathbb{C}_{\mathcal{X}}$中各能力的状态转移条件 C 中依赖的其他能力遵循部分序关系,算法 2 的第 3 行调用 Reorder 函数将$\mathbb{C}_{\mathcal{X}}$中的能力按这种序关系重排得到$\mathbb{C}'_{\mathcal{X}}$,这样在第 8 行就可以尝试按此顺序依次检查动作转移的可行性.另外,对于$\mathbb{C}'_{\mathcal{X}}$中的能力 $\boldsymbol{c}_i$,其状态值域$V_{\boldsymbol{c}_i}$可能是无限的实数,而$M_{\boldsymbol{c}_i}$是有限的,故第 5 行的 SplitRangeByAction 函数会检查$M_{\boldsymbol{c}_i}$中到达期望值 v_i 的各个状态转移动作$\mathcal{J}$和转移条件 C,将$\mathcal{J}$,C 相同的状态转移的起始状态涉及的各状态值划分到同一个子集,从而形成$V_{\boldsymbol{c}_i}$的有限子集划分 $Sets_i$,进而可能减少第 7 行要处理的$\mathbb{C}'_{\mathcal{X}}$初始状态情形的数目,也能够减少后续规则合并的运算次数.

下面结合图 4 的例子来解释.该例子的动作实体为"电扇",有 3 条 SS 规则.这些规则按期望状态组$\mathcal{X}$是否相同分成两组$\mathbb{R}_{\mathcal{X}_1}^S$(①③)和$\mathbb{R}_{\mathcal{X}_2}^S$(②).以$\mathbb{R}_{\mathcal{X}_2}^S$组为例,根据该组的$\mathcal{X}_2$"(开关,开)(风速,低)"和图 2 电扇的状态机,按照

算法2求出到达$\mathcal{X}_2$的所有执行前提-动作序列对(图4左下方的4种动作序列).另一方面,该模块也要输出$\mathbb{R}^S_{\mathcal{X}_2}$不兼容的条件$\Psi_{\mathcal{X}_2}$用于稍后的事件筛选,$\Psi_{\mathcal{X}_2}$=模式.状态==回家模式 AND 温湿度传感器.温度 >28°C AND (电扇.开关!=开 OR 电扇.风速!=低).

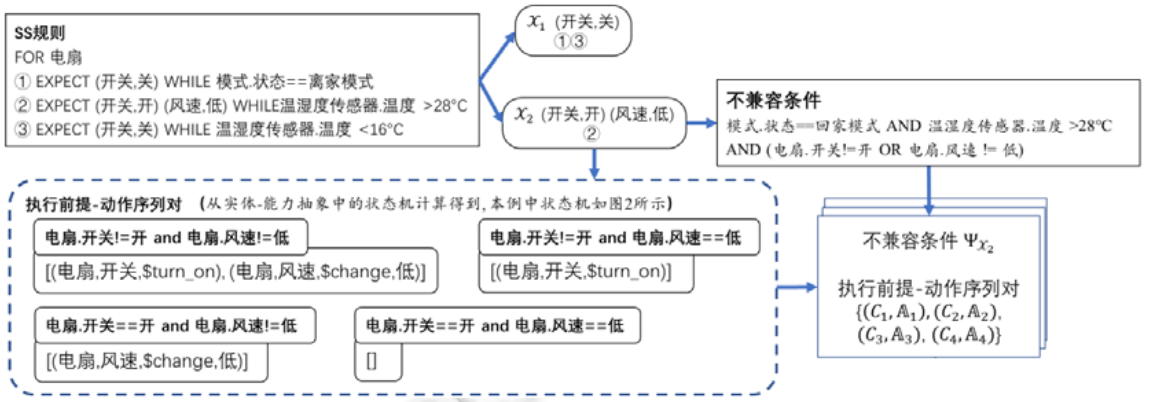


Fig.4 Generating sequences of commands information

图4 动作序列信息生成

- 事件筛选模块

该模块先使用算法3生成候选事件,接着进一步使用算法4,根据系统可能的状态变迁对事件进行筛选.算法3主要根据一个执行前要满足的条件C生成候选事件集,旨在以较低运算成本快速得到较小的待筛选事件集.其中:第2行分别获取C中出现的实体能力集合ECs以及原子条件集合C_{atom},原子条件有当前状态判断$(\boldsymbol{e},\boldsymbol{c},O,v)$和历史状态判断$(\boldsymbol{e},\boldsymbol{c},O_h,t,v)$这两种形式(如表7);第4行的SplitRangeByCond则是根据C_{atom}对实体$\mathbb{E}$的能力$\boldsymbol{c}$的状态值域$V_{\boldsymbol{c}}$进行划分,将对所有原子条件的判断影响一致的状态取值划分到同一个子集,从而将实数类型的事件转移转化为了有限情形;第5行~第7行分别产生状态变化事件E^l和状态保持事件E^H,能够涵盖C中出现的原子条件取值变化的情形.对于历史状态原子条件$(\boldsymbol{e},\boldsymbol{c},O_h,t,v)$,由于只支持二值类型的能力$\boldsymbol{c}$,在固定$\boldsymbol{e},\boldsymbol{c},t$下,$O_h$和$v$最多有8种组合,我们为之按$v$的两种取值产生2种状态保持事件,忽略$O_h$;而被忽略的$O_h$仍在$C$中,传递到后续由算法5产生的EE规则$R^E=(\mathbb{E},C,\mathbb{A})$中.这种状态保持事件$E^H$能在状态迁移中快速知道实体能力保持状态的时长,再结合C中的O_h进行状态保持的可满足性检查.

算法3. 候选事件生成算法(CandidateEvents).

输入:条件表达式C,系统实体-能力抽象W;

输出:包含所有可能使C的值变化的事件集合$\mathbb{E}$.

1. $\mathbb{E}=\varnothing$
2. $ECs=C|_{(\boldsymbol{e},\boldsymbol{c})}$; $C_{atom}=AtomFormulas(C)$
3. **for each** $(\boldsymbol{e},\boldsymbol{c})$ in ECs,由W得$\boldsymbol{c}$的值域$V_{\boldsymbol{c}}$ **do**
4. $Sets=SplitRangByCond(\boldsymbol{e},\boldsymbol{c},V_{\boldsymbol{c}},C_{atom})$ //划分值域$V_{\boldsymbol{c}}$
5. $\mathbb{E}=\mathbb{E}\cup\{(\boldsymbol{e},\boldsymbol{c},V_1,V_2)|\forall V_1,V_2\in Sets,V_1\neq V_2\}$
6. $T=\{t|(\boldsymbol{e},\boldsymbol{c},O_h,t,v)\in C_{atom}\}$ //C中原子条件涉及对$\boldsymbol{c}$的时长t的要求
7. $\mathbb{E}=\mathbb{E}\cup\{(\boldsymbol{e},\boldsymbol{c},v,t)|\forall v\in V_{\boldsymbol{c}},\forall t\in T\}$
8. **end for**

算法4. 事件排除判定算法(ShouldRejectEvent).

输入:事件E,事件发生后应为真的条件Ψ,

实体-能力抽象 W,事件筛选策略$\mathcal{P}=\mathcal{P}_{default}$;

输出:该事件是否应被排除.

1. $C_E=E$ 形如$(\boldsymbol{e},\boldsymbol{c},V_s,V_d)$?$EventCond(M_{\boldsymbol{c}},V_s,V_d)$:True
   //从 $M_{\boldsymbol{c}}$ 中获取 $V_s\to V_d$ 的事件转移必要条件
2. $Assertion = SysStateAssert_{\mathcal{P}}(E,!\Psi,\Psi,C_E)$
   //$\mathcal{P}_{default}$ 为 E 发生前$!\Psi$为真、E 发生后Ψ,为真
   //事件发生前后 C_E 成立
3. $SATResult$=Z3.$Satisfiable$($Assertion$,W)
4. **if** $SATResult$="UNSAT" **then return true**
5. **else return false**

算法 5. 同一动作序列的 EE 规则生成(SameActionsRules).

输入:筛选后的事件集$\mathbb{E}'$,不兼容条件 $\Psi_{\mathcal{X}}$,动作序列$\mathbb{A}_j$,执行前提 C_j,系统实体-能力抽象 W;

输出:与动作序列$\mathbb{A}_j$ 相同的规则集合 $\mathbb{R}_{\mathbb{A}}^{E}$.

1. **if** $\mathbb{A}_j$ 为空序列 **then return**
2. $\mathbb{R}_{\mathbb{A}}^{E}=\varnothing$
3. **for each** E in $\dot{\mathbb{E}}'$ **do**
4. $C_E=E$ 形如$(\boldsymbol{e},\boldsymbol{c},V_s,V_d)$?$EventCond(M_{\boldsymbol{c}},V_s,V_d)$:True
5. $C=\Psi_{\mathcal{X}}$ and $C_j|_{Postcond(E)}$ //在事件 E 发生后,$\Psi_{\mathcal{X}}$ and C_j 成立的条件
6. $Assertion = SysStateAssert_{\mathcal{P}}(E,!\Psi,C,C_E)$ //默认策略仍假设事件发生前$!\Psi$为真
7. **if** Z3.$Satisfiable$($Assertion$,W)="UNSAT" **then continue**
8. **if** $\mathbb{A}_j$="IMPOSSIBLE" **then error** "无法保证翻译规则的一致性"
9. $\mathbb{R}_{\mathbb{A}}^{E}=\mathbb{R}_{\mathbb{A}}^{E}\cup(\{E\},C,\mathbb{A}_j)$
10. **end for**

算法 4 旨在判断事件 E 可否发生并在发生后使得 Ψ($\mathbb{R}_{\mathcal{X}}^{S}$ 不兼容的条件)为真(进而导致 Ψ对应的某个动作序列的执行),需要根据筛选策略$\mathcal{P}$做关于系统抽象状态的可满足性判定.筛选策略从保守到激进有多种策略:激进的策略需要对事件发生之前的系统状态做出更强的假设,而保守的策略只需要相对较弱、能够更快判定的假设.本文提出的默认策略$\mathcal{P}_{default}$ 为:事件发生前 Ψ为假($\mathbb{R}_{\mathcal{X}}^{S}$ 兼容),事件发生后 Ψ为真($\mathbb{R}_{\mathcal{X}}^{S}$ 不兼容),且事件发生前后 $V_s\to V_d$ 的事件转移条件 C_E 成立,则该事件不会被排除.

继续以图 4 的规则组 $\mathbb{R}_{\mathcal{X}_2}^{S}$ 为例,其不兼容条件为 $\Psi_{\mathcal{X}_2}$.对于事件 E=(模式,状态,{离家模式},{回家模式}),由于 $M_{\text{模式状态}}$的离家模式到回家模式的转移没有必要条件,故 C_E=TRUE.按照默认筛选策略,将模式.状态在事件发生前后的值分别代入$!\Psi_{\mathcal{X}_2}$ 和 $\Psi_{\mathcal{X}_2}$,则 $Assertion$ 为"!(离家模式=回家模式 AND 温湿度传感器.温度>28°C AND (电扇.开关!=开 OR 电扇.风速!=低)) AND (回家模式=回家模式 AND 温湿度传感器.温度>28°C AND (电扇.开关!=开 OR 电扇.风速!=低))".这一断言是可满足的,所以事件 E 不被排除,将交由 EE 中间表示生成处理.

以图 4 中 $\mathbb{R}_{\mathcal{X}_1}^{S}$ 规则组为例,本文的默认事件筛选策略与系统抽象状态的可能转移之间的联系如图 5,左上方是以 G^S 中的若干规则组的状态为局部视角时系统抽象状态的转移图,按照一次一个外部事件、一次执行一条规则的系统执行假设,不会发生的系统抽象状态转移用点线表示.按照当前默认的事件筛选策略$\mathcal{P}_{default}$,认为可能到达 $\mathbb{R}_{\mathcal{X}_1}^{S}$ 的不兼容状态的状态转移由右上图中 4 条深色的系统抽象状态转移边表示(与左上图对比,边③是在当前系统抽象和规则执行假设下不会发生的状态转移),但是在左下方粗粒度的系统抽象状态转移图上,却对应着相同的转移边.使用较保守的事件筛选策略的缺点就在于可能未排除边③这样的边,进而存在生成出不能执行到的规则的可能性.需要注意的是:EE 规则中的事件和图中的系统抽象状态转移边不一定一一对应,能够

触发同一条 EE 规则的所有可能的状态转移在细粒度的状态转移图上可能对应多条边.例如 EE 规则“IF 模式.状态 FROM {回家模式} TO {离家模式} WHILE 电扇.开关==开 THEN 电扇.开关 关闭”,该规则被触发的情况可能是图 5 右上图中的状态转移①,也可能是状态转移②.从这一角度来说,使用较保守的事件筛选策略的优点在于判定条件较少,运算更快(判定的是宽泛的系统抽象状态集合间的转移).

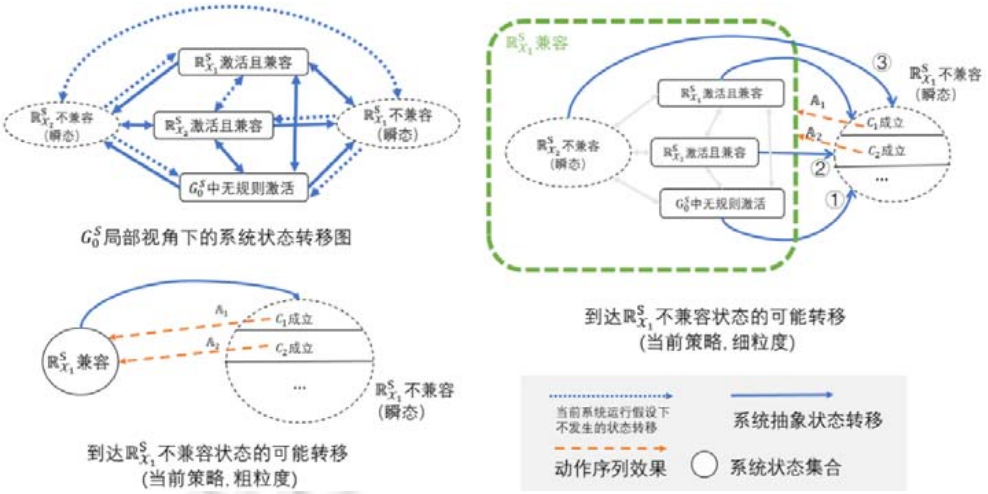


Fig.5 Relation between current event filtering strategy and state transition

图 5 当前使用的事件筛选策略与状态转移的关系

- EE 中间表示生成模块

该模块调用算法 5 生成 EE 规则.以上一步骤筛选输出的事件 E=(模式,状态,{离家模式},{回家模式})、规则组 $\mathbb{R}^S_{\mathcal{X}_2}$ 的不兼容条件 $\Psi_{\mathcal{X}_2}$ 为例,对于图 4 左下方中的 4 个执行前提-动作序列对,都要判断执行前提是否可能满足并生成规则.这里以 4 对中的左上一对为例,由算法 5 的第 5 行得到的条件:“回家模式=回家模式 AND 温湿度传感器.温度>28°C AND (电扇.开关!=开 OR 电扇.风速!=低) AND (电扇.开关!=开 AND 电扇.风速!=低)”作为动作执行的完整前提条件,同时也应当是 E 发生后为真的条件,结合事件发生前应当为真的条件!Ψ,得到关于系统状态的断言并判定是可满足的,因此生成一条规则:

IF 模式.状态 FROM {离家模式} TO {回家模式} WHILE 温湿度传感器.温度>28°C AND
电扇.开关!=开 AND 电扇.风速!=低 THEN 电扇.开关 打开,电扇.风速 设定 低.

算法 4 虽然和这里的算法 5 都需要对一组系统状态的断言做可满足性求解,但是算法 4 不考虑动作序列的执行前提,其筛选结果能够被同一$\mathcal{X}$的多个动作序列复用;且能够通过算法 4 筛选的只有较少数量事件,这些事件至少能够与一个动作序列组合成为规则.算法 4 的存在,减少了转译过程中总的可满足性判定的次数.若直接将候选事件集合与多种动作序列的组合输入算法 5,可能需要较多次数的不必要的求解.

4 方法评估

为了验证 SSRules 的系统设计的有效性,基于 Python 开发了 SSRules 运行时子系统和离线转译器;基于 Javascript 开发了 SSRules 前端用户交互模块,方便终端用户设置 SS 规则(如图 6 左上方);同时,结合已有用户调查[15]构建智能家居使用情景,在这些场景下,就 SSRules 系统运行情况和规则转译效果进行评估和讨论.

4.1 智能家居实验场景搭建

为了能够评估 SSRules,需要将一些设备连接到 HA,并让设备和环境产生足够多的状态转移,以覆盖各种可能的使用情形.但是,使用真实设备无法在短时间得到这样的测试场景并且调试不便.为此,我们基于 Unity 游戏引擎开发了一个能接入 HA 的智能家居模拟器 HA-Simulator(其界面如图 6 下方).利用 HA-Simulator 可以将时

间加速,以较高的速度发生系统的状态变化,并对温湿度/烟雾浓度等环境因素合理建模.这使得在短时间内测试大量的智能家居系统状态变迁并验证 SSRules 成为可能.

在 HA-Simulator 中模拟了 12 种智能家居设备,布局和类型如图 6 右上方所示,这些设备通过 HA 的 MQTT Discovery 集成方式接入 HA.此外,在 HA 中以自定义方式提供模式、天气和时钟这 3 种虚拟设备.在模拟器中,同一房间内的设备受同一组环境变量(温度、湿度、烟雾浓度)的影响.模拟器可以设置环境变量改变的速度,进而间接改变虚拟传感器的取值;同时,模拟器中的设备可以被手动调节或者由模糊测试程序随机改变.

Fig.6 SSRules user interface, Smart home configuration and HA-simulator

图 6 SSRules 用户交互界面与智能家居场景配置和模拟器示意图

4.2 实验结果分析

- SS→HA 翻译对比

首先参考以往用户调查[15]中用户的智能家居需求,构造了 10 组 SS 规则作为测试集,其中:编号为 1~6 的规则组不含历史状态,编号为 T1~T4 的规则组含有历史状态.每一组规则至少有一个动作实体(设备),规则中会引用多个相关实体的状态.另外,规则组 6、规则组 T3、规则组 T4 含有多个动作实体.按照当前的事件筛选策略,翻译前后的规则条数比较见表 8.最终生成的 HA 规则条数均多于 SS 规则条数,比例为 2 倍~4 倍.

- HA 规则运行覆盖率

在由 SS 转译得到 HA 规则之后,每一组 HA 规则均在模拟器上运行 20 分钟.除第 1 组规则(包含空调与温湿度传感器)和第 3 组规则(包含排气扇以及其他 4 个相关实体)之外,其余组的 HA 规则均全部被执行过.这说明 SS→HA 转译器为其余 8 组规则产生的 HA 规则集中没有多余的规则,当前的事件筛选策略对于多数测试样例是比较合理的.对于含有多余规则的情形,而第 1 组的 4 条 SS 规则为:

FOR 客厅空调

EXPECT (模式,制冷)(温度,18°C) WHILE 客厅温湿度传感器.温度>28°C

EXPECT (模式,制热)(温度,20°C) WHILE 客厅温湿度传感器.温度<10°C

EXPECT (模式,干燥) WHILE 客厅空调.模式!=制热 AND 客厅空调.模式!=制冷 AND 客厅温湿度传感器.湿度>65%

EXPECT (模式,关) WHILE 客厅温湿度传感器.温度<22°C AND 客厅温湿度传感器.温度>14°C

而在转译出的 13 条 HA 规则中,没有被执行过的(可能是多余的)一条 HA 规则用 EE 范式表达为:

IF 客厅温湿度传感器.温度 FROM (–30°C,10°C)∪(28°C,70°C) TO [10°C,28°C] WHILE 客厅空调.模式!=制冷 AND 客厅空调.模式!=制热 AND 客厅空调.模式!=干燥 AND 客厅温湿度传感器.湿度>65% THEN 客厅空调.模式设定干燥

Table 8 Comparison of SS and HA rules and test results

表 8 SS→HA 翻译对比和测试运行

规则组编号	动作实体	相关实体	SS#	EE#	HA#	覆盖率(HA)	规则执行次数	瞬时异常	连续异常
1	3 客厅空调	3,11	4	13	13	12/13	325	1/1200	无
2	7 客厅氛围灯	7,12~15	4	14	14	14/14	206	1/1200	无
3	9 厨房排气扇	1,2,9,13,15	3	13	13	12/13	237	1/1200	无
4	6 客厅加湿器	6,11,13	3	6	6	6/6	257	0/1200	无
5	4 厨房窗户	1,4,13~15	3	9	9	9/9	187	0/1200	无
6	2,4,9	1,2,4,9,15	5	11	11	11/11	509	3/1200	无
T1	7 客厅氛围灯	7,12,13	2	5	5	5/5	201	1/1200	无
T2	5 大门门锁	5,12,13	2	6	6	6/6	395	0/1200	无
T3	2~4,7	1~4,7,11~13	8	17	17	17/17	390	0/1200	无
T4	4,8	1,4,8,13~15	6	13	13	13/13	298	0/1200	无

原因在于:当前的筛选策略没有假设系统状态在事件发生之前的瞬间与整个规则组 G^S 的描述相符,而仅仅假设系统状态与第 3 条 SS 规则的描述是兼容的(即第 3 条规则未激活或者其期望状态已满足).但是在测试运行中,当温度低于 10°C 或者高于 28°C 时,系统状态持续与 G^S 的描述相符,空调模式一定是制热(第 2 条 SS 规则激活)或者制冷(第 1 条 SS 规则激活),所以该条 EE 规则是多余的,通过使用更激进的事件筛选策略可将其消除.

- HA 规则运行异常检测

表 8 的后 3 列分别是在 20 分钟的随机测试下,各组的 HA 规则的执行次数、异常检测模块发现的异常次数(系统状态不满足动作实体的 SS 规则集合中激活的那一条 SS 规则的期望状态)与总检测次数之比、是否存在连续两次检测到异常的情况.各组总的规则执行次数相差不大.异常检测模块目前以定时轮询方式实现,在 20 分钟内进行了 1 200 次异常检查,未发现连续出现异常的情形.对于单次检测出现异常的情况,经过查看记录的状态日志和 HA 规则的执行序列,这些异常均因网络延迟(动作执行不及时)造成.

5 相关工作

智能家居目前在市场上受到了广泛欢迎,越来越多的家庭配置了智能家居的设备.智能家居最吸引人的功能之一就是以应用程序的形式支持自定义自动化.但是由于用户未经过专业的培训,导致其自定义的规则存在一定的缺陷,甚至无法定制规则,这引起了人们对物联网增强生活的风险的担忧.这些风险远非仅仅是学术上的,更紧迫的是对用户的生命财产的威胁.Triger-Action 编程(TAP)是一种流行的终端用户编程方式,可以让用户实现其智能设备和云服务的交互.然而,用户有时并不能通过 TAP 正确表达自己的意图并实现相应的功能.随着此类系统部署在越来越复杂的智慧家居场景中,用户必须能够识别编程错误并解决.

为了给终端用户提供更高效的服务,首先需要理解用户编程出现错误和规则冲突的原因.Huang 等人[6]认为,现有 TAP 编程系统(如 IFTTT)的过分简化限制了程序的表达能力.提出了改进 IFTTT 界面的建议,以减轻因心理模型不正确而引起的问题.Corno 等人[28]认为:现有的 TAP 接口界面暴露了太多功能,并迫使用户在众多混乱的网格菜单中进行搜索,导致用户无法得到原本需要的功能.为此,作者提出了 EUDoptimizer,旨在以交互方式协助终端用户使用循环中的优化器来组合 IF-THEN 规则,减少了编写 Triger-Action 规则所需的工作.Brackenbury 等人[17]提出了 10 类常见的 TAP 编程错误类型,作者发现:错误的存在,使参与者更难正确预测规则的行为.Davidoff 等人[29]发现,智能家居用户的日常活动与编程任务并不匹配.论文描述了家庭想要的控制,并提出了有助于终端用户编程系统实现这种控制的 7 种设计原则.Brush 等人[20]认为:为使智能家居得到更广泛的使

用,需要解决成本高昂、设备不灵活、客观理性差和安全保障这 4 个障碍.论文还为进一步研究提供了几个方向,包括消除变化结构的需要、为用户提供简单的安全原语并支持家庭设备的组合.

因此,为减少终端用户编程的错误,TAP平台需要对用户生成TAP规则进行指导,并实现模型的检查,自动发现规则漏洞或冲突.Wang 等人[14]利用 IFTTT 平台,探讨了在 Triger-Action 平台中可能存在的规则漏洞,并借助于自然语言处理的方法推断Triger-Action信息,实现了一个模型检查系统iRuler,用于发现物联网中的规则间漏洞.Celik 等人[19,30,31]认为,物联网中存在大量的设备交互.因此,不仅需要验证单个设备,还需要对设备的联合行为进行检查.Soteria[30]是一种静态分析系统,可从 IoT 应用程序的源代码中提取状态模型,并通过模型检查来验证 IoT 应用程序或 IoT 环境是否符合已标识的属性.考虑到静态分析在估计物联网状态转换方面的局限性,Celik 等人[19]继续开发了一个动态分析系统 IoTGuard,可通过在运行时监视设备执行行为,来强制执行已标识的属性,并可以通过阻止违反属性的设备操作,或在运行时要求用户批准或拒绝违规操作来应对属性违规.肖丁等人提出的基于知识图谱的 KGIID 算法[32]能够检测 TAP 编程模型中冲突的动作(隐式冲突).孟岩等人提出的HODETECT 检测工具[33]利用无线侧信道技术从智能家居应用中提取工作逻辑,并用有限状态自动机建模,通过在应用运行时监听无线加密流量的元数据来推测应用的执行状态,间接检测恶意应用.陈星等人[34]提出了智能家居情境感知服务的运行时建模与执行方法,能够降低开发者开发智能家居情境感知服务的难度和复杂度.

更进一步地,TAP 服务提供平台还需要对存在缺陷的规则进行修复.Nandi 等人[26]注意到,终端用户在编写规则时所犯的一个常见错误是触发器数量不足.因此,作者开发了 TrigGen,它可以根据规则中的操作自动生成一组必要和充分的触发器,来帮助终端用户正确地编写规则.这种做法在一定程度上减少了意外行为和安全漏洞的发生率,但是其对于潜在冲突的检测还有提升空间.为帮助非专业的物联网用户系统地实现高置信度的实时 HA-IoT 系统,Bu 等人[16]介绍了一种自动化端到端编程辅助框架 MenShen,它能够检查自动化规则集是否违反规范,从而为用户提供可能的解决方案.Zave 等人[24]建议在运行时通过优先级来减少设备管理和交互的成本,并通过组合机制实现了目标.Zhang等人[15]将用户需要表达的属性转换为线性时序逻辑(LTL),自动合成满足属性的 TAP 规则,并修复现有的 TAP 规则.

6 总 结

基于规则的智能家居系统是目前流行的一种智能家居系统,而编写琐碎、修改困难、错误难以追踪是以往用户调查[15]所反映的问题.本文提出的 SS 范式以及配套的 SSRules 系统实现方式,使得用户规则编写和修改的复杂程度大为减低;同时,设计的 SS 规则到 HA 规则的转译算法使得输入的 SS 规则能够在现有的智能家居系统 HA 规则引擎上运行,并且通过辅助的 SSRules 运行时模块,能够提供规则执行异常检测的能力.

未来的工作重点在以下两个方面:一方面,采用模型检测技术实现对 SS 范式无法避免的缺陷进行检测,并设计辅助用户编写 SS 规则和避免缺陷的实时反馈算法;另一方面,对于 SS 范式目前不能够表达的需求,进行用户调查并寻找合适的多范式协同输入对策.

References:

[1] MARKETWATCH. Global Smart Home Market Size, Growth, Opportunity and Forecast to 2025. 2019.

[2] iResearch. 2018 China Smart Home Industry Research Report. 2018 (in Chinese). http://report.iresearch.cn/report/201808/3256.shtml

[3] IFTTT. If this then that. 2018. https://ifttt.com/

[4] Home assistant: Awaken your home. 2019. https://www.home-assistant.io/

[5] Ur B, McManus E, Ho MPY, *et al*. Practical trigger-action programming in the smart home. In: Proc. of the Conf. on Human Factors in Computing Systems. 2014. 803−812. [doi: 10.1145/2556288.2557420]

[6] Huang J, Cakmak M. Supporting mental model accuracy in trigger-action programming. In: Proc. of the 2015 ACM Int'l Joint Conf. on Pervasive and Ubiquitous Computing (UbiComp 2015). 2015. 215−225. [doi: 10.1145/2750858.2805830]

[7] Brich J, Walch M, Rietzler M, *et al*. Exploring end user programming needs in home automation. ACM Trans. on Computer-Human Interaction, 2017,24(2):11:1−11:35. [doi: 10.1145/3057858]

[8] Ghiani G, Manca M, Paternò F, *et al*. Personalization of context-dependent applications through trigger-action rules. ACM Trans. on Computer-Human Interaction, 2017,24(2):14:1−14:33. [doi: 10.1145/3057861]

[9] Oswald E. IFTTT competitor stringify gets a major update. 2016. https://www.techhive.com/article/3086988/ifttt-competitor-stringify-gets-a-major-update.html

[10] Gruber J. Codeless automation: IFTTT vs zapier vs microsoft flow. 2018. https://medium.com/better-programming/codeless-automation-ifttt-vs-zapier-vs-microsoft-flow-57d5bc56fc0e

[11] Rahmati A, Fernandes E, Jung J, *et al*. IFTTT vs. zapier: A comparative study of trigger-action programming frameworks, 2017. https://arxiv.org/abs/1709.02788

[12] Brice. Announcing SmartRules 2.0. 2016. http://smartrulesapp.com/2016/04/26/announcing-smartrules-2-0/

[13] Nacci A, Rana V, Balaji B, *et al*. BuildingRules: A trigger-action–based system to manage complex commercial buildings. ACM Trans. on Cyber-Physical Systems, 2018,2(2). [doi: 10.1145/3185500]

[14] Wang Q, Datta P, Yang W, *et al*. Charting the attack surface of trigger-action IoT platforms. In: Proc. of the ACM Conf. on Computer and Communications Security. New York: Association for Computing Machinery, 2019. 1439−1453. [doi: 10.1145/3319535.3345662]

[15] Zhang L, He W, Martinez J, *et al*. AutoTap: Synthesizing and repairing trigger-action programs using LTL properties. In: Proc. of the Int'l Conf. on Software Engineering. 2019. 281−291. [doi: 10.1109/ICSE.2019.00043]

[16] Bu L, Xiong W, Liang CJM, *et al*. Systematically ensuring the confidence of real-time home automation IoT systems. ACM Trans. on Cyber-Physical Systems, 2018,2(3):1−23. [doi: 10.1145/3185501]

[17] Brackenbury W, Deora A, Ritchey J, *et al*. How users interpret bugs in trigger-action programming. In: Proc. of the Conf. on Human Factors in Computing Systems. 2019. [doi: 10.1145/3290605.3300782]

[18] De Moura L, Bjørner N. Z3: An efficient SMT solver. In: Ramakrishnan CR, Rehof J, eds. Proc. of the Tools and Algorithms for the Construction and Analysis of Systems. Berlin, Heidelberg: Springer, 2008. 337−340.

[19] Celik ZB, Tan G, McDaniel P. IoTGuard: Dynamic enforcement of security and safety policy in commodity IoT. In: Proc. of the 26th Annual Network and Distributed System Security Symp. 2019. [doi: 10.14722/ndss.2019.23326]

[20] Brush AJB, Lee B, Mahajan R, *et al*. Home automation in the wild: Challenges and opportunities. In: Proc. of the Conf. on Human Factors in Computing Systems. New York: Association for Computing Machinery, 2011. 2115−2124. [doi: 10.1145/1978942.1979249]

[21] Yarosh S, Zave P. Locked or not? Mental models of IoT feature interaction. In: Proc. of the Conf. on Human Factors in Computing Systems. 2017. 2993−2997. [doi: 10.1145/3025453.3025617]

[22] Liang CJM, Bu L, Li Z, *et al*. Systematically debugging IoT control system correctness for building automation. In: Proc. of the 3rd ACM Conf. on Systems for Energy-Efficient Built Environments (BuildSys 2016). 2016. 133−142. [doi: 10.1145/2993422.2993426]

[23] Dey AK, Sohn T, Streng S, *et al*. iCAP: Interactive prototyping of context-aware applications. LNCS 3968, 2006. 254−271. [doi: 10.1007/11748625_16]

[24] Zave P, Cheung E, Yarosh S. Toward user-centric feature composition for the Internet of things. arXiv preprint arXiv:1510.06714, 2015.

[25] Cao J, Rector K, Park TH, *et al*. A debugging perspective on end-user mashup programming. In: Proc. of the 2010 IEEE Symp. on Visual Languages and Human-Centric Computing (VL/HCC 2010). 2010. 149−156. [doi: 10.1109/VLHCC.2010.29]

[26] Nandi C, Ernst MD. Automatic trigger generation for rule-based smart homes. In: Proc. of the 2016 ACM Workshop on Programming Languages and Analysis for Security. New York: Association for Computing Machinery, 2016. 97−102. [doi: 10.1145/2993600.2993601]

[27] Wang Q, Hassan WU, Bates A, *et al*. Fear and logging in the Internet of things. In: Proc. of the 22nd Network and Distributed Security Symp. 2018. [doi: 10.14722/ndss.2018.23282]

[28] Corno F, De Russis L, Roffarello AM. EUDoptimizer: Assisting end users in composing if-then rules through optimization. IEEE Access, 2019,7:37950−37960. [doi: 10.1109/ACCESS.2019.2905619]

[29] Davidoff S, Lee MK, Yiu C, *et al*. Principles of smart home control. LNCS (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2006,4206:19−34. [doi: 10.1007/11853565_2]

[30] Celik ZB, McDaniel P, Tan G. Soteria: Automated iot safety and security analysis. In: Proc. of the USENIX Annual Technical Conf. Boston: USENIX Association, 2018. 147−158.

[31] Celik ZB, McDaniel P, Tan G, *et al*. Verifying Internet of things safety and security in physical spaces. IEEE Security Privacy, 2019,17(5):30−37. [doi: 10.1109/MSEC.2019.2911511]

[32] Xiao D, Wang QY, Cai M, *et al*. Research on implicit interference detection based on knowledge graph in smart home automation. Chinese Journal of Computers, 2019,42(6):1190−1204 (in Chinese with English abstract).

[33] Meng Y, Li SF, Zhang YC, *et al*. Cyber physical system security of smart home. Journal of Computer Research and Development, 2019,56(11):2349 (in Chinese with English abstract). [doi: 10.7544/issn1000-1239.2019.20190412]

[34] Chen X, Huang ZM, Ye XS, *et al*. Approach to modeling and executing context-aware services of smart home at runtime. Ruan Jian Xue Bao/Journal of Software, 2019,30(11):3297−3312 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5802.htm [doi: 10.13328/j.cnki.jos.005802]

附中文参考文献:

[2] 艾瑞咨询公司.2018 年中国智能家居行业研究报告.2018. http://report.iresearch.cn/report/201808/3256.shtml

[32] 肖丁,王乾宇,蔡铭,等.智能家居场景联动中基于知识图谱的隐式冲突检测方法研究.计算机学报,2019,42(6):1190−1204.

[33] 孟岩,李少锋,张亦弛,等.面向智能家居平台的信息物理融合系统安全.计算机研究与发展,2019,56(11):2349. [doi: 10.7544/issn1000-1239.2019.20190412]

[34] 陈星,黄志明,叶心舒,等.智能家居情境感知服务的运行时建模与执行方法.软件学报,2019,30(11):3297−3312. http://www.jos.org.cn/1000-9825/5802.htm [doi: 10.13328/j.cnki.jos.005802]

王博(1997－),男,学士,主要研究领域为物联网,智能家居.

耿佳宁(1997－),女,硕士生,主要研究领域为物联网安全.

张昱(1972－),女,博士,副教授,CCF 杰出会员,主要研究领域为面向新领域的编程框架与系统优化,软件分析与软件安全,量子软件.

李向阳(1971－),男,博士,教授,博士生导师,CCF 专业会员,主要研究领域为移动计算,无源网络,智能感知,物联网,安全隐私,数据共享和交易.