软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2018,29(6):1622−1634 [doi: 10.13328/j.cnki.jos.005466]

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

普适计算应用时空性质的运行时验证*

李暄松[1,2], 陶先平[2], 宋 巍[1,2]

[1](南京理工大学 计算机科学与工程学院,江苏 南京 210094)

[2](计算机软件新技术国家重点实验室(南京大学),江苏 南京 210023)

通讯作者: 李暄松, Email: lixs@njust.edu.cn

摘 要: 运行时验证是提升普适计算应用可靠性的重要手段.这类应用的很多性质同时涉及时间关系和空间位置关系,这样的时空性质给运行时的验证带来了特有挑战:一方面,传统的时态逻辑难以描述空间性质;另一方面,适合描述空间性质的 Ambient Logic 在真值不确定等情况下不能很好地支持有限轨迹中时间性质的描述.为支持普适计算应用时空性质的运行时验证,引入三值逻辑语义,提出了 AL_3(3-valued ambient logic);并在此基础上设计实现了基于 AL_3 的性质检验算法和运行时监控器.最后,通过案例分析和运行效率实验阐明了所提方法的有效性和可行性.

关键词: 普适计算;三值语义;运行时验证

中图法分类号: TP301

中文引用格式: 李暄松,陶先平,宋巍.普适计算应用时空性质的运行时验证.软件学报,2018,29(6):1622−1634. http://www.jos.org.cn/1000-9825/5466.htm

英文引用格式: Li XS, Tao XP, Song W. Runtime verification of spatio-temporal properties for pervasive computing applications. Ruan Jian Xue Bao/Journal of Software, 2018,29(6):1622−1634 (in Chinese). http://www.jos.org.cn/1000-9825/5466.htm

Runtime Verification of Spatio-Temporal Properties for Pervasive Computing Applications

LI Xuan-Song[1,2], TAO Xian-Ping[2], SONG Wei[1,2]

[1](School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing 210094, China)

[2](State Key Laboratory for Novel Software Technology (Nanjing University), Nanjing 210023, China)

Abstract: Runtime verification is an important method for improving software reliability of pervasive computing applications. Some properties of these applications consider both temporal and spatial relationships. Such spatio-temporal properties bring some specific challenges for runtime verification. On one hand, traditional temporal logic cannot describe spatial properties. On the other hand, while ambient logic is suitable for spatial properties, it does not properly support the description of temporal properties in finite traces, especially when the truth values cannot be decided. In order to support the runtime verification of spatio-temporal properties for pervasive computing applications, this paper firstly imports 3-valued semantics and proposes AL_3 (3-valued ambient logic). On the basis of AL_3, it designs and implements an algorithm for properties checking and a runtime monitor. Moreover, the paper uses a case study and a performance measurement to clarify the usability and feasibility of the proposed approach.

Key words: pervasive computing; 3-value semantics; runtime verification

* 基金项目: 国家重点研发计划(2017YFB1001801); 国家自然科学基金(61702263, 61373011); 江苏省自然科学基金(BK20171427); 中央高校基本科研业务费专项资金(30917011322)

Foundation item: National Key R&D Program of China (2017YFB1001801); National Natural Science Foundation of China (61702263, 61373011); Natural Science Foundation of Jiangsu Province (BK20171427); Fundamental Research Funds for the Central Universities (30917011322)

本文由形式化方法的理论基础专题特约编辑傅育熙教授、李国强副教授、田聪教授推荐.

收稿时间: 2017-07-01; 修改时间: 2017-09-01; 采用时间: 2017-11-06; jos 在线出版时间: 2017-12-28

CNKI 网络优先出版: 2017-12-29 13:19:23, http://kns.cnki.net/kcms/detail/11.2560.TP.20171229.1318.008.html

普适计算[1]这一新型计算模式强调借助传感器、嵌入式设备、移动设备、网络通信、智能控制等技术的最新成果,将计算融入日常生活的环境,对用户而言,计算是无处不在而又透明的.在普适计算环境中,计算系统可随时感知用户的状态(如动作、身体指标等)和周围的环境,并基于这样的感知信息为用户提供相应的服务,满足用户需求(即,具有上下文感知[2,3]的能力).近年来,普适计算应用的开发、运行支撑研究得到发展[3−6],也有一批示范系统得到实现.在包括老人看护[7]、智能驾驶[8]在内的很多场景下,普适计算系统是安全攸关的.采用形式化方法对于这类系统需要满足的性质进行规约并基于形式规约进行验证,是提升安全攸关系统可靠性、保障用户需求能满足的重要手段.

运行时(动态)验证[9]是形式化验证领域的研究方向之一,其目的是验证软件的运行状态是否满足给定的性质,一般通过监控器(monitor)监控系统的状态,检验其状态是否与性质相符[10].由于普适计算应用的行为受到环境的影响常常出现变化(如一个 RFID 读取器可能因为天线方向、电池电量、摆放位置等各种原因失效),难以在执行之前进行预测、分析,所以在设计阶段进行静态验证的正确性不能等同于运行时的正确性,运行时验证技术特别适合用于普适计算应用的验证[11].

与传统运行时验证问题一样,普适计算应用的性质常涉及时间关系.与此同时,由于普适计算应用需要监控人员、设备的位置,描述移动性,其性质往往也涉及空间位置关系.如“如果一位老人进入浴室洗澡,则应当监测到他离开浴室的动作(以防洗澡时发生晕倒等危险)”这一性质,既描述了两个动作的时间顺序关系,又描述了老人与浴室的空间位置关系,我们将这类性质称为普适计算的时空性质.

为对于普适计算应用的性质进行描述,研究者们从较为基础的一阶逻辑[12]开始,使用各种不同的形式化方法展开了广泛研究.从时态关系角度考虑,使用 LTL(线性时态逻辑)[13]、CTL(树状分支时态逻辑)[14]等时态逻辑(temporal logic,在部分中文文献中被译为“时序逻辑”,为与时序逻辑电路等术语区分,本文译为时态逻辑)描述性质是这类研究的常用手段.从空间关系角度考虑,采用 Ambient Calculus[15]为形式工具描述应用状态、Ambient Logic[16]为逻辑基础定义性质亦得到了广泛的尝试[17−19].

然而,使用上述逻辑系统描述普适计算应用的时空性质存在以下挑战.

- 时态逻辑难以描述空间性质:LTL,CTL 等时态逻辑适合描述时间相关的性质,但是普适计算应用的性质中广泛存在人员与设备位于物理中、人员持有设备乃至软件实体(如 agent)位于设备中等物理上或逻辑上的空间包含关系,时态逻辑并不适合描述这类性质;
- 空间逻辑难以描述有限轨迹中的时间性质:Ambient Logic 可以较为方便地描述空间相关的性质,尽管它也定义了一些时间算子,但与部分传统时态逻辑类似,只适合描述状态转换系统(一般由 Ambient Calculus 描述)无限轨迹中的性质.而运行时验证通常是取一段有限的运行轨迹,获得其状态变化过程(有时被称为一个有限前缀)进行性质验证,我们需要特定的方法对此进行处理;
- 现有方法不能很好地处理真值暂时无法判定的情况:在有限的运行状态中,可能存在一些性质暂时无法判定是否被满足的情况[20],如前文提到的“如果一位老人进入浴室洗澡,则应当监测到他离开浴室的动作”,在通过有限运行状态检验这一性质的过程中,如果老人尚未离开浴室,则无法判断该性质是否被满足.需要有更方便、直观的手段应对这类真值不确定的情况.

为应对上述挑战,本文针对普适计算应用时空性质的运行时验证问题进行以下工作.

- 在 Ambient Logic 的基础上引入三值语义(3-valued semantics),提出了 AL_3(3-valued ambient logic)以描述有限轨迹下的真值不确定情况;
- 设计了基于 AL_3 的性质检验算法以及相应的运行时监控器,以支持普适计算应用的运行时验证;
- 通过案例分析和性能实验,证实上述方法的有效性和可行性.

本文第 1 节介绍相关背景知识.第 2 节描述我们提出的 AL_3 三值语义时空性质规约方法.第 3 节给出运行时验证方法.第 4 节展示案例分析以及性能实验.第 5 节讨论相关工作.第 6 节总结全文并展望未来工作.

1 背 景

首先介绍时态逻辑、Ambient Calculus/Ambient Logic 等形式工具的理论基础;之后,介绍普适计算应用运行时状态以及性质的规约方法.这是我们后续研究 AL_3 及其验证方法的基础.

1.1 理论基础

1) 时态逻辑

时态逻辑是一类依据时间进行命题描述、推理的逻辑系统,典型的时态逻辑包括 LTL[13],CTL[14]等.以下算子是时态逻辑中的部分常用算子.

- $\Phi_1\ U\ \Phi_2$(直到,until):表示在状态满足 Φ_2 之前,一直满足 Φ_1;
- $X\Phi$(下一个,next):表示下一个状态满足 Φ;
- $\Diamond\Phi$(终究,eventually):表示至少有一个状态满足 Φ;
- $\Box\Phi$(总是,always):表示所有状态都满足 Φ.

研究者们对于时态逻辑的算子、语义解释进行了较多研究,包括普适计算在内的计算系统时间性质可得到较便捷的描述,也有一些基于时态逻辑的模型检验器被开发完成,该领域研究极大推动了软件验证技术的发展.

2) Ambient Calculus/Ambient Logic

Ambient Calculus[15]是一种进程演算,它适合用于描述参与计算的物理实体(如人员、设备、房间)以及计算实体(如 agent)的空间包含关系,也可以描述这些实体在空间中的移动.在这个工具中,上述实体皆被看做进程(process),其中最重要的概念是一类特殊的进程 ambient(中文中有时被译为环境,为和 environment 等近义词区分,更清晰地阐述这一概念,本文直接使用英文),它指计算发生的一个有界限的位置(a bounded place where computation happens),ambient 的性质包括:存在边界,可被嵌套在其他 ambient 中,可以作为一个整体进行移动.

Ambient Calculus 中常用的进程描述包括以下符号.

- P,Q::= processes;
- $n[P]$ ambient;
- $P|Q$ composition.

其中,$n[P]$为一个 ambient,n 是这个 ambient 的名,P 是运行于其中的进程(当然,也可以是另一个嵌套其中的 ambient);二元操作 $P|Q$ 表示进程的并行,这个符号符合交换律和结合律.

例如,表达式:$n[P_1|...|P_j|m_1[...]|...|m_k[...]]$描述了一个名为 n 的 ambient 中有 j 个名为 $P_1,...,P_j$ 的进程以及 k 个名称为 $m_1,...,m_k$ 的 ambient.

Ambient Calculus 本身还提供了一些其他算子.如,用“!”描述进程的复制,由于使用该算子会导致相应的检验问题不可判定[21],本文中将不使用该算子;该形式工具还针对进程的移动提供了进入 ambient(in)、离开 ambient(out)、打开 ambient(open)的能力,对于普适计算应用而言,这些算子可用于描述给定软件设计规约情况下的计算实体移动,适合用于静态验证(这类工作见文献[17,18,22]).本文以运行时验证为目标,将直接对于运行时各个时刻的状态进行描述,所以亦不引入这些算子.

通过这个形式工具,我们可以用一个树状结构(本文称为 ambient 树)对于普适计算应用中各计算实体的空间关系进行规约.进一步地,我们可以通过以下介绍的性质规约方法,对于进程的性质进行定义,从而更好地对于应用状态进行描述.

Ambient Logic[16]是一种针对 Ambient Calculus 设计的模态逻辑,Ambient Logic 的公式可用以定义 ambient 相关的性质.满足关系 $P\vDash\Phi$指进程 P 满足封闭的公式 Φ(Φ没有自由变元),可用以描述、检验相应进程的性质.例如,我们可以用如下表达式描述老人 Bob 正在执行阅读这一动作.

$Bob[reading1]$

where $Bob[reading1]\vDash isElder, reading1\vDash hasActType(reading)$

Ambient Logic 的公式的语法定义如下:

定义 1(ambient logic 公式的语法). 设集合 AP 为原子公式集合,则 Ambient Logic 公式的语法定义如下:

$$\Phi ::= T|a|\neg\Phi|\Phi_1\vee\Phi_2|\forall x.\Phi|\Phi@n|n[\Phi]|\Phi_1|\Phi_2|\nabla\Phi|\Diamond\Phi|\Phi_1\triangleright\Phi_2.$$

其中,a 是原子公式,即 $a\in AP$.

否定(¬)、析取(∨)算子与命题逻辑中一致,并可同样地通过这两个算子定义合取(∧)符号 $\Phi_1\wedge\Phi_2\equiv\neg(\neg\Phi_1\vee\neg\Phi_2)$和蕴含(→)符号 $\Phi_1\rightarrow\Phi_2\equiv\neg\Phi_1\vee\Phi_2$.

全称量词(∀)和谓词逻辑一致,同样地,可以用¬∀¬定义存在量词(∃).

该逻辑工具提供了描述 ambient 相关性质的手段,包括:

$P\vDash\Phi@n$:指的是 $n[P]\vDash\Phi$;$P\vDash n[\Phi]$:指的是存在一个进程 P',使得 $P\equiv n[P']\wedge P'\vDash\Phi$;$P\vDash\Phi_1|\Phi_2$:指的是存在 P',P'',使得 $P\equiv P'|P''\wedge P'\vDash\Phi_1\wedge P''\vDash\Phi_2$.

该逻辑工具定义了位置算子和时态算子,其中,位置算子是某处(somewhere∇)$P\vDash\nabla\Phi$指的是 P 进程中的某个位置(可能被嵌套在 ambient 中)存在进程 P',使得 $P'\vDash\Phi$,即:

$$\exists P',P\downarrow^{*}P'\wedge P'\vDash\Phi,$$

其中,↓符号描述进程间的嵌套关系,$P\downarrow Q$ 表示 Q 进程在 P 进程中的某个 ambient 中,$\downarrow^{*}$是↓的传递,即,一个进程在另一个进程嵌套若干层的 ambient 内.

本文着重考察 Ambient Logic 中的时态算子,对于终究(eventually)算子◇,$P\vDash\Diamond\Phi$描述了存在 P',使得 $P'\vDash\Phi$ 且 P'是 P 进程通过进入 ambient、离开 ambient 等操作演化得到的后继进程.相应地,可以定义总是(always)算子□,等价于¬◇¬.从语义上看,这两个算子是用以处理 Ambient Calculus 描述的状态转换系统(使用 in,out,open 等算子)无限轨迹下的时间性质,并不能很好地处理运行时验证中的有限轨迹问题,亦不能应对运行时某一时刻某些性质暂时无法确定是否被满足的场景.

该逻辑工具还提供了保证(guarantee)算子▷,$P\vDash\Phi_1\triangleright\Phi_2$ 表示存在一个进程 P',如果 $P'\vDash\Phi_1$,则 $P'|P\vDash\Phi_2$.由于该算子会导致相应的检验问题不可判定[21],本文中将不引入这个算子.

1.2 普适计算应用的运行时状态与性质规约

1) 普适计算应用运行时状态规约

Ambient Calculus 工具提供了空间嵌套关系的描述手段,对于描述普适计算应用运行时状态(广泛存在位置、人员、计算实体等嵌套关系)是适合的.以下给出一个用该形式工具描述普适计算应用运行状态的例子:

*room*1[*bedroom*1[*nis*1]|*livingroom*2[*lis*1|*la*1|*Bob*[*reading*1]]|*bathroom*3[]]

where *Bob*[*reading*1]⊨*isElder*,*reading*1⊨*hasActType*(*reading*)

该表达式描述了 *room*1 房间包含了 3 个位置:卧室(*bedroom*1)、客厅(*livingroom*2)和浴室(*bathroom*3).其中,卧室部署有噪音传感器(*nis*1),客厅部署有光强传感器(*lis*1)、一盏可控制的灯(*la*1),用户 Bob 是一位老人,正位于客厅中阅读.

2) 普适计算应用性质规约

通过 Ambient Logic 中的位置算子和描述 ambient 相关性质的手段,我们可以较容易、清晰地描述空间位置关系相关的性质,例如:

$$\forall loc\in Location.loc[P]\vDash isKitchen\rightarrow(loc[sensingFea(temperature)|sensingFea(CO)])$$

定义了“厨房中需要部署温度传感器和一氧化碳传感器”这一性质.这个性质描述了如果一个位置(ambient)是厨房,那么该位置中必须有两个传感器(嵌套在该 ambient 中的进程)分别感知温度和一氧化碳浓度;

$$\forall loc\in Location.loc[P]\vDash isLabBuilding\rightarrow\nabla isSecurityStaff$$

定义了“实验楼中必须有安保人员值班”这一性质.在这个性质中,实验楼 ambient(*loc*[*P*])一般嵌套着楼层、房间等其他 ambient,而安保人员可能在楼内任意位置巡逻,只要任意位置有安保人员这条性质即可得到满足,所以使用了∇算子.

然而如前文所述,Ambient Logic 对于时间性质的描述有所不足.

采用时态逻辑描述时间性质亦在普适计算领域得到了广泛的尝试,如老人看护应用中“老人吃饭前必须吃药”这一性质可表示如下:

$$\neg \text{meal}\ U\ \text{medicine}.$$

然而,这类时态逻辑无法很好地描述空间相关的性质,对于普适计算时空性质的描述支持尚有不足.

2 AL_3:普适计算应用时空性质规约

在本文中,我们针对普适计算用的时空性质的运行时验证问题提出了 AL_3.AL_3 采用与 Ambient Logic 一样的语法(如前文所述,不引入▷算子),区别在于引入了三值逻辑描述非确定性情况下的语义.在本节中,我们首先讨论引入三值逻辑的研究用例,之后将给出 AL_3 的定义.

2.1 研究用例

普适计算中的时间性质反映了环境或者计算实体的运行、变化过程,这类性质可以用于触发普适计算中环境自适应的过程,也可以用于保障系统的一些性质始终被满足.对于终究(◇)、总是(□)这两个时态算子而言,通过运行时验证过程获取的部分运行轨迹,往往不能判断性质是得到满足还是被违反[20,23].

将时间性质与 Ambient Logic 描述的空间性质结合,我们可以描述一些普适计算中的时空性质.以老人看护应用为例,性质“如果一位老人进入浴室洗澡,则应当监测到他离开浴室的动作(以免洗澡时发生事故)”可描述如下:

$$\forall loc\in Location.loc[P]\vDash isBathroom\rightarrow(\forall user\in Person.loc[user[hasActType(entering)\vee hasActType(bathing)]]\rightarrow \Diamond loc[user[hasActType(leaving)]]) \quad (1)$$

该性质存在◇算子.对于这个性质而言,如果我们在监测到进入浴室动作后的运行轨迹中监测到了离开浴室的动作,则该性质得到满足;但如果没有监测到离开浴室的动作,则无法判断系统的运行是满足还是违反了该性质.

相似地,我们可以描述性质“老人的动作始终要被监测到”如下:

$$\forall user\in Person.user[P]\vDash isElder\rightarrow\Box(\exists act\in Activity.user[hasActType(act)]) \quad (2)$$

该性质存在□算子.对于这个性质而言,如果某个时刻我们无法监控到老人的动作类型,则该性质被违反;但如果一直能够监控到老人的动作类型,则无法判断系统的运行是满足还是违反了该性质.

进一步地,可以描述性质“人员进入一个房间后一定要能监测到其位置(可能在嵌套的位置中)”如下:

$$\forall loc\in Location.loc[P]\vDash\exists user\in Person.(loc[user[hasActType(entering)]]\rightarrow\Box(\neg loc[user[hasActType(leaving)]]\rightarrow \nabla hasPerson(user))) \quad (3)$$

该性质同时出现时态算子□以及空间算子∇,其中,$\nabla hasPerson(user)$表示嵌套在 loc 中的某个 ambient(在该应用中即为嵌套的房间)中有人员 $user$.对于这个性质而言,在人员进入 loc 位置而尚未离开时,如无法监控到该用户,则该性质被违反;如果一直能在 loc 的子 ambient 监控到该用户,则无法判断系统的运行是满足还是违反了该性质.

为在运行时验证中更好地处理类似上述情况的时空性质,我们引入三值语义的思想,提出了 AL_3,在语义中引入了真假值以外的第 3 个值“无结论”,表示为“?”.

2.2 AL_3的三值语义定义

首先,定义几个与 AL_3 语义相关的概念和符号.

定义 2. 有限轨迹的 Ambient Calculus 进程转换模型定义为 $M=\langle \Pi,LR,TR,\Pi_0,\Lambda,AP,L\rangle$,其中,

- Π为进程(包括 ambient 这类特殊进程)的集合;
- $LR\subseteq\Pi\times\Pi$为两个进程的空间关系,$(P_1,P_2)\in LR$ 相当于 $P_1\downarrow P_2$;
- $TR\subseteq\Pi\times\Pi$为两个进程的时间顺序关系,$(P_1,P_2)\in TR$ 表示同一个 ambient 层次中不同时刻的两个进程

P_1,P_2,P_2 在运行时轨迹中发生在 P_1 之后;

- $\Pi_0\subseteq\Pi$为 ambient 树形结构中根节点进程的集合;
- Λ为名的集合(包括 ambient 名、一般进程名等);
- AP 为原子公式的集合;
- $L:\Pi\to 2^{AP}$ 为进程与原子公式的对应关系,对于进程 P,记集合 $L(P)\subseteq AP$ 表示 P 满足的原子公式集合.例如在第 1.3 节的例子中,原子公式 $isElder\in L(Bob[Reading1]),hasActType(reading)\in L(Reading1)$.

定义 3(真值). 进程 P 是否满足公式Φ的真值关系表示为$[P\vDash\Phi]=\{T,?,F\}$.

我们采用 Kleene[24]给出的三值逻辑语义对于$\neg,\vee$两个符号进行定义.

定义 4($\neg,\vee$的三值语义). 否定算子与析取算子的语义定义如下:

$$\neg(T)=F \qquad \neg(F)=T \qquad \neg(?)=?$$

$$F\vee T=T \qquad ?\vee T=T \qquad ?\vee F=?$$

以下,我们定义 AL_3 的满足语义(satisfaction semantics).

定义 5. AL_3 的满足语义定义如下(其中,$L(P),\Lambda,\Pi$等符号的定义见定义 2):

$$[P\vDash T]=T$$

$$[P\vDash a]=\begin{cases}T, & a\in L(P).\\ F, & \text{o.w.}\end{cases}$$

该表达式描述了原子公式的满足性.

$$[P\vDash\neg\Phi]=\neg([P\vDash\Phi])$$

$$[P\vDash\Phi_1\vee\Phi_2]=([P\vDash\Phi_1])\vee([P\vDash\Phi_2])$$

$$[P\vDash\forall x.\Phi]=\begin{cases}T, & \forall m\in\Lambda.[P\vDash\Phi\{x\leftarrow m\}]=T\\ F, & \exists m\in\Lambda.[P\vDash\Phi\{x\leftarrow m\}]=F\\ ?, & \text{o.w.}\end{cases}.$$

通过上述 3 个符号,我们可以用与命题逻辑、谓词逻辑相似的方式定义合取算子$\wedge$、蕴含算子$\to$、存在量词$\exists$的语义.

以下算子可用以描述时空相关的性质.

$$[P\vDash\Phi@n]=[n[P]\vDash\Phi]$$

$$[P\vDash n[\Phi]]=[\exists P'\in\Pi.P\equiv n[P']\wedge P'\vDash\Phi]$$

$$[P\vDash\Phi_1\,|\,\Phi_2]=\begin{cases}T, & \exists P',P''\in\Pi.(P\equiv P'\mid P'')\wedge([P'\vDash\Phi_1]=T)\wedge([P''\vDash\Phi_2]=T)\\ F, & \forall P',P''\in\Pi.(P\equiv P'\mid P'')\to((([P'\vDash\Phi_1]=T)\vee([P'\vDash\Phi_1]=?))\to([P''\vDash\Phi_2]=F))\\ ?, & \text{o.w.}\end{cases}.$$

$$[P\vDash\nabla\Phi]=\begin{cases}T, & \exists P'\in\Pi.(P\downarrow^{*}P')\wedge([P'\vDash\Phi]=T)\\ F, & \forall P'\in\Pi.(P\downarrow^{*}P')\to([P'\vDash\Phi]=F)\\ ?, & \text{o.w.}\end{cases}$$

$$[P\vDash\Diamond\Phi]=\begin{cases}T, & \exists P'\in\Pi.(P,P')\in TR\wedge([P'\vDash\Phi]=T)\\ ?, & \text{o.w.}\end{cases}$$

我们可以定义用$\neg\Diamond\neg$定义时态算子$\Box$(总是),该算子语义亦可解释如下:

$$[P\vDash\Box\Phi]=\begin{cases}F, & \exists P'\in\Pi.(P,P')\in TR\wedge([P'\vDash\Phi]=F)\\ ?, & \text{o.w.}\end{cases}.$$

3 普适计算应用的运行时验证方法

本节讨论基于 AL_3 进行运行时验证的方法,我们首先给出 AL_3 的检验算法,之后将讨论普适计算应用运行时监控器的实现.

3.1 AL_3的性质检验

(1) 运行时轨迹规约

根据定义2,使用Ambient Calculus对于运行时状态进行规约后,将形成具有时间顺序的轨迹,轨迹中的每个部分是由一个时刻的系统运行状态(包括各个实体、它们之间的关系、符合的性质)转换而成的一个 ambient 树,由于一个 ambient 表达式可能无法转换为统一的根节点的 ambient 树(即没有统一的最外层 ambient,如 *room*1[]|*room*2[]),我们统一增加一个最外层 ambient 作为树的根节点以确保 ambient 树的形成,这些根节点构成了Π_0集合.

图 1 给出了运行轨迹的实例,这个例子描述了在用户 Bob 在一个房间里生活的片段,Bob 首先在客厅里阅读,之后进入浴室开始洗澡.在这样轨迹中,根节点集合为Π_0={*s*0,*s*1,*s*2}.

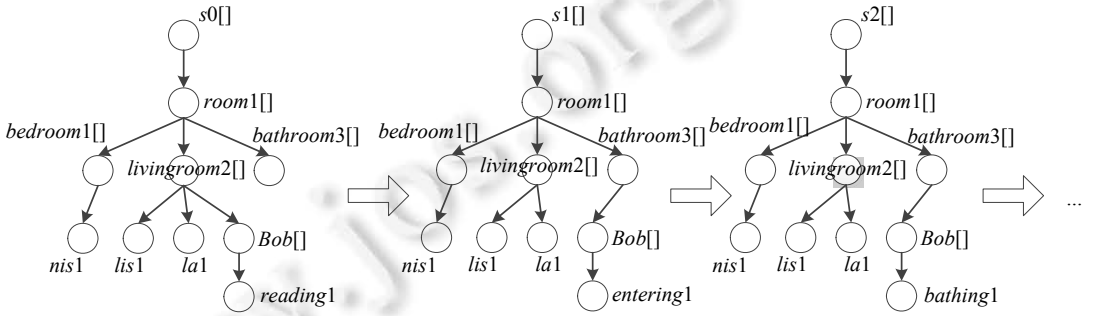


Fig.1 An example of runtime traces depicted by Ambient Calculus

图 1 Ambient Calculus 描述的运行轨迹示例

基于这样的运行轨迹规约,进程的空间关系(*LR* 集合)可通过 ambient 树节点的父子关系完成构造.时间顺序关系(*TR* 集合)可通过不同状态中的相同进程名进行构造,在图 1 的例子中,尽管 Bob 进程发生了移动,但是状态 2 的 Bob 进程仍然是状态 1 中 Bob 进程的后续进程.

各个进程满足的原子公式集合 *L* 基于应用需求和运行状态进行设置.

(2) 性质的预处理

为方便起见,我们提供了使用由基本算子定义的其他算子(如∧,→,∃,□)描述性质的能力;同时,为处理便捷,我们规定:描述公式时,需将连续的合取项($\Phi_1\wedge\Phi_2\wedge\ldots\wedge\Phi_n$)以及连续的析取项($\Phi_1\vee\Phi_2\vee\ldots\vee\Phi_n$)置入括号中.

在执行检验之前,需要对非基本算子进行预处理,转化为基本算子.

算法 1. 性质预处理 preprocess.

Input:公式,Φ;

Output:转换后的公式,Φ'.

1: Φ':=Φ;

2: Φ'中所有$\exists x.\Phi_1$转为$\neg(\forall x.\neg\Phi_1)$;

3: Φ'中所有$\Box\Phi_1$转为$\neg(\Diamond\neg\Phi_1)$;

4: Φ'中所有$\Phi_1\rightarrow\Phi_2$转为$\neg\Phi_1\vee\Phi_2$

5:将Φ'各个部分(包括◇,∀,∇等符号内的子公式)中的所有合取式$\Phi_1\wedge\Phi_2\wedge\ldots\wedge\Phi_n$转为

$$\neg(\neg\Phi_1\vee\neg\Phi_2\vee\ldots\vee\neg\Phi_n);$$

6: **return** Φ';

3) 性质检验算法

我们在文献[21]给出的算法基础上设计了基于 AL_3 公式的性质检验算法,其目的是判断进程 *P* 是否满足公式Φ($P\vDash\Phi$).需要注意的是:由于我们引入了三值逻辑,算法的返回值不再是布尔型(Boolean)结果,而是一个由{*T*,?,*F*}组成的枚举类型 AL3Result.在算法实现中,为处理方便,我们分别设 *T*=1,?=0,*F*=−1,这样,AL3Result 的结

果是可以进行数值上的大小比较的.

算法 2. 性质检验算法 check.

Input:进程,P;公式,Φ;

Output:AL3Result 类型的结果,*result*.

```
1:    switch (公式Φ呈现的形式):
2:    case (T)          : return T;
3:    case (a∈AP)       : if (a∈L(P)) return T; else return F;   //公式Φ是 P 满足的原子公式
4:    case (¬Φ')        : return –check(P,Φ'));
5:    case (Φ1∨Φ2)      : return max{check(P,Φ1),check(P,Φ2)};
6:    case (n[Φ'])      : if (P.name=n)
7:                            if (Φ'是 k 个公式组成的Φ1|Φ2|…|Φk 的形式)
8:                                if (存在 P1…PK,满足(P,Pi)∈LR,且 check(Pi,Φi)皆为 T)
9:                                    return T;
10:                               else if (存在 P1...PK,满足(P,Pi)∈LR,且 check(Pi,Φi)皆为 T 或?)
11:                                   return ?;
12:                               else return F;
13:                           else return max{check(Pi,Φ')|(P,Pi)∈LR}
14:                       else return F;
15:   case (Φ'@n)       : if (存在进程 P',使得(P',P)∈LR AND P'.name=n)
16:                           return check(P',Φ');
17:                       else return F;
18:   case (∀x.Φ')      : return min{check(P,Φ'{x←fn})|fn∈Λ};
19:   case (∇Φ')        : maxTemp:=check(P,Φ');
20:                       for each P'满足(P,P')∈LR do begin
21:                           maxTemp:=max{maxTemp,check(P',∇Φ')};
22:                       return maxTemp;
23:   case(◇Φ')         : if(check(P,Φ')=T) return T;
24:                       for each P'满足(P,P')∈TR do begin
25:                           if (check(P',Φ')=T) return T;
26:                       return ?;
27: end switch
```

该算法按照 AL_3 的语义递归地处理每个算子,最终返回三值逻辑表示的 $P\models\Phi$性质的满足性.在实际使用时,验证一条性质往往需要调用多次,如验证第 2.1 节的性质(1)这类表示为$\forall loc\in Location.loc[P]\models\Phi$的性质时,需要用轨迹中所有类型为 Location 的各个进程分别进行验证.

从算法复杂度的角度看,与标准 Ambient Logic 检验算法[21]一样,本算法可能需要从一组 $P_1|P_2|\ldots|P_m$ 的进程中选出 k 个以检验公式$\Phi_1|\Phi_2|\ldots|\Phi_k$(第 8 行、第 10 行),这样的排列问题理论上的最坏时间复杂度是阶乘级的,但是在实现过程中,我们进行了一些优化,主要包括:

- 记忆化搜索:在搜索过程中,如果已经检验过 P_i 是否满足公式Φ_j,则将$[P_i\models\Phi_j]$值计入数组,以避免重复计算的时间开销;
- 剪枝:当出现$[P_i\models\Phi_j]=F$ 时,相关的排列可以不再继续搜索;如果$[P_i\models\Phi_j]$=?且当前已经找到结果为?的排列,同样可以不再继续搜索;如果找到结果为 T 的排列,则直接返回;
- 随机化:随机选择下一个搜索的进程,以避免特殊格式进程、性质导致的搜索开销过大情况.

通过以上优化,结合实际应用规模,我们的验证算法时间开销基本可以接受(参见第 4.2 节实验).

3.2 普适计算应用监控器

我们通过普适计算应用监控器执行运行时验证、监控普适计算的运行状态是否满足给定的性质.根据应用需求的不同,这一监控过程可能采取应用定时推送状态(push)的方式,也可能由监控器主动获取(pull).监控器的结构如图 2 所示,分为以下几个部分.

- 应用状态翻译器:该组件接收普适计算应用运行轨迹中的一系列状态,将其转为 Ambient Calculus 的形式系统,从实现上说,即是补充根节点、构造若干 ambient 树结构,树中的每个节点皆是一个进程(包括 ambient 这种特殊进程).具体的层次关系、转换方式和应用类型有关,我们的前期工作[19]有针对特定应用进行形式规约的例子;
- 进程转换系统构造器:对于构造完成的 ambient 树进行分析,建立本次监控对应的有限轨迹进程转换系统,包括空间关系 *LR* 集合、时间顺序关系 *TR* 集合等;
- 性质预处理器:使用算法 1 对于性质进行预处理,使得公式中只含有基本算子,将这样的性质保存、推送进性质验证器;
- 性质检验器:使用算法 2 对于普适计算的运行轨迹是否满足给定的性质进行检验,以三值逻辑的形式输出检验结果,对于一个轨迹而言,只要出现进程的结果为 F 则输出 F;否则,如果出现进程的结果为?则输出?,如果所有相关进程检验结果皆为 T 则输出 T;
- 监控结果反馈设施:接收到检验结果后,将相应结果反馈给计算系统.需要注意的是:对于监控结果(尤其是结果为?的情况下)的解释是和具体性质有关的,需要设计应用性质的人员实现相应的解释器.

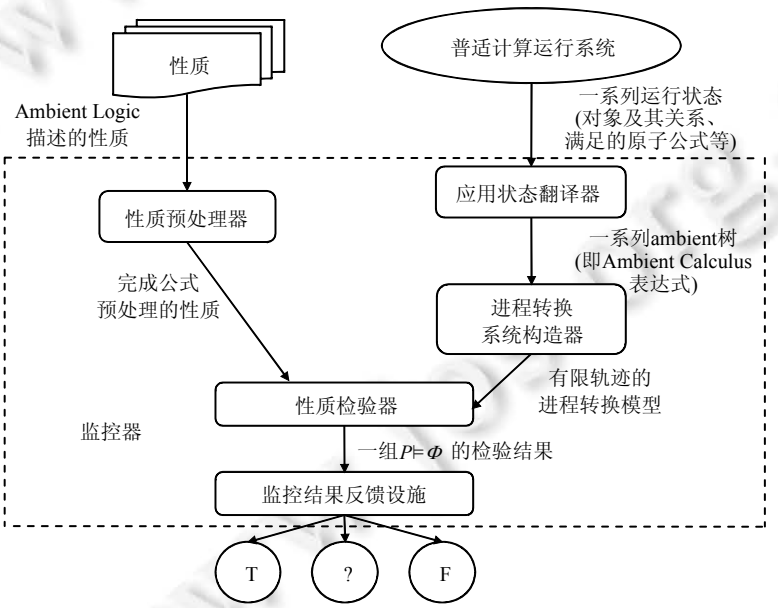


Fig.2 Structure of the monitor for pervasive computing applications

图 2 普适计算应用监控器的结构

4 评 估

本节首先通过一个案例讨论将 AL_3 方法应用于普适计算应用运行时验证的有效性,再通过一个性能评估实验讨论该方法的效率.

4.1 案例研究

老人看护系统是一类常见的普适计算应用,老人在普适计算环境中自主地进行日常生活,计算系统感知老人的身体状况、位置、动作等信息,也感知环境的温度、湿度、光线强度、噪音等信息,根据应用需求调整环境的状态(如开灯),或者给老人及其家人一些相关的提示.

(1) 应用运行状态规约

我们通过监控器中的应用状态翻译器将计算系统转为形式系统.采用 Ambient Calculus 作为形式工具对于应用的运行状态轨迹进行规约,并根据应用需求定义进程满足的原子性质,以图 1 所示的运行轨迹为例,其 3 个状态可分别规约为:

- $s0[room1[bedroom1[nis1]|livingroom2[lis1|la1|Bob[reading1]]|bathroom3[]]]$
  where $Bob[P]\vDash isElder, reading1\vDash hasActType(reading)$;
- $s1[room1[bedroom1[nis1]|livingroom2[lis1|la1]|bathroom3[Bob[entering1]]]]$
  where $Bob[P]\vDash isElder, entering1\vDash hasActType(entering)$;
- $s2[room1[bedroom1[nis1]|livingroom2[lis1|la1]|bathroom3[Bob[bathing1]]]]$
  where $Bob[P]\vDash isElder, bathing1\vDash hasActType(bathing)$.

当然,此外还有一些进程满足其他的原子性质,例如,“*nis*1 是噪音传感器”、“*lis*1 是光线传感器”可描述如下:

$$nis1\vDash sensingFea(noiseIntensity), lis1\vDash sensingFea(lightIntensity).$$

进一步地,我们构造有限轨迹的进程转换系统,用进程 P 满足的原子性质构造 $L(P)$集合,通过 ambient 层次关系构造 LR 集合,通过不同时刻的进程名构造 TR 集合.

(2) 运行时验证

与普通的二值 Ambient Logic 一样,AL_3 的性质检验器也可用于针对一个特定时刻的运行状态,检验不含时态算子的性质.当然,其特点和优势在于可以针对运行时的有限运行轨迹,检验同时含有空间包含关系和时间顺序关系的时空性质.

以第 2.1 节的性质(1)为例,即“如果一位老人进入浴室洗澡,则应当监测到他离开浴室的动作(以免洗澡时发生事故)”,如果将该性质输入性质检验器,对于图 1 的 3 个状态构成的有限轨迹进行检验,返回结果为?(因为 Bob 尚未离开浴室).该结果将由和性质对应的解释器进行解释,例如出现“?”时间过长需要告知家人.

如果在图 1 的这 3 个状态之后出现了

$$s3[room1[bedroom1[nis1]|livingroom2[lis1|la1]|bathroom3[Bob[leaving1]]]]$$
$$\text{where } Bob[P]\vDash isElder,\ bathing1\vDash hasActType(leaving)$$

这一状态,则说明 Bob 已经离开了浴室,性质检验结果为 T.

对于 2.1 节的性质(2),即“老人的动作始终要被监测到”,如果将该性质输入性质检验器,对于图 1 的 3 个状态构成的有限轨迹进行检验,返回结果为“?”,该结果将由和性质对应的解释器进行解释,这是系统中有老人存在时的正常情况.如果系统中没有老人存在,则返回结果将是 T(蕴含式前设 *isElder* 不成立).如果老人的动作未被监测到或动作类型无法被识别(老人 ambient 中没有表示动作的进程),则返回结果为 F.

对于 2.1 节的性质(3),即“人员进入一个房间后一定要能监测到其位置(可能在嵌套的位置中)”,如果将该性质输入性质检验器,对于图 1 的三个状态构成的有限轨迹进行检验,返回结果为“?”,该结果将由和性质对应的解释器进行解释,这是系统中有人员进入某个位置后的正常情况;如果系统中没有出现“进入(*entering*)”动作,则返回结果将是 T;如果有人员进入之后尚未离开却无法监测到其位置,则返回结果为 F;如果监测到“离开(*leaving*)”动作,则由解释器停止该性质的检验.

4.2 性能实验

本节对于提出并实现的普适计算应用时空性质运行时验证方法的运行效率进行实验分析.

(1) 实验设计

我们设计了一种模拟的普适计算环境,包括具有空间包含关系的房间,随机设置的用户在这些房间中进行动作,房间中也配置有相应的模拟传感器、设备等,模拟产生数据,包括用户位置、用户动作、环境状态数据等,这些数据在系统中以 RDF 三元组存储.在本实验中,我们以每个运行状态产生的三元组数目体现应用规模,如一个含有 5 个房间、5 类传感器、4 个用户的家居场景(如老人看护)每个运行状态产生 50 个左右的三元组;一个含有 500 个房间、10 类传感器、300 个用户的办公楼场景每个运行状态产生 5 000 个左右的三元组.在运行时,监控器中的应用状态翻译器将当前运行状态翻译为 Ambient Calculus 表示的形式系统.

我们用 Ambient Logic 描述了待验证的运行时性质.在这些性质中,一半性质设定为较为简单,包含 1~2 个原子公式(如第 2.1 节的性质(2));另一半较为复杂,包含 3 个或 3 个以上原子公式(如第 2.1 节的性质(1)).实验中涉及的性质基本皆是类似于第 2.1 节的 3 个例子,针对普遍的用户、位置、设备等实体进行描述而不是针对一个预先给定的特定实体,对于这类性质进行验证的实验,可使得实验结果更具有一般性.

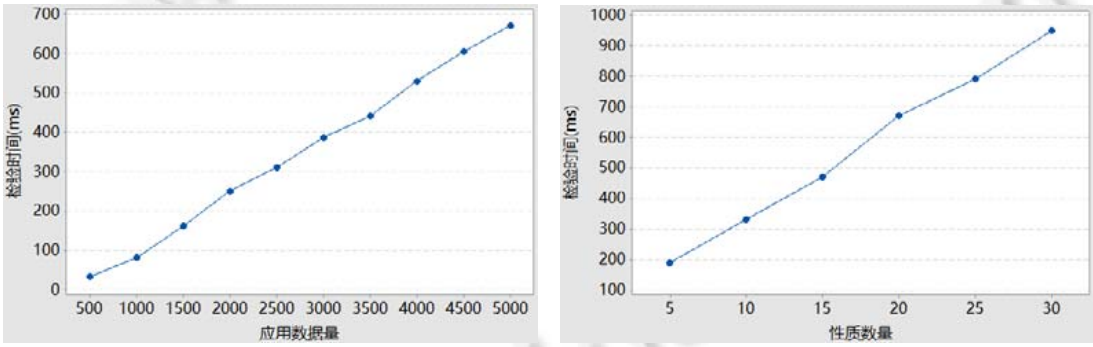
我们分别进行了以下两组实验.

- 实验 1:固定性质数量为 20 条(其中 10 条涉及时空性质),以不同规模的模拟数据量(即 RDF 三元组个数)进行实验,分别在每次运行状态数据量为 500 个~5000 个三元组的不同规模下记录运行时验证的时间开销,考察应用状态数据量对于验证时间的影响;
- 实验 2:固定每次运行状态数据规模为 5 000,以性质数量分别为 5~30(其中一半左右涉及时空性质)的情况进行实验,记录运行时验证的时间开销,考察性质数量对于验证时间的影响.

上述实验运行以 4G RAM,Intel 3.2GHz 双核 CPU 的计算机作为服务器运行,网络带宽等因素保持不变,忽略其影响.每组实验进行 20 次取平均值.

(2) 实验结果与讨论

图 3(a)和图 3(b)分别展示了两组实验的结果.



(a) 实验 1:应用数据量与时间开销的关系 (b) 实验 2:性质数量与时间开销的关系

Fig.3 Results of the performance evaluation

图 3 性能评估实验的结果

从这个结果中我们可以看出:随着计算系统数据规模的扩大、性质数量的增加,运行时验证的时间开销会相应地增加.尽管时间开销仍可能受到这两个因素之外的其他因素(如具体应用相关的性质复杂程度,本实验已尽可能地设置了多种类型的性质)影响,但是通过本实验可以看出:本文提出的运行时验证方法的时间开销是可以接受的,每次运行状态产生 5 000 个左右三元组的普适计算系统、30 条性质已可满足一个较大规模的智能环境的要求,在这样的情况下,运行时验证的时间开销在 1s 以内.

5 相关工作

普适计算领域的运行时验证技术已得到国内外研究者广泛关注,已有若干工作提出了针对医疗护理[25]、自治车辆[26]等典型普适计算应用,以及 CPS(cyber-physical system)[27]、云计算[28]等相关计算模式的运行时验

证方法.这些方法基本都是使用时态逻辑的各种变种作为逻辑基础进行验证,没有空间性质方面的考虑.

对于空间性质的验证而言,使用 Ambient Calculus 进行系统状态规约、使用 Ambient Logic 进行性质规约是一种常用的方法,已有 Ranganathan[17],Coronato[18]等研究者进行了这样的尝试.针对 Ambient Logic 在有限轨迹时间性质方面表达能力的不足,Coronato[18]以及我们的前期工作[19]分别给出了使用绝对时间定量地进行时间性质表达的做法,相比于这类尝试,采用定性的三值逻辑语义更加具有一般性.

引入三值逻辑是在轨迹有限或者状态不全情况下进行时态逻辑性质验证的常用手段.Brus 等人[29]将三值语义引入了局部克里普克结构(partial Kripke structure);Bauer 等人[20]提出了三值 LTL 语义,该语义被 Yu 等人[27]采用进行 CPS 的验证;Wei 等人[23]提出了三值 CTL 语义,并用以验证普适计算中的部分性质.这一系列工作对我们有所启发,但是这些工作仅有时间性质的验证手段,对于时空性质的验证,仍需要给出新的形式工具.

6 总结与展望

本文针对普适计算应用的时空性质运行时验证问题进行研究:首先,将三值语义引入 Ambient Logic 提出了 AL_3 方法;在此基础上,进一步地设计实现了支持基于 AL_3 进行运行时验证的性质检验算法和运行时监控器.在未来工作中,我们考虑进行以下方向的推进:首先,考虑改进验证算法,如采用滑动窗口等方法减少空间开销;其次,考虑进一步地在更真实的场景中进行应用的实践,并面向真实应用的需求,对于监控器的设计、性质的定义方式进行改进.

References:

[1] Weiser M. The computer for the 21st century. Scientific American, 1991,261(30):94−104. [doi: 10.1145/329124.329126]

[2] Satyanarayanan M. Pervasive computing: Vision and challenges. IEEE Personal Communications, 2001,8(4):10−17. [doi: 10.1109/98.943998]

[3] Dey AK, Abowd GD, Salber D. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. Human-Computer Interaction, 2001,16(2):97−166. [doi: 10.1207/S15327051HCI16234_02]

[4] Román M, Hess C, Cerqueira R, Ranganathan A, Campbell RH, Nahrstedte K. A middleware infrastructure for active spaces. IEEE Pervasive Computing, 2002,1(4):74−83. [doi: 10.1109/MPRV.2002.1158281]

[5] Kulkarni D, Ahmed T, Tripathi A. A generative programming framework for context-aware CSCW applications. ACM Trans. on Software Engineering and Methodology (TOSEM), 2012,21(2):11. [doi: 10.1145/2089116.2089121]

[6] Gu T, Pung HK, Zhang DQ. Toward an OSGi-based infrastructure for context-aware applications. IEEE Pervasive Computing, 2004,3(4):66−74. [doi: 10.1109/MPRV.2004.19]

[7] Arcelus A, Jones MH, Goubran R, Knoefel F. Integration of smart home technologies in a health monitoring system for the elderly. In: Proc. of the 21st IEEE Int'l Conf. on Advanced Information Networking and Applications (AINA) Workshops. 2007. 820−825. [doi: 10.1109/AINAW.2007.209]

[8] Julien C, Roman GC. Egospaces: Facilitating rapid development of context-aware mobile applications. IEEE Trans. on Software Engineering (TSE), 2006,32(5):281−298. [doi: 10.1109/TSE.2006.47]

[9] Artho C, Barringer H, Goldberg A, Havelund K, Khurshid S, Lowry M, Pasareanu C, Rosu G, Sen K, Visser W. Combining test case generation and runtime verification. Theoretical Computer Science (TCS), 2005,336(2):209−234. [doi: 10.1016/j.tcs.2004.11.007]

[10] Zhang X, Dong W, Qi ZC. Conflicts detection in runtime verification based on AOP. Ruan Jian Xue Bao/Journal of Software, 2011, 22(6):1224−1235 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4016.htm [doi: 10.3724/SP.J.1001.2011.04016]

[11] Bakhouya M, Campbell R, Coronato A, De Pietro G, Ranganathan A. Introduction to special section on formal methods in pervasive computing. ACM Trans. on Autonomous and Adaptive Systems (TAAS), 2012,7(1):6. [doi: 10.1145/2168260.2168266]

[12] Ranganathan A, Campbell RH. An infrastructure for context-awareness based on first order logic. Personal and Ubiquitous Computing (PUC), 2003,7(6):353−364. [doi: 10.1007/s00779-003-0251-x]

[13] Pnueli A. The temporal logic of programs. In: Proc. of 18th Annual Symp. on Foundations of Computer Science. IEEE, 1977. 46−57. [doi: 10.1109/SFCS.1977.32]

[14] Emerson EA, Halpern JY. Decision procedures and expressiveness in the temporal logic of branching time. In: Proc. of the 14th Annual ACM Symp. on Theory of Computing. ACM Press, 1982. 169−180. [doi: 10.1145/800070.802190]

[15] Cardelli L, Gordon AD. Mobile ambients. In: Proc. of the Int'l Conf. on Foundations of Software Science and Computation Structure. Springer Berlin Heidelberg, 1998. 140−155. [doi: 10.1007/BFb0053547]

[16] Cardelli L, Gordon AD. Ambient logic. In: Proc. of the Mathematical Structures in Computer Science. 2003.
[17] Ranganathan A, Campbell RH. Provably correct pervasive computing environments. In: Proc. of the 6th IEEE Int'l Conf. on Pervasive Computing and Communications (PerCom). 2008. 160−169. [doi: 10.1109/PERCOM.2008.116]
[18] Coronato A, De Pietro G. Formal specification of wireless and pervasive healthcare applications. ACM Trans. on Embedded Computing Systems (TECS), 2010,10(1):12. [doi: 10.1145/1814539.1814551]
[19] Li XS, Tao XP, Lü J, Song W. Specification and runtime verification for activity-oriented context-aware applications. Ruan Jian Xue Bao/Journal of Software, 2017,28(5):1167−1182 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5215.htm [doi: 10.13328/j.cnki.jos.005215]
[20] Bauer A, Leucker M, Schallhart C. Runtime verification for LTL and TLTL. ACM Trans. on Software Engineering and Methodology (TOSEM), 2011,20(4):14. [doi: 10.1145/2000799.2000800]
[21] Charatonik W, Zilio SD, Gordon AD, Mukhopadhyay S, Talbot J. Model checking mobile ambients. Theoretical Computer Science, 2003,308(1-3):277−331. [doi: 10.1016/S0304-3975(02)00832-0]
[22] Li XS, Tao XP, Lü J. Programming method and formalization for activity-oriented context-aware applications. In: Proc. of the 12th Int'l Conf. on Ubiquitous Intelligence and Computing (UIC). IEEE, 2015. 174−181. [doi: 10.1109/UIC-ATC-ScalCom-CBDCom-IoP.2015.48]
[23] Wei H, Huang Y, Cao J, Ma XX, Lü J. Formal specification and runtime detection of temporal properties for asynchronous context. In: Proc. of the 10th Int'l Conf. on Pervasive Computing and Communications (PerCom). IEEE, 2012. 30−38. [doi: 10.1109/PerCom.2012.6199846]
[24] Kleene S. Introduction to Metamathematics. Wolters-Noordhoff, 1971.
[25] Jiang Y, Liu H, Kong H, Wang R, Hosseini M, Sun J, Sha L. Use runtime verification to improve the quality of medical care practice. In: Proc. of the 38th IEEE/ACM Int'l Conf. on Software Engineering Companion. ACM Press, 2016. 112−121.
[26] Kane A, Chowdhury O, Datta A, Koopman P. A case study on runtime monitoring of an autonomous research vehicle (ARV) system. In: Proc. of the 6th Int'l Conf. on Runtime Verification. LNCS 9333. Springer Int'l Publishing, 2015. 102−117. [doi: 10.1007/978-3-319-23820-3_7]
[27] Yu K, Chen Z, Dong W. A predictive runtime verification framework for cyber-physical systems. In: Proc. of the 8th IEEE Int'l Conf. on Software Security and Reliability-Companion. IEEE, 2014. 223−227. [doi: 10.1109/SERE-C.2014.43]
[28] Zhou J, Chen Z, Wang J, Zheng Z, Dong W. A runtime verification based trace-oriented monitoring framework for cloud systems. In: Proc. of IEEE Int'l Symp. on Software Reliability Engineering Workshops. IEEE, 2014. 152−155. [doi: 10.1109/ISSREW.2014.84]
[29] Bruns G, Godefroid P. Model checking partial state spaces with 3-valued temporal logics. In: Proc. of the Int'l Conf. on Computer Aided Verification. Berlin, Heidelberg: Springer-Verlag, 1999. 274−287. [doi: 10.1007/3-540-48683-6_25]

附中文参考文献:

[10] 张献,董威,齐治昌.基于 AOP 的运行时验证中的冲突检测.软件学报,2011,22(6):1224−1235. http://www.jos.org.cn/1000-9825/4016.htm [doi: 10.3724/SP.J.1001.2011.04016]
[19] 李暄松,陶先平,吕建,宋巍.面向动作的上下文感知应用的规约与运行时验证.软件学报,2017,28(5):1167−1182. http://www.jos.org.cn/1000-9825/5215.htm [doi: 10.13328/j.cnki.jos.005215]

李暄松(1985−),男,山东博兴人,博士,讲师,CCF 专业会员,主要研究领域为软件工程与方法学,形式化方法,普适计算技术.

宋巍(1981−),男,博士,副教授,CCF 高级会员,主要研究领域为软件工程与方法学,形式化方法,服务计算.

陶先平(1970−),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为软件中间件技术,网构软件方法学,普适计算技术.