软件学报 ISSN 1000-9825, CODEN RUXUEW
2024,35(11):4993−5015 [doi: 10.13328/j.cnki.jos.007031] [CSTR: 32375.14.jos.007031]

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

面向物联网设备移动与通信行为的建模及验证*

刘靖宇[1], 李暄松[1,2], 陈芝菲[1,2], 叶海波[3], 宋 巍[1]

[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)

[3](南京航空航天大学 计算机科学与技术学院, 江苏 南京 211106)

通信作者: 李暄松, E-mail: lixs@njust.edu.cn

摘 要: 物联网设备的使用范围正在不断扩张. 模型检测是提升这类设备可靠性和安全性的有效手段, 但常用的模型检测方法不能很好地刻画这类设备常见的跨空间移动和通信行为. 为此, 提出一种面向物联网设备移动与通信行为的建模及验证方法, 以实现对这类设备时空相关性质的验证. 通过将推拉动作和全局通信机制融入 ambient calculus, 提出全局通信移动环境演算 (ACGC) 并给出了 ACGC 对 ambient logic 的模型检测算法; 在此基础上, 提出描述物联网设备移动和通信行为的移动通信建模语言 (MLMC), 并给出将 MLMC 描述转换为 ACGC 模型的方法; 进一步地, 实现模型检测工具 ACGCCk 以验证物联网设备的性质是否得到满足, 并通过一些优化加快检测速度; 最后, 通过案例研究和实验分析阐明所提方法的有效性.

关键词: 模型检测; 物联网; 形式化验证; 建模语言

中图法分类号: TP311

中文引用格式: 刘靖宇, 李暄松, 陈芝菲, 叶海波, 宋巍. 面向物联网设备移动与通信行为的建模及验证. 软件学报, 2024, 35(11): 4993–5015. http://www.jos.org.cn/1000-9825/7031.htm

英文引用格式: Liu JY, Li XS, Chen ZF, Ye HB, Song W. Modeling and Verification for Mobile and Communication Behaviors of IoT Devices. Ruan Jian Xue Bao/Journal of Software, 2024, 35(11): 4993–5015 (in Chinese). http://www.jos.org.cn/1000-9825/7031.htm

Modeling and Verification for Mobile and Communication Behaviors of IoT Devices

LIU Jing-Yu[1], LI Xuan-Song[1,2], CHEN Zhi-Fei[1,2], YE Hai-Bo[3], SONG Wei[1]

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

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

[3](College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China)

Abstract: The utilization range of Internet of Things (IoT) devices is expanding. Model checking is an effective approach to improve the reliability and security of such devices. However, the commonly adopted model checking methods cannot well describe the cross-space movement and communication behavior common in such devices. To this end, this study proposes a modeling and verification method for the mobile and communication behavior of IoT devices to verify their spatio-temporal properties. Additionally, push/pull action and global communication mechanism are integrated into ambient calculus to propose the ambient calculus with global communication (ACGC) and provide a model checking algorithm for ACGC against the ambient logic. Then, the modeling language for mobility and communication (MLMC) is put forward to describe mobile and communication behavior of IoT devices. Additionally, a method to convert the MLMC-based description into an ACGC model is given. Furthermore, a model checking tool ACGCCk is implemented to verify whether the properties of IoT devices are satisfied. Meanwhile, some optimizations are conducted to accelerate the checking. Finally, the effectiveness of the proposed method is demonstrated by case study and experimental analysis.

Key words: model checking; Internet of Things (IoT); formal verification; modeling language

---

* 基金项目: 国家自然科学基金 (61702263, 61761136003); CCF-华为创新研究计划 (CCF-HuaweiFM2021004)
收稿时间: 2022-09-18; 修改时间: 2023-05-07; 采用时间: 2023-08-25; jos 在线出版时间: 2024-02-05
CNKI 网络首发时间: 2024-02-09

物联网是在互联网的基础上进行扩展和延伸、形成的物物相连的网络. 随着无线通信技术和环境感知技术的不断发展, 物联网在人们生活中的应用已经越来越普遍. 研究显示, 2021 年全球物联网市场规模达到 1 579 亿美元, 同比增长 22.4%[1]. 可以说, 物联网的出现和发展, 打破了时间和地点的限制, 促进了人-机-物三元融合的万物互联时代的来临. 随着物联网逐步融入人们的生产和生活, 物联网设备的可靠性也愈来愈值得重视, 尤其是在诸如智慧交通、老人看护等安全攸关的场景中, 物联网设备的行为将直接关系到用户的生命与财产安全. 传统的软件测试方法对于测试用例的选取往往不够完全, 而形式化验证利用数学的方法验证系统关于特定性质的满足性, 可以较好地解决传统测试方法存在的覆盖率不足的问题.

在物联网设备运行场景中, 实体 (如设备、用户、物品等) 常具有移动和通信的能力. 以酒店送餐机器人应用场景为例, 送餐机器人和电梯均可以移动, 并且送餐机器人与电梯及旅客之间存在着通信交互. 在本文中, 我们重点关注物联网设备等实体的以下几种行为.

(1) 主动移动, 如送餐机器人自主地进入电梯、离开电梯.

(2) 被动移动, 如送餐机器人到达客房门口时, 餐点被送餐机器人给出.

(3) 相互通信, 并且这种通信经常发生在位于不同空间的实体之间, 如送餐机器人呼叫电梯.

这类场景中需要满足的性质常与时间和空间关系有关. 现有的一些模型检测方法在这类场景下有以下不足.

(1) 常用的时序逻辑 (如 LTL[2]和 CTL[3]等) 缺乏描述空间相关性质的算子, 在涉及空间性质上的表达能力不足.

(2) Ambient logic (AL)[4]是一种对时空相关性质有一定的描述能力的模态逻辑, 但是与其对应的进程代数 ambient calculus (AC)[5]的原语比较简单: 虽然 AC 的能力原语 *in M* 和 *out M* 可以刻画实体在不同空间之间的主动移动, 但缺少原语去刻画它们的被动移动, 除此之外 AC 的通信原语 (*n*). *P* 和⟨*M*⟩只支持本地通信, 不能较好地刻画实体之间跨空间的通信.

(3) 已有一些针对 AC 语法语义扩展的研究, 相关学者提出了 mobile safe ambients (SA)[6]、boxed ambients (BA)[7]、push and pull ambient calculus (PAC)[8]和 malculus of mobility and communication (CMC)[9]等扩展演算, 但是上述研究仅停留在理论阶段, 缺乏系统性的形式化验证方法.

针对上述问题, 本文提出了一种针对物联网设备移动与通信行为的建模及验证方法, 通过将物联网设备的行为建模为移动通信建模语言 (modeling language for mobility and communication, MLMC) 模型并转换为全局通信移动环境演算 (ambient calculus with global communication, ACGC) 模型、将需要满足的性质规约为 AL 公式、使用模型检测工具 ACGC checker (ACGCCk) 对 ACGC 模型和 AL 公式进行检测, 实现了上述场景下时空相关性质的自动化验证. 本文的主要贡献如下.

(1) 提出了全局通信移动环境演算 (ACGC), 并给出了 ACGC 对 AL 的模型检测算法和正确性证明, 由此证明了 ACGC 对 AL 的模型检测问题是可判定的.

(2) 提出了描述物联网设备移动与通信行为的移动通信建模语言 (MLMC), 并给出了从 MLMC 到 ACGC 的转换方法.

(3) 实现了 ACGC 对 AL 的模型检测工具 ACGCCk, 并引入了一些优化以加快检测速度.

(4) 通过案例研究说明了本文提出的方法的有效性, 通过实验分析说明了 ACGCCk 具有实际应用的价值.

本文第 1 节介绍背景知识. 第 2 节提出全局通信移动环境演算 ACGC, 并给出 ACGC 对 AL 的模型检测算法以及其正确性证明. 第 3 节提出移动通信建模语言 MLMC, 并给出将 MLMC 模型转换为 ACGC 模型的方法. 第 4 节介绍 ACGCCk 工具的实现以及性能的优化. 第 5 节通过一个机器人送餐场景的案例研究说明了所提方法的有效性, 并通过实验分析了 ACGCCk 工具的性能. 第 6 节回顾了一些相关工作. 第 7 节总结全文并进行展望.

1 预备知识

本节首先介绍 AC 及其相关扩展, 这是我们针对物联网设备使用场景设计 ACGC 的基础, 然后介绍用以描述场景性质的 AL.

1.1 Ambient calculus (AC)

AC[5]是由 Cardelli 等人提出的一种进程代数. AC 以 π-calculus[10]为基础, 引入了“ambient”的概念用以刻画计算发生的有界场所, 如网页、虚拟地址空间、笔记本电脑等. Ambient 不仅可以作为整体移动, 其内部还可以嵌套其他 ambient. AC 的语法如下:

$P, Q ::= (\nu n)\ P \mid \mathbf{0} \mid P \mid Q \mid !P \mid M[P] \mid M.P \mid (n).P \mid \langle M \rangle$

$M, N ::= n \mid in\ M \mid out\ M \mid open\ M \mid \varepsilon \mid M.N$

其中, M, N 表示能力, n 表示名字, $in\ M$ 表示进入名字为 M 的 ambient 的动作, $out\ M$ 表示离开名字为 M 的 ambient 的动作, $open\ M$ 表示打开名字为 M 的 ambient 的动作, ε 表示空能力, $M.N$ 表示能力的串行. P, Q 表示进程, $(\nu n)P$ 表示名字 n 仅在进程 P 内部使用, $\mathbf{0}$ 表示空进程, $P \mid Q$ 表示进程 P 和 Q 并行, $!P$ 表示进程 P 的无限复制, $M[P]$ 表示名字为 M 且内部进程为 P 的 ambient, $M.P$ 表示先执行能力 M 然后继续进程 P, $(n).P$ 表示输入能力并绑定在名字 n 上然后继续进程 P, $\langle M \rangle$表示输出能力 M.

AC 的语义由进程的二元关系结构同余 ($\equiv$) 和归约 ($\rightarrow$) 组成. 结构同余表示进程的语义相同 (尽管语法可能不同), 例如 $P \mid Q \equiv Q \mid P$ 表明了进程的并行满足交换律. 归约表示进程经过一步演化可能得到的结果, 例如 $n[in\ m.P \mid Q] \mid m[R] \rightarrow m[n[P \mid Q] \mid R]$ 表示能力 $in\ m$ 可以控制所在的 ambient 进入与其并行且名字为 m 的 ambient 的内部. AC 的结构同余和归约的完整规则可参见文献 [4,5], 本文不再赘述. 下面通过一个对移动代理身份认证 (mobile agent authentication) 过程建模的案例[5]来说明 AC 的表达能力.

$Home[(\nu n)(open\ n.\mathbf{0} \mid Agent[out\ home.in\ home.n[out\ Agent.open\ Agent.P]])]$

$\equiv (\nu n)Home[open\ n.\mathbf{0} \mid Agent[out\ home.in\ home.n[out\ Agent.open\ Agent.P]]]$

$\rightarrow (\nu n)(Home[open\ n.\mathbf{0}] \mid Agent[in\ home.n[out\ Agent.open\ Agent.P]])$

$\rightarrow (\nu n)Home[open\ n.\mathbf{0} \mid Agent[n[out\ Agent.open\ Agent.P]]]$

$\rightarrow (\nu n)Home[open\ n.\mathbf{0} \mid n[open\ Agent.P] \mid Agent[\mathbf{0}]]$

$\rightarrow (\nu n)Home[\mathbf{0} \mid open\ Agent.P \mid Agent[\mathbf{0}]]$

$\rightarrow (\nu n)Home[\mathbf{0} \mid P \mid \mathbf{0}]$

$\equiv Home[P]$

上述案例描述了进程 P 离开名为 $Home$ 的 ambient 并且重新返回的过程. 进程 P 被包装在名为 $Agent$ 的 ambient 内, 当 $Agent$ 返回 $Home$ 时需要通过一个只在 $Home$ 内部被知晓的名字 n 进行验证, 验证通过后才会释放出进程 P.

1.2 AC 的相关扩展

自 AC 提出以来, 不少学者对其展开了研究, 其中针对 AC 进行语法语义扩展是一个重要的研究方向. 下面介绍 AC 的两种扩展演算——PAC 和 CMC.

1.2.1 Push and pull ambient calculus (PAC)

PAC[8]是由 Phillips 等人提出的一种基于 AC 的扩展演算. PAC 在保留 AC 大部分语法语义不变的同时, 将进入和离开 ambient 的动作替换为推送和拉取 ambient 的动作. PAC 的语法如下:

$P, Q ::= (\nu n)P \mid \mathbf{0} \mid P \mid Q \mid !P \mid n[P] \mid M.P \mid (n).P \mid \langle n \rangle$

$M ::= push_m\ n \mid pull_m\ n \mid open\ n$

其中, $push_m\ n$ 表示将名字为 n 的 ambient 从名字为 m 的 ambient 内推送出去动作, $pull_m\ n$ 表示将名字为 n 的 ambient 拉取进入名字为 m 的 ambient 的动作. 下面是 PAC 对移动代理身份认证过程的建模, 可以看到在该案例中 PAC 比 AC 更加简洁直观.

$Home[(\nu n)(push_{home}\ n.pull_{home}\ n.open\ n.\mathbf{0} \mid n[P])]$

$\equiv (\nu n)Home[push_{home}\ n.pull_{home}\ n.open\ n.\mathbf{0} \mid n[P]]$

$\rightarrow (\nu n)(Home[pull_{home}\ n.open\ n.\mathbf{0}] \mid n[P])$

$\rightarrow (\nu n)Home[open\ n.\mathbf{0} \mid n[P]]$

$\rightarrow (\nu n)Home[\mathbf{0} \mid P]$

$\equiv Home[P]$

1.2.2 Calculus of mobility and communication (CMC)

CMC[9]是由 Gul 提出的一种基于 AC 的扩展演算. CMC 引入了通过信道进行全局通信的机制, 使得信息的传递不再局限于并行的进程之间. CMC 的部分语法如下:

$P, Q ::= (\nu n_A)P \mid \mathbf{0} \mid P \mid Q \mid n_A[P] \mid M.P \mid a(x).P \mid \overline{a}(M).P$

$M, N ::= x \mid in\ n_A \mid out\ n_A \mid \varepsilon \mid M.N$

其中, x 表示变量. $n_A[P]$ 表示名字为 n 的 ambient 并且允许名字集合 A 中的所有信道跨越 ambient 的边界进行全局通信, $a(x).P$ 表示在名字为 a 的信道上输入能力 x 然后继续进程 P, $\overline{a}(M).P$ 表示在名字为 a 的信道上输出能力 M 然后继续进程 P. 除此之外, CMC 还有其他一些原语, 例如进程的分支执行 $P+Q$, 由于其他原语与本文工作无关, 因此不再赘述.

1.3 Ambient logic (AL)

AL[4]是一种描述 AC 模型时空性质的模态逻辑, 其语法如下:

$$\mathcal{A},\mathcal{B} ::= \mathbf{T} \mid \neg\mathcal{A} \mid \mathcal{A} \vee \mathcal{B} \mid 0 \mid (\mathcal{A} \mid \mathcal{B}) \mid \mathcal{A} \triangleright \mathcal{B} \mid \eta[\mathcal{A}] \mid \mathcal{A}@\eta \mid \eta \circledR \mathcal{A} \mid \mathcal{A} \otimes \eta \mid \diamond\mathcal{A} \mid \nabla\mathcal{A} \mid \forall x.\mathcal{A}$$

$$\eta ::= n \mid x$$

其中, η 表示名字或变量; $\mathcal{A},\mathcal{B}$ 表示公式, 其语义稍后给出.

令 $fn(\mathcal{A})$ 和 $fv(\mathcal{A})$ 分别为公式 $\mathcal{A}$ 中自由出现的名字和变量集合, 其定义如下:

$fn(n) \triangleq \{n\}$	$fv(n) \triangleq \varnothing$
$fn(x) \triangleq \varnothing$	$fv(x) \triangleq \{x\}$
$fn(\mathbf{T}) \triangleq \varnothing$	$fv(\mathbf{T}) \triangleq \varnothing$
$fn(\neg\mathcal{A}) \triangleq fn(\mathcal{A})$	$fv(\neg\mathcal{A}) \triangleq fv(\mathcal{A})$
$fn(\mathcal{A} \vee \mathcal{B}) \triangleq fn(\mathcal{A}) \cup fn(\mathcal{B})$	$fv(\mathcal{A} \vee \mathcal{B}) \triangleq fv(\mathcal{A}) \cup fv(\mathcal{B})$
$fn(\mathbf{0}) \triangleq \varnothing$	$fv(\mathbf{0}) \triangleq \varnothing$
$fn(\mathcal{A}\mid\mathcal{B}) \triangleq fn(\mathcal{A}) \cup fn(\mathcal{B})$	$fv(\mathcal{A}\mid\mathcal{B}) \triangleq fv(\mathcal{A}) \cup fv(\mathcal{B})$
$fn(\mathcal{A} \triangleright \mathcal{B}) \triangleq fn(\mathcal{A}) \cup fn(\mathcal{B})$	$fv(\mathcal{A} \triangleright \mathcal{B}) \triangleq fv(\mathcal{A}) \cup fv(\mathcal{B})$
$fn(\eta[\mathcal{A}]) \triangleq fn(\eta) \cup fn(\mathcal{A})$	$fv(\eta[\mathcal{A}]) \triangleq fv(\eta) \cup fv(\mathcal{A})$
$fn(\mathcal{A}@\eta) \triangleq fn(\eta) \cup fn(\mathcal{A})$	$fv(\mathcal{A}@\eta) \triangleq fv(\eta) \cup fv(\mathcal{A})$
$fn(\eta \circledR \mathcal{A}) \triangleq fn(\eta) \cup fn(\mathcal{A})$	$fv(\eta \circledR A) \triangleq fv(\eta) \cup fv(\mathcal{A})$
$fn(\mathcal{A} \otimes \eta) \triangleq fn(\eta) \cup fn(\mathcal{A})$	$fv(\mathcal{A} \otimes \eta) \triangleq fv(\eta) \cup fv(\mathcal{A})$
$fn(\diamond\mathcal{A}) \triangleq fn(\mathcal{A})$	$fv(\diamond\mathcal{A}) \triangleq fv(\mathcal{A})$
$fn(\nabla\mathcal{A}) \triangleq fn(\mathcal{A})$	$fv(\nabla\mathcal{A}) \triangleq fv(\mathcal{A})$
$fn(\forall x.\mathcal{A}) \triangleq fn(\mathcal{A})$	$fv(\forall x.\mathcal{A}) \triangleq fv(\mathcal{A}) - \{x\}$

我们称公式 $\mathcal{A}$ 是封闭的, 当且仅当 $fv(\mathcal{A}) = \varnothing$. $P \vDash \mathcal{A}$ 表示进程 P 满足封闭的公式 $\mathcal{A}$ (此后指的公式均是封闭的公式). 令 Π 为进程的无限可数集, Λ 为名字的无限可数集, Φ 为公式的无限可数集. 用 $\mathcal{A}\{x \leftarrow n\}$ 表示将公式 $\mathcal{A}$ 中自由出现的变量 x 替换为名字 n. 用 $P \downarrow Q$ 表示进程 P 单层嵌套进程 Q, 具体地, $P \downarrow Q$ 当且仅当 $\exists n \in \Lambda, \exists R \in \Pi$, $P \equiv n[Q] \mid R$. 用 $\rightarrow^*$ 和 $\downarrow^*$ 分别表示 $\rightarrow$ 和 $\downarrow$ 的自反传递闭包. AL 的语义如下:

$$\begin{aligned}
&P \vDash \mathbf{T} \\
&P \vDash \neg\mathcal{A} \quad \triangleq \quad \neg P \vDash \neg\mathcal{A} \\
&P \vDash \mathcal{A} \vee \mathcal{B} \quad \triangleq \quad P \vDash \mathcal{A} \vee P \vDash \mathcal{B} \\
&P \vDash \mathbf{0} \quad \triangleq \quad P \equiv \mathbf{0} \\
&P \vDash \mathcal{A} \mid \mathcal{B} \quad \triangleq \quad \exists Q,R \in \Pi, P \equiv Q \mid R \wedge Q \vDash \mathcal{A} \wedge R \vDash \mathcal{B}
\end{aligned}$$

$$
\begin{aligned}
P \vDash \mathcal{A} \triangleright \mathcal{B} &\triangleq \forall Q \in \Pi, Q \vDash \mathcal{A} \Rightarrow P \mid Q \vDash \mathcal{B} \\
P \vDash n[\mathcal{A}] &\triangleq \exists Q \in \Pi, P \equiv n[Q] \wedge Q \vDash \mathcal{A} \\
P \vDash \mathcal{A}@n &\triangleq n[P] \vDash \mathcal{A} \\
P \vDash n\textcircled{R}\mathcal{A} &\triangleq \exists Q \in \Pi, P \equiv (\nu n)Q \wedge Q \vDash \mathcal{A} \\
P \vDash \mathcal{A} \otimes n &\triangleq (\nu n)P \vDash \mathcal{A} \\
P \vDash \diamond \mathcal{A} &\triangleq \exists Q \in \Pi, P \to * Q \wedge Q \vDash \mathcal{A} \\
P \vDash \nabla \mathcal{A} &\triangleq \exists Q \in \Pi, P \downarrow * Q \wedge Q \vDash \mathcal{A} \\
P \vDash \forall x.\mathcal{A} &\triangleq \forall n \in \Lambda, P \vDash \mathcal{A}\{x \leftarrow n\}
\end{aligned}
$$

任意进程均满足公式 **T**. 进程 P 满足公式 $\neg\mathcal{A}$ 当且仅当进程 P 不满足公式 $\mathcal{A}$. 进程 P 满足公式 $\mathcal{A} \vee \mathcal{B}$ 当且仅当进程 P 满足公式 $\mathcal{A}$ 或 $\mathcal{B}$. 进程 P 满足公式 **0** 当且仅当进程 P 与空进程结构同余. 进程 P 满足公式 $\mathcal{A} \mid \mathcal{B}$ 当且仅当进程 P 与进程 $Q \mid R$ 结构同余, 且进程 Q 和 R 分别满足公式 $\mathcal{A}$ 和 $\mathcal{B}$. 进程 P 满足公式 $\mathcal{A} \triangleright \mathcal{B}$ 当且仅当对任意满足公式 $\mathcal{A}$ 的进程 Q 都有进程 $P \mid Q$ 满足公式 $\mathcal{B}$. 进程 P 满足公式 $n[\mathcal{A}]$ 当且仅当进程 P 与进程 $n[Q]$ 结构同余, 且进程 Q 满足公式 $\mathcal{A}$. 进程 P 满足公式 $\mathcal{A}@n$ 当且仅当进程 $n[P]$ 满足公式 $\mathcal{A}$. 进程 P 满足公式 $n\textcircled{R}\mathcal{A}$ 当且仅当进程 P 与进程 $(\nu n)Q$ 结构同余, 且进程 Q 满足公式 $\mathcal{A}$. 进程 P 满足公式 $\mathcal{A} \otimes n$ 当且仅当进程 $(\nu n)P$ 满足公式 $\mathcal{A}$. 进程 P 满足公式 $\diamond\mathcal{A}$ 当且仅当进程 P 若干步归约后得到的某个进程 Q 满足公式 $\mathcal{A}$. 进程 P 满足公式 $\nabla\mathcal{A}$ 当且仅当进程 P 若干层嵌套的某个进程 Q 满足公式 $\mathcal{A}$. 进程 P 满足公式 $\forall x.\mathcal{A}$ 当且仅当对任意名字 n, 进程 P 满足公式 $\mathcal{A}\{x \leftarrow n\}$. 除此之外, AL 还允许使用一些派生的算子: $\mathcal{A} \wedge \mathcal{B}$ 表示 $\neg(\neg A \vee \neg\mathcal{B})$, $\Box\mathcal{A}$ 表示 $\neg\diamond\neg\mathcal{A}$, $\Delta\mathcal{A}$ 表示 $\neg\nabla\neg A$, $\exists x.\mathcal{A}$ 表示 $\neg\forall x.\neg\mathcal{A}$, 等.

文献 [11] 研究了 AC 对 AL 模型检测问题的可判定性, 证明了 AC 中的 $!P$ 原语以及 AL 中的 $\triangleright$ 算子将导致模型检测问题的不可判定, 且证明了不含 $(\nu n)P$ 和 $!P$ 原语的 AC 对不含 ®、$\otimes$ 和 $\triangleright$ 算子的 AL 的模型检测是可判定的, 因此在本文工作中使用的 AL 将不包含 ®、$\otimes$ 和 $\triangleright$ 算子.

2 全局通信移动环境演算 ACGC (ambient calculus with global communication)

为了从原子操作的层面上刻画设备和用户的被动移动及其跨空间通信, 我们参考文献 [8,9] 的工作, 将推拉动作和全局通信的机制引入 AC, 提出全局通信移动环境演算 ACGC. 本节首先介绍 ACGC 的语法和语义, 然后给出一个 ACGC 对 AL 的模型检测算法, 最后证明我们的算法的正确性.

2.1 ACGC 的语法

定义 1 (**ACGC 的语法**). 我们的 ACGC 在保留 AC 大部分语法的同时, 省略了给模型检测问题带来困难的进程原语 $(\nu n)P$ 和 $!P$, 融合了 PAC 和 CMC 的部分思想, 引入拉取和推送 ambient 的能力原语 *pull M* 和 *push M*, 并将 AC 原有的通信原语 $(x).P$ 和 $\langle M\rangle$ 替换为在信道 M 上的全局通信原语 $M(n).P$ 和 $M\langle N\rangle.P$. 得到的 ACGC 的语法如下:

$P, Q ::= \mathbf{0} \mid P \mid Q \mid M[P] \mid M.P \mid M(n).P \mid M\langle N\rangle.P$

$M, N ::= n \mid in\ M \mid out\ M \mid open\ M \mid push\ M \mid pull\ M \mid \varepsilon \mid M.N$

其中, *push M* 表示推送名字为 M 的 ambient 的动作, *pull M* 表示拉取名字为 M 的 ambient 的动作. $M(n).P$ 表示在名字为 M 的信道上输入能力 n 然后继续进程 P, $M\langle N\rangle.P$ 表示在名字为 M 的信道上输出能力 N 然后继续进程 P. 其余各原语的含义与 AC 相同.

2.2 ACGC 的语义

定义 2 (**ACGC 的自由名字**). 令 $fn(P)$ 为进程 P 中自由出现的名字集合, 其定义如下:

$$
\begin{aligned}
&fn(n) \triangleq \{n\} && fn(in\ M) \triangleq fn(M) \\
&fn(out\ M) \triangleq fn(M) && fn(open\ M) \triangleq fn(M) \\
&fn(push\ M) \triangleq fn(M) && fn(pull\ M) \triangleq fn(M) \\
&fn(\varepsilon) \triangleq \varnothing && fn(M.N) \triangleq fn(M) \cup fn(N)
\end{aligned}
$$

$$fn(P \mid Q) \triangleq fn(P) \cup fn(Q) \qquad fn(M[P]) \triangleq fn(M) \cup fn(P)$$
$$fn(M.P) \triangleq fn(M) \cup fn(P) \qquad fn(M(n).P) \triangleq fn(M) \cup fn(P) - \{n\}$$
$$fn(M\langle N\rangle.P) \triangleq fn(M) \cup fn(P) \cup fn(N)$$

定义 3 (ACGC 的结构同余). ACGC 的结构同余规则如下:

$P \equiv P$ (Struct Refl)
$P \equiv Q \Rightarrow Q \equiv P$ (Struct Symm)
$P \equiv Q, Q \equiv R \Rightarrow P \equiv R$ (Struct Trans)
$P \equiv Q \Rightarrow P \mid R \equiv Q \mid R$ (Struct Par)
$P \equiv Q \Rightarrow M[P] \equiv M[Q]$ (Struct Amb)
$P \equiv Q \Rightarrow M.P \equiv M.Q$ (Struct Action)
$P \equiv Q \Rightarrow M(n).P \equiv M(n).Q$ (Struct Glob Input)
$P \equiv Q \Rightarrow M\langle N\rangle.P \equiv M\langle N\rangle.Q$ (Struct Glob Output)
$\varepsilon.P \equiv P$ (Struct ε)
$(M.N).P \equiv M.N.P$ (Struct .)
$P \mid \mathbf{0} \equiv P$ (Struct Zero Par)
$P \mid Q \equiv Q \mid P$ (Struct Par Comm)
$(P\ Q) \mid R \equiv P \mid (Q \mid R)$ (Struct Par Assoc)

定义 4 (ACGC 的标号迁移). 为了描述全局通信原语 $M(n).P$ 和 $M\langle N\rangle.P$ 的语义, 我们需要引入标号迁移. 用 $P \xrightarrow{\alpha} Q$ 表示进程 P 通过标号为 α 的迁移转换为进程 Q. 用 $P\{n \leftarrow N\}$ 表示将进程 P 中自由出现的名字 n 替换为能力 N. ACGC 的标号迁移规则如下:

$\alpha ::= m(N) \mid m\langle N\rangle$

$m(n).P \xrightarrow[m\langle N\rangle]{m(N)} P\{n \leftarrow N\}$ (LTS Glob Input)
$m\langle N\rangle.P \xrightarrow{} P$ (LTS Glob Output)
$P \xrightarrow{\alpha} Q \Rightarrow P \mid R \xrightarrow{\alpha} Q \mid R$ (LTS Par)
$P \xrightarrow{\alpha} Q \Rightarrow n[P] \xrightarrow{\alpha} n[Q]$ (LTS Amb)

定义 5 (ACGC 的归约). ACGC 的归约规则如下:

$n[in\ m.P \mid Q] \mid m[R] \rightarrow m[n[P \mid Q] \mid R]$ (Red In)
$m[n[out\ m.P \mid Q] \mid R] \rightarrow n[P \mid Q] \mid m[R]$ (Red Out)
$open\ n.P \mid n[Q] \rightarrow P \mid Q$ (Red Open)
$m[push\ n.P \mid n[Q] \mid R] \rightarrow m[P \mid R] \mid n[Q]$ (Red Push)
$m[pull\ n.P \mid Q] \mid n[R] \rightarrow m[P \mid Q \mid n[R]]$ (Red Pull)
$P \xrightarrow{m(N)} P', Q \xrightarrow{m\langle N\rangle} Q' \Rightarrow P \mid Q \rightarrow P' \mid Q'$ (Red Glob Comm)
$P \rightarrow Q \Rightarrow P \mid R \rightarrow Q \mid R$ (Red Par)
$P \rightarrow Q \Rightarrow n[P] \rightarrow n[Q]$ (Red Amb)
$P \equiv P', P \rightarrow Q, Q \equiv Q' \Rightarrow P' \rightarrow Q'$ (Red ≡)

下面通过一个使用扫地机器人的家居场景案例来说明 ACGC 的表达能力. 在一个由客厅 *LivingRoom*、卧室 *Bedroom*、浴室 *Bathroom*、厨房 *Kitchen* 组成的居室中, 位于 *Bedroom* 的住户 *Bob* 想要命令 *LivingRoom* 里的扫地机器人 *SwpRob* 打扫 *Bathroom*, 使用 ACGC 构建的模型如下:

$$LivingRoom[SwpRob[c(R).out\ LivingRoom.in\ R.\mathbf{0}]] \mid Bedroom[Bob[c\langle Bathroom\rangle.\mathbf{0}]] \mid Bathroom[\mathbf{0}] \mid Kitchen[\mathbf{0}]$$
$$\rightarrow LivingRoom[SwpRob[out\ LivingRoom.in\ Bathroom.\mathbf{0}]] \mid Bedroom[Bob[\mathbf{0}]] \mid Bathroom[\mathbf{0}] \mid Kitchen[\mathbf{0}]$$

$\rightarrow LivingRoom[\mathbf{0}] \mid SwpRob[in\ Bathroom.\mathbf{0}] \mid Bedroom[Bob[\mathbf{0}]] \mid Bathroom[\mathbf{0}] \mid Kitchen[\mathbf{0}]$

$\rightarrow LivingRoom[\mathbf{0}] \mid Bedroom[Bob[\mathbf{0}]] \mid Bathroom[SwpRob[\mathbf{0}]] \mid Kitchen[\mathbf{0}]$

在该案例中, *Bob* 与 *SwpRob* 在事先约定好的信道 *c* 上进行通信. *Bob* 将目的地 *Bathroom* 通过信道 *c* 发送给 *SwpRob*, *SwpRob* 从信道 *c* 接收到目的地信息后即离开 *LivingRoom*, 并前往目的地 *R* (通信后被替换为 *Bathroom*).

2.3 模型检测算法

文献 [4] 已经给出了一个 AC 对 AL 的模型检测算法, 我们在此基础上增加了对 *push M.P*、*pull M.P*、$M(n).P$ 和 $M\langle N\rangle.P$ 进程的正规式计算规则, 并设计了计算可达进程集 (*Reachable*(*P*)) 和子位置进程集 (*SubLocations*(*P*)) 的算法, 使其适应 ACGC 对 AL 的模型检测.

定义 6 (主进程和进程的正规式). 素进程是并非形如 **0**、$P \mid Q$、$\varepsilon.P$ 和 $(M.N).P$ 的进程. 用 π 表示素进程.

$\pi ::= M[P] \mid n.P \mid in\ M.P \mid out\ M.P \mid open\ M.P \mid push\ M.P \mid pull\ M.P \mid M(n).P \mid M\langle N\rangle.P$

假设有一些带下标的进程 $P_1,\ldots,P_k$, 而 S 为这些下标 $1,\ldots,k$ 组成的集合, 我们将这些进程外加一个空进程的并行记为 $\Pi_{i\in S}P_i$:

$\Pi_{i\in S}P_i \triangleq P_1 \mid \ldots \mid P_k \mid \mathbf{0}$ where $S=\{1,\ldots,k\}$

令 *Norm*(*P*) 为进程 *P* 的正规式, 其定义如下:

$Norm(\mathbf{0}) \triangleq []$

$Norm(P \mid Q) \triangleq [\pi_1,\ldots,\pi_k,\pi'_1,\ldots,\pi'_{k'}]$ where $Norm(P)=[\pi_1,\ldots,\pi_k]$ and $Norm(Q)=[\pi'_1,\ldots,\pi'_{k'}]$

$Norm(M[P]) \triangleq [M[P]]$

$Norm(M.P) \triangleq [M.P]$ if $M \in \{n, in\ N, out\ N, open\ N, push\ N, pull\ N\}$

$Norm(\varepsilon.P) \triangleq Norm(P)$

$Norm((M.N).P) \triangleq Norm(M.(N.P))$

$Norm(M(n).P) \triangleq [M(n).P]$

$Norm(M\langle N\rangle.P) \triangleq [M\langle N\rangle.P]$

在判断 $P \vDash \Diamond\mathcal{A}$ 和 $P \vDash \nabla\mathcal{A}$ 这两类问题时, 分别需要求出进程 *P* 多步归约和多层嵌套的进程. 与文献 [4] 的思路类似, 我们直接定义满足以下性质的进程集合是进程 *P* 的可达进程集 (*Reachable*(*P*)) 和子位置进程集 (*SubLocations*(*P*)), 然后给出构造这两种集合的算法.

引理 1. 对任意进程 *P*, 如果进程 $Q \in Reachable(P)$ 则 $P \rightarrow^* Q$, 并且对于任意满足 $P \rightarrow^* R$ 的进程 *R*, 存在进程 $R' \in Reachable(P)$ 使得 $R' \equiv R$.

引理 2. 对任意进程 *P*, 如果进程 $Q \in SubLocations(P)$ 则 $P \downarrow^* Q$, 并且对于任意满足 $P \downarrow^* R$ 的进程 *R*, 存在进程 $R' \in SubLocations(P)$ 使得 $R' \equiv R$.

下面分别给出求 *Reachable*(*P*) 和 *SubLocations*(*P*) 的算法 *MakeReachable*(*P*) (见算法 1). *MakeSubLocations*(*P*) (见算法 2).

算法 1. *MakeReachable*(*P*).

输入: An ACGC process *P*;

输出: An array of ACGC processes *RP*, which represents *Reachable*(*P*).

1. $RP := [P]$, $i := 1$ //初始化 *RP*, 其中只包含 *P*
2. **while** $i \leqslant length(RP)$ **do** //遍历 *RP* 中的所有进程 $RP[i]$
3. **if** $\exists Q \in \Pi, \forall R \in RP$, $RP[i] \rightarrow Q \wedge R \neq Q$ **then** //如果 $RP[i]$ 可以一步归约出某个进程 *Q*, 并且 *Q* 不与 *RP* 中的任意进程结构同余
4. push *Q* in the back of *RP* //将 *Q* 添加至 *RP* 末尾
5. **else**

6. $i := i+1$ //否则遍历至下一个进程 $RP[i+1]$
7. **return** RP //返回 RP 作为 P 的可达进程集

算法 2. *MakeSubLocations*(P).

输入: An ACGC process P;
输出: An array of ACGC processes SP, which represents *SubLocations*(P).

1. $SP := [P]$, $i := 1$ //初始化 SP, 其中只包含 P
2. **while** $i \leqslant length(SP)$ **do** //遍历 SP 中的所有进程 $SP[i]$
3. **if** $\exists Q \in \Pi, \forall R \in SP$, $SP[i]\downarrow Q \wedge R \neq Q$ **then** //如果 $SP[i]$ 单层嵌套某个进程 Q, 并且 Q 不与 SP 中的任意进程结构同余
4. push Q in the back of SP //将 Q 添加至 SP 末尾
5. **else**
6. $i := i+1$ //否则遍历至下一个进程 $SP[i+1]$
7. **return** SP //返回 SP 作为 P 的子位置进程集

Check($P, \mathcal{A}$) 的算法如算法 3.

算法 3. *Check*(P, $\mathcal{A}$).

输入: An ACGC process P and an AL formula $\mathcal{A}$;
输出: true($P \vDash \mathcal{A}$) or false($\neg P \vDash \mathcal{A}$).

1. **if** $\mathcal{A}$=**T then**
2. **return** true
3. **if** $\mathcal{A}=\neg \mathcal{B}$ **then**
4. **return not** *Check*(P, $\mathcal{B}$)
5. **if** $\mathcal{A}=\mathcal{B} \vee C$ **then**
6. **return** *Check*(P, $\mathcal{B}$) **or** *Check*(P, C)
7. **if** $\mathcal{A}$=**0 then**
8. **return** *Norm*(P)=[]
9. **if** $\mathcal{A}=\mathcal{B} \mid C$ **then**
10. let *Norm*(P)= $[\pi_1,\ldots,\pi_k]$, $S=\{1,\ldots,k\}$
11. **for each** $S' \in 2^S$, $S''=S-S'$ **do**
12. **if** *Check*($\Pi_{i\in S'}\pi_i, \mathcal{B}$) **and** *Check*($\Pi_{i\in S''}\pi_i, C$) **then**
13. **return** true
14. **return** false
15. **if** $\mathcal{A}=n[\mathcal{B}]$ **then**
16. **return** $\exists Q \in \Pi$, *Norm*(P)=[n[Q]] **and** *Check*($Q, \mathcal{B}$)
17. **if** $\mathcal{A}=\mathcal{B}@n$ **then**
18. **return** *Check*(n[P], $\mathcal{B}$)
19. **if** $\mathcal{A}=\diamond\mathcal{B}$ **then**
20. **for each** $Q \in$ *Reachable*(P) **do**
21. **if** *Check*($Q, \mathcal{B}$) **then**

```
22.                 return true
23.           return false
24. if 𝒜 = ℬ then
25.       for each Q ∈ SubLocations(P) do
26.             if Check(Q,ℬ) then
27.                 return true
28.       return false
29. if 𝒜 = ∀x.ℬ then
30.       let n ∈ Λ − fn(P) − fn(ℬ), Λ′ = fn(P) ∪ fn(ℬ) ∪ {n}
31.       for each m ∈ Λ′ do
32.             if not Check(P, ℬ{x ← m}) then
33.                 return false
34.           return true
```

2.4 模型检测问题的可判定性

我们通过证明算法 $Check(P,\mathcal{A})$ 的正确性来证明 ACGC 对 AL 模型检测问题的可判定性. 具体地, 通过引理 1–引理 9 引出引理 11 (算法 $Check(P,\mathcal{A})$ 的可靠性) 和引理 12 (算法 $Check(P,\mathcal{A})$ 的完备性), 结合引理 10 (算法 $Check(P,\mathcal{A})$ 的终止性) 可证明该算法的正确性, 进而证明 ACGC 对 AL 模型检测问题的可判定性.

引理 3. 对任意进程 P, $Norm(P)=[]$ 当且仅当 $P\equiv\mathbf{0}$.

我们分如下情况进行证明.

(1) $Norm(P)=[]\Rightarrow P\equiv\mathbf{0}$

根据正规式的构造规则, $Norm(P)=[]$ 当且仅当下列 4 种情况之一: (a) $P=\mathbf{0}$; (b) $P=Q\mid R$ 且 $Norm(Q)=[]$ 且 $Norm(R)=[]$; (c) $P=\varepsilon.Q$ 且 $Norm(Q)=[]$; (d) $P=(\varepsilon.M).Q$ 且 $Norm(M.Q)=[]$. 根据结构同余性质, 上述 4 种情况均有 $P\equiv\mathbf{0}$.

(2) $P\equiv\mathbf{0}\Rightarrow Norm(P)=[]$

根据结构同余性质可知, $P\equiv\mathbf{0}$ 当且仅当下列 4 种情况之一: (a) $P=\mathbf{0}$; (b) $P=Q\mid R$ 且 $Q\equiv\mathbf{0}$ 且 $R\equiv\mathbf{0}$; (c) $P=\varepsilon.Q$ 且 $Q\equiv\mathbf{0}$; (d) $P=(\varepsilon.M).Q\wedge M.Q\equiv\mathbf{0}$. 根据正规式的构造规则, 上述 4 种情况均有 $Norm(P)=[]$.

引理 4. 对任意进程 P, 如果 $Norm(P)=[\pi_1,\ldots,\pi_k]$ 且 $S=\{1,\ldots,k\}$, 那么 $P\equiv\Pi_{i\in S}\pi_i$.

引理 5. 对任意进程 P, Q, R, 如果 $Norm(P)=[\pi_1,\ldots,\pi_k]$ 且 $S=\{1,\ldots,k\}$ 且 $P\equiv Q\mid R$, 那么存在集合 $S'\in 2^S$, $S''=S-S'$, 使得 $\Pi_{i\in S'}\pi_i\equiv Q$ 且 $\Pi_{i\in S''}\pi_i\equiv R$.

引理 6. 对任意进程 P, Q, 如果 $Norm(P)=[M[Q]]$, 那么 $P\equiv M[Q]$.

引理 7. 对任意进程 P, Q, 如果 $P\equiv M[Q]$, 那么存在进程 R, 使得 $Norm(P)=[M[R]]$ 且 $R\equiv Q$.

引理 4–引理 7 可以通过正规式的构造规则出发进行证明. 由于篇幅所限, 其证明从略.

引理 8. 对任意进程 P 和公式 $\mathcal{A}$, 如果名字 $m\notin fn(P)\cup fn(\mathcal{A})$, 那么 $P\vDash\mathcal{A}\Rightarrow P\{n\leftarrow m\}\vDash\mathcal{A}\{n\leftarrow m\}$.

证明从略.

引理 9. 对任意进程 P 和公式 $\mathcal{A}$, 如果名字 $m\notin fn(P)\cup fn(\mathcal{A})$ 且名字 $n\notin fn(P)$, 那么 $P\vDash\mathcal{A}\Rightarrow P\vDash\mathcal{A}\{n\leftarrow m\}$.

我们分如下情况进行证明.

(1) $n\notin fn(\mathcal{A})$

显然有 $\mathcal{A}\{n\leftarrow m\}=\mathcal{A}$, 因此 $P\vDash\mathcal{A}\Rightarrow P\vDash\mathcal{A}\{n\leftarrow m\}$.

(2) $n\in fn(\mathcal{A})$

根据引理 8, $P\vDash\mathcal{A}\Rightarrow P\{n\leftarrow m\}\vDash\mathcal{A}\{n\leftarrow m\}$. 因为 $n\notin fn(P)$, 所以 $P\{n\leftarrow m\}=P$, 所以 $P\vDash\mathcal{A}\Rightarrow P\vDash\mathcal{A}\{n\leftarrow m\}$.

该引理说明, 如果 $P \vDash \mathcal{A}$, 那么将公式 $\mathcal{A}$ 中自由出现且不在进程 P 自由出现的某个名字 n 替换为一个没有在进程 P 和公式 $\mathcal{A}$ 自由出现名字 m, 会有 $P \vDash \mathcal{A}\{n \leftarrow m\}$ 的性质. 因此在检测. $P \vDash \forall x.\mathcal{A}$ 这类问题时, 不需要将无限名字集 Λ 中的所有名字一一代入公式 $\mathcal{A}$, 而是只需要将进程 P 和公式 $\mathcal{A}$ 中自由出现的所有名字, 以及任意一个没有在进程 P 和公式 $\mathcal{A}$ 中自由出现的名字一一带入公式 $\mathcal{A}$ 进行检测即可.

引理 10. 对任意进程 P, Q 和公式 $\mathcal{A}$, 如果 $P \vDash \mathcal{A}$ 且 $P \equiv Q$, 那么 $Q \vDash \mathcal{A}$.

结构同余代表进程在语义上的等价, 因此结构同余的一对进程对于同一个公式具有相同的满足性.

引理 11 (算法 $Check(P, \mathcal{A})$ 的终止性). 算法 $Check(P, \mathcal{A})$ 能终止.

证明: 因为 $Check(P, \mathcal{A})$ 总是递归调用 $\mathcal{A}$ 的子公式的检测算法, 而 $Check(P, \mathbf{T})$ 和 $Check(P, \mathbf{0})$ 显然能终止, 因此 $Check(P, \mathcal{A})$ 能终止.

引理 12 (算法 $Check(P, \mathcal{A})$ 的可靠性). 对任意进程 P 和公式 $\mathcal{A}$, 如果 $Check(P, \mathcal{A}) = \text{true}$ 那么 $P \vDash \mathcal{A}$.

我们使用归纳法, 分如下情况进行证明.

(1) $\mathcal{A}$ 是原子公式

$\mathcal{A} = \mathbf{T}$ 时, 显然对任意进程 P 都有 $P \vDash \mathbf{T}$.

$\mathcal{A} = \mathbf{0}$ 时, 由 $Check(P, \mathbf{0}) = \text{true}$ 可得 $Norm(P) = []$. 根据引理 3, 有 $P \equiv \mathbf{0}$, 则 $P \vDash \mathbf{0}$.

(2) $\mathcal{A}$ 不是原子公式时, 因为 $\mathcal{A}$ 是通过递归构造而成, 不妨归纳假设对其子公式的检测满足可靠性

$\mathcal{A} = \neg\mathcal{B}$ 时, 由 $Check(P, \neg\mathcal{B}) = \text{true}$ 可得 $\neg Check(P, \mathcal{B}) = \text{true}$. 引理 13 将独立地证明 $\mathcal{B}$ 为原子公式时 $Check(P, \mathcal{B})$ 满足完备性, 因此不妨假设 $Check(P, \mathcal{B})$ 满足完备性, 即 $\neg P \vDash \mathcal{B}$, 则 $P \vDash \neg\mathcal{B}$.

$\mathcal{A} = \mathcal{B} \vee \mathcal{C}$ 时, 由 $Check(P, \mathcal{B} \vee \mathcal{C}) = \text{true}$ 可得 $Check(P, \mathcal{B}) = \text{true}$ 或 $Check(P, \mathcal{C}) = \text{true}$. 由归纳假设有 $P \vDash \mathcal{B}$ 或 $P \vDash \mathcal{C}$, 则 $P \vDash \mathcal{B} \vee \mathcal{C}$.

$\mathcal{A} = \mathcal{B} \mid \mathcal{C}$ 时, 令 $Norm(P) = [\pi_1, \ldots, \pi_k]$ 且 $S = \{1, \ldots, k\}$, 根据引理 4, 有 $P \equiv \Pi_{i \in S}\pi_i$. 由 $Check(P, \mathcal{B} \mid \mathcal{C}) = \text{true}$ 可得存在集合 $S' \in 2^S$, $S'' = S - S'$, 使 $Check(\Pi_{i \in S'}\pi_i, \mathcal{B}) = \text{true}$ 且 $Check(\Pi_{i \in S''}\pi_i, \mathcal{C}) = \text{true}$. 由归纳假设有 $\Pi_{i \in S'}\pi_i \vDash \mathcal{B}$ 且 $\Pi_{i \in S''}\pi_i \vDash \mathcal{C}$, 则 $\Pi_{i \in S'}\pi_i \mid \Pi_{i \in S''}\pi_i \vDash \mathcal{B} \mid \mathcal{C}$. 根据结构同余规则 Struct Par 和 Struct Par Comm, 从 $P \equiv \Pi_{i \in S}\pi_i$ 可进一步得到 $P \equiv \Pi_{i \in S'}\pi_i \mid \Pi_{i \in S''}\pi_i$. 再根据引理 10, 有 $P \vDash \mathcal{B} \mid \mathcal{C}$.

$\mathcal{A} = n[\mathcal{B}]$ 时, 由 $Check(P, n[\mathcal{B}]) = \text{true}$ 可得存在进程 Q 使得 $Norm(P) = [n[Q]]$ 且 $Check(Q, \mathcal{B}) = \text{true}$. 根据引理 6, 有 $P \equiv n[Q]$. 由归纳假设有 $Q \vDash \mathcal{B}$, 则 $P \vDash n[\mathcal{B}]$.

$\mathcal{A} = \mathcal{B}@n$ 时, 由 $Check(P, \mathcal{B}@n) = \text{true}$ 可得 $Check(n[P], \mathcal{B}) = \text{true}$. 由归纳假设有 $n[P] \vDash \mathcal{B}$, 则 $P \vDash \mathcal{B}@n$.

$\mathcal{A} = \diamond\mathcal{B}$ 时, 由 $Check(P, \diamond\mathcal{B}) = \text{true}$ 可得存在进程 $Q \in Reachable(P)$ 使得 $Check(Q, \mathcal{B}) = \text{true}$. 根据引理 1, 有 $P \rightarrow^* Q$. 由归纳假设有 $Q \vDash \mathcal{B}$, 则 $P \vDash \diamond\mathcal{B}$.

$\mathcal{A} = \nabla\mathcal{B}$ 时, 由 $Check(P, \nabla\mathcal{B}) = \text{true}$ 可得存在进程 $Q \in SubLocations(P)$ 使得 $Check(Q, \mathcal{B}) = \text{true}$. 根据引理 2, 有 $P \downarrow^* Q$. 由归纳假设有 $Q \vDash \mathcal{B}$, 则 $P \vDash \nabla\mathcal{B}$.

$\mathcal{A} = \forall x.\mathcal{B}$ 时, 任取名字 $n \in \Lambda - fn(P) \cup fn(\mathcal{B})$ 并令 $\Lambda' = fn(P) \cup fn(\mathcal{B}) \cup \{n\}$. 由 $Check(P, \forall x.\mathcal{B}) = \text{true}$ 可得对任意的名字 $m \in \Lambda'$ 都有 $Check(P, \mathcal{B}\{x \leftarrow m\}) = \text{true}$. 由归纳假设, 对任意的名字 $m \in \Lambda'$ 都有 $P \vDash \mathcal{B}\{x \leftarrow m\}$, 根据引理 9, 可得对任意的名字 $m \in \Lambda$ 都有 $P \vDash \mathcal{B}\{x \leftarrow m\}$, 则 $P \vDash \forall x.\mathcal{B}$.

引理 13 (算法 $Check(P, \mathcal{A})$ 的完备性). 对任意进程 P 和公式 $\mathcal{A}$, 如果 $P \vDash \mathcal{A}$ 那么 $Check(P, \mathcal{A}) = \text{true}$.

我们使用归纳法, 分如下情况进行证明.

(1) $\mathcal{A}$ 是原子公式

$\mathcal{A} = \mathbf{T}$ 时, 显然对任意进程 P 都有 $Check(P, \mathbf{T}) = \text{true}$.

$\mathcal{A} = \mathbf{0}$ 时, 由 $P \vDash \mathbf{0}$ 可得 $P \equiv \mathbf{0}$. 根据引理 3, $Norm(P) = []$, 则 $Check(P, \mathbf{0}) = \text{true}$.

(2) $\mathcal{A}$ 不是原子公式, 因为 $\mathcal{A}$ 是通过递归构造而成, 不妨归纳假设对其子公式的检验满足完备性

$\mathcal{A} = \neg\mathcal{B}$时, 由 $P \vDash \neg\mathcal{B}$可得 $\neg P \vDash \mathcal{B}$. 引理 12 证明了 $\mathcal{B}$为原子公式时 $Check(P, \mathcal{B})$满足可靠性, 则有 $\neg Check(P, \mathcal{B}) = \text{true}$, 则 $Check(P, \neg\mathcal{B}) = \text{true}$.

$\mathcal{A} = \mathcal{B} \vee \mathcal{C}$ 时由 $P \vDash \mathcal{B} \vee \mathcal{C}$ 可得 $P \vDash \mathcal{B}$ 或 $P \vDash \mathcal{C}$. 由归纳假设有 $Check(P, \mathcal{B}) = \text{true}$ 或 $Check(P, \mathcal{C}) = \text{true}$, 则

$Check(P,\mathcal{B}\vee C)=\text{true}$.

$\mathcal{A}=\mathcal{B}\mid C$时, 由 $P\vDash\mathcal{B}\mid C$可得存在进程 Q,R, 使得 $P\equiv Q\mid R$且 $Q\vDash\mathcal{B}$且 $R\vDash C$. 令 $Norm(P)=[\pi_1,\ldots,\pi_k]$且 $S=1,\ldots,k$, 根据引理 5, 存在集合 $S'\in 2^S$, $S''=S-S'$, 使得 $\Pi_{i\in S'}\pi_i\equiv Q$且 $\Pi_{i\in S''}\pi_i\equiv R$. 根据引理 10, 有 $\Pi_{i\in S'}\pi_i\vDash\mathcal{B}$且 $\Pi_{i\in S''}\pi_i\vDash C$. 由归纳假设有 $Check(\Pi_{i\in S'}\pi_i,\mathcal{B})=\text{true}$且 $Check(\Pi_{i\in S''}\pi_i,C)=\text{true}$, 则 $Check(P,\mathcal{B}\mid C)=\text{true}$.

$\mathcal{A}=n[\mathcal{B}]$时, 由 $P\vDash n[\mathcal{B}]$可得存在进程 Q使得 $P\equiv n[Q]$且 $Q\vDash\mathcal{B}$. 根据引理 7, 可得存在进程 R使得 $Norm(P)=[n[R]]$且 $R\equiv Q$. 根据引理 10, 有 $R\vDash\mathcal{B}$. 由归纳假设有 $Check(R,\mathcal{B})=\text{true}$, 则 $n[R]\vDash n[\mathcal{B}]$, 则 $Check(P,n[\mathcal{B}])=\text{true}$.

$\mathcal{A}=\mathcal{B}@n$时, 由 $P\vDash\mathcal{B}@n$可得 $n[P]\vDash\mathcal{B}$. 由归纳假设有 $Check(n[P],\mathcal{B})=\text{true}$, 则 $Check(P,\mathcal{B}@n)=\text{true}$.

$\mathcal{A}=\diamond\mathcal{B}$时, 由 $P\vDash\diamond\mathcal{B}$可得存在进程 Q使得 $P\rightarrow^{*}Q$且 $Q\vDash\mathcal{B}$. 根据引理 1, 存在进程 $R\in Reachable(P)$使得 $R\equiv Q$. 根据引理 10, 有 $R\vDash\mathcal{B}$. 由归纳假设有 $Check(R,\mathcal{B})=\text{true}$, 则 $Check(P,\diamond\mathcal{B})=\text{true}$.

$\mathcal{A}=\nabla\mathcal{B}$时, 由 $P\vDash\nabla\mathcal{B}$可得存在进程 Q使得 $P\downarrow^{*}Q$且 $Q\vDash\mathcal{B}$. 根据引理 2, 存在进程 $R\in SubLocations(P)$使得 $R\equiv Q$. 根据引理 10, 有 $R\vDash\mathcal{B}$. 由归纳假设有 $Check(R,\mathcal{B})=\text{true}$, 则 $Check(P,\nabla\mathcal{B})=\text{true}$.

$\mathcal{A}=\forall x.\mathcal{B}$时, 由 $P\vDash\forall x.\mathcal{B}$可得对任意名字 m都有 $P\vDash\mathcal{B}\{x\leftarrow m\}$. 任取名字 $n\in\Lambda-fn(P)\cup fn(\mathcal{B})$并令 $\Lambda'=fn(P)\cup fn(\mathcal{B})\cup\{n\}$. 因为 $\Lambda'\subset\Lambda$, 所有对任意名字 $m\in\Lambda'$都有 $P\vDash\mathcal{B}\{x\leftarrow m\}$. 由归纳假设, 对任意名字 $m\in\Lambda'$都有 $Check(P,\mathcal{B}\{x\leftarrow m\})=\text{true}$, 则 $Check(P,\forall x.\mathcal{B})=\text{true}$.

定理 1 (ACGC 对 AL 的可判定性). 对任意进程 P和公式 $\mathcal{A}$, $P\vDash\mathcal{A}$的问题是可判定的.

我们已经给出了一个检测 $P\vDash\mathcal{A}$问题的算法 $Check(P,\mathcal{A})$并证明了该算法的终止性、可靠性和完备性, 因此算法 $Check(P,\mathcal{A})$是正确的, 并且 $P\vDash\mathcal{A}$的问题可以通过 $Check(P,\mathcal{A})$进行判定.

3 移动通信建模语言 MLMC (modeling language for mobility and communication)

ACGC 是一种较为抽象的进程代数, 用户直接使用较为困难, 因此我们提出 MLMC 以帮助用户对物联网设备的行为进行建模. MLMC 可以较为清晰地描述物联网设备的移动和通信行为, 同时支持其模型到 ACGC 模型的转换. 本节首先介绍 MLMC 的语法, 然后介绍将 MLMC 模型转换为 ACGC 模型的方法.

3.1 MLMC 的语法

定义 7 (MLMC 的语法). MLMC 模型由实体声明、初始位置定义和行动规则定义 3 部分组成. 实体声明的语法如图 1 所示.

```
<entities-declaration> ::= "ent" "{" <identifier-list> ";" "}"
<identifier-list>      ::= <identifier> | <identifier> "," <identifier-list>
```

图 1　实体声明的语法

实体声明 (*entities-declaration*) 由关键字 ent 和需要参与建模的所有实体的标识符列表 (*identifier-list*) 组成. 标识符列表是一些标识符 (*identifier*) 组成的列表. MLMC 的标识符是由字母、数字和下划线组成的非空字符串, 用来命名实体、信道以及等待通信后确定的变量. 规定一个 MLMC 模型中所有实体的名字互不相同.

实体声明的实例如图 2 所示, 该实例描述了一个家居场景中的 7 个实体, 分别为客厅 *LivingRoom*、卧室 *Bedroom*、浴室 *Bathroom*、厨房 *Kitchen*、扫地机器人 *SwpRob*1 和 *SwpRob*2 以及人员 *Bob*.

```
1  ent {
2      LivingRoom , Bedroom, Bathroom, Kitchen , SwpRob1 , SwpRob2 , Bob;
3  }
```

图 2　实体声明的实例

初始位置定义的语法如图 3 所示. 初始位置定义 (*init-loc-definition*) 由关键字 loc 和嵌套关系列表 (*nested-relation-list*) 组成. 嵌套关系列表是一些空间上的嵌套关系 (*nested-relation*) 组成的列表. 嵌套关系由一个实体的标识符和它初始时单层嵌套的所有实体的标识符列表组成.

初始位置定义的实例如图 4 所示, 该实例表示场景初始时 *SwpRob*1 和 *SwpRob*2 位于 *LivingRoom* 内, 而 *Bob* 则位于 *Bedroom* 内.

```
<init-loc-definition>  ::=  "loc" "{" <nested-relation-list> "}"
<nested-relation-list>::=  <nested-relation> | <nested-relation> <nested-relation-list>
<nested-relation>      ::=  <identifier> ":" <identifier-list> ";"
```

图 3 初始位置定义的语法

```
1  loc {
2     LivingRoom: SwpRob1, SwpRob2;
3     Bedroom: Bob;
4  }
```

图 4 初始位置定义的实例

行动规则定义的语法如图 5 所示. 行动规则定义 (*action-rules-definition*) 由关键字 act 和行动规则列表 (*action-rules-list*) 组成. 行动规则列表是若干实体地行动规则 (*action-rules*) 组成的列表. 行动规则由一个实体的标识符和描述其行动过程的过程列表 (*process-list*) 组成. 过程列表是由若干并行的过程 (*process*) 组成的列表. 过程包含一段语句序列 (*statement-seq*). 语句序列是若干语句 (*statement*) 的顺序执行序列.

```
<action-rules-definition>  ::=  "act" "{" <action-rules-list> "}"
<action-rules-list>        ::=  <action-rules> | <action-rules> <action-rules-list>
<action-rules>             ::=  <identifier> "{" <process-list> "}" ";"
<process-list>             ::=  <process> | <process> <process-list>
<process>                  ::=  "{" <statement-seq> "}"
<statement-seq>            ::=  <statement> | <statement> <statement-seq>
<statement>                ::=  <move-statement> | <comm-statement>
<move-statement>           ::=  <identifier> ";" | "enter" <identifier> ";" | "exit" <identifier> ";"
                              | "get" <identifier> ";" | "put" <identifier> ";"
<comm-statement>           ::=  "send" "{" <move-statement-seq> "}" "to" <identifier> ";"
                              | "recv" <identifier1> "from" <identifier2> ";" | "send notice to" <identifier> ";"
                              | "recv notice from" <identifier> ";"
<move-statement-seq>       ::=  <move-statement> | <move-statement> <move-statement-seq>
```

图 5 行动规则定义的语法

语句分为移动型语句 (*move-statement*) 和通信型语句 (*comm-statement*). 为了方便描述, 这里约定, 如果两个实体同时被另一个实体单层嵌套, 或者同时不被任何实体嵌套, 就称两个实体位于同一空间层次. 动作型语句可以是: (1) <*identifier*>";", 表示一个实体、信道或者一些通信后确定的变量; (2) "enter"<*identifier*>";", 表示进入名为 *identifier* 的实体 (本实体必须和进入的目标实体位于同一空间层次); (3) "exit"<*identifier*>";", 表示离开名为 *identifier* 的实体 (离开的目标实体必须单层嵌套本实体); (4) "get"<*identifier*>";", 表示收取名为 *identifier* 的实体 (本实体必须和收取的目标实体位于同一空间层次); (5) "put"<*identifier*>";", 表示送出名为 *identifier* 的实体 (本实体必须单层嵌套送出的目标实体). 通信型语句可以是: (1) "send""{"<*move-statement-seq*>"}""to"<*identifier*>";", 表示向名为 *identifier* 的信道发送由 *move-statement-seq* 描述的一系列指令; (2) "recv"<$identifier_1$>"from"<$identifier_2$>";", 表示从名为 $identifier_2$ 的信道接收一系列指令, 并用其替换本过程后续语句中出现的变量 $identifier_1$; (3) "send notice to"<*identifier*>";", 表示向名为 *identifier* 的信道发送通知类消息 (一般用于同步); (4) "recv notice

from”<*identifier*>“;”, 表示从名为 *identifier* 的信道接收通知类消息 (一般用于同步).

行动规则定义的实例如图 6 所示, 该实例描述了 *Bob* 命令 *SwpRob*1 和 *SwpRob*2 分别进入 *Bathroom* 和 *Kitchen* 的过程. 由于 *Bob* 命令 *SwpRob*1 和 *SwpRob*2 的顺序并不固定, 因此 *Bob* 命令 *SwpRob*1 和 *SwpRob*2 的行为可以视为并行的过程. *Bob* 在信道 *c*1 和 *c*2 上分别发送 *SwpRob*1 和 *SwpRob*2 的目的地 (*Bathroom* 和 *Kitchen*), *SwpRob*1 和 *SwpRob*2 在对应的信道上接收 *Bob* 发送过来的目的地信息, 然后离开客厅并进入各自的目的地.

```
1   act {
2       Bob {
3           {
4               send { Bathroom; } to c1;
5           }
6           {
7               send { Kitchen; } to c2;
8           }
9       };
10      SwpRob1 {
11          {
12              recv R1 from c1;
13              exit LivingRoom;
14              enter R1;
15          }
16      };
17      SwpRob2 {
18          {
19              recv R2 from c2;
20              exit LivingRoom;
21              enter R2;
22          }
23      };
24  }
```

图 6　行动规则定义的实例

3.2　MLMC 到 ACGC 的转换

首先, 根据实体声明构建每个实体所对应的 ambient. Ambient 的名字与实体的名字相同, 其内部的 ACGC 进程先设为空进程. 例如, 根据上述家居场景的实体声明可得到 7 个 ambient, 分别为: *LivingRoom*[**0**]、*Bedroom*[**0**]、*Bathroom*[**0**]、*Kitchen*[**0**]、*SwpRob*1[**0**]、*SwpRob*2[**0**] 和 *Bob*[**0**].

然后, 根据行动规则定义向代表实体的 ambient 内添加表示其行为的 ACGC 进程. 每个实体的每个过程都要转换为一个对应的 ACGC 进程. 将过程转换为 ACGC 进程的方法是: 根据表 1 的转换规则, 依次将过程中的每条语句转换为对应的能力或者通信前缀并串接起来, 然后添加在一个空进程之前. 如果某个实体的行动规则由多个并行的过程构成, 那么由这些过程转换而来的 ACGC 进程同样以并行的方式加入代表实体的 ambient 中. 例如根据上述家居场景的行动规则定义, 可在代表 *Bob*、*SwpRob*1 和 *SwpRob*2 的 ambient 内添加描述其行动的 ACGC 进程, 分别得到 *Bob*[*c*1⟨*Bathroom*⟩.**0**|*c*2⟨*Kitchen*⟩.**0**]、*SwpRob*1[*c*1(*R*1).*out LivingRoom*.*in R*1.**0**] 以及 *SwpRob*2[*c*2(*R*2).*out LivingRoom*.*in R*2.**0**].

表 1　MLMC 语句到能力或通信前缀的转换规则

MLMC语句	能力或通信前缀
<*identifier*>	*n*, *n*与*identifier*同名
“enter” <*identifier*>	*in n*, *n*与*identifier*同名
“exit”<*identifier*>	*out n*, *n*与*identifier*同名
“get”<*identifier*>	*pull n*, *n*与*identifier*同名
“put”<*identifier*>	*push n*, *n*与*identifier*同名
“send” “{” <*move-statement-seq*> “}” “to” <*identifier*>	*m*⟨*N*⟩, *m*与*identifier*同名, *N*是由*move-statement-seq*递归转换而来的一串能力
“recv” <$identifier_1$>“from”<$identifier_2$>	*m*(*n*), *m*与$identifier_2$同名, *n*与$identifier_1$同名
“send notice to”<*identifier*>	*m*⟨*ε*⟩, *m*与*identifier*同名
“recv notice from”<*identifier*>	*m*(*n*), *m*与*identifier*同名, *n*可以是任意名字但不能与MLMC模型中已存在的标识符同名

最后, 根据初始位置定义将代表实体的 ambient 嵌套起来. 对于一个给定的实体, 将它在初始位置定义中单层嵌套的所有实体的 ambient 移入该实体的 ambient 内部, 该实体 ambient 内部表示其行为的 ACGC 进程与其该实体单层嵌套的实体的 ambient 并行. 对于那些没有被任何实体嵌套的实体, 将它们的 ambient 并行起来, 合并成一个 ACGC 进程. 例如根据上述家居场景的初始位置定义 (*LivingRoom* 单层嵌套 *SwpRob*1 和 *SwpRob*2, *Bedroom* 单层嵌套 *Bob*), 将 *SwpRob*1 和 *SwpRob*2 的 ambient 移入 *LivingRoom* 的 ambient, 将 *Bob* 的 ambient 移入 *Bedroom* 的 ambient. 得到一个描述上述家居场景初始状态的 ACGC 进程:

$$\begin{aligned}&\mathit{LivingRoom}[\mathit{SwpRob}1[c1(R1).\mathit{out}\ \mathit{LivingRoom}.\mathit{in}\ R1.\mathbf{0}]\\&\mid \mathit{SwpRob}2[c2(R2).\mathit{out}\ \mathit{LivingRoom}.\mathit{in}\ R2.\mathbf{0}]]\mid \mathit{Bedroom}[\mathit{Bob}[c1\langle \mathit{Bathroom}\rangle.\mathbf{0}\mid c2\langle \mathit{Kitchen}\rangle.\mathbf{0}]]\\&\mid \mathit{Bathroom}[\mathbf{0}]\mid \mathit{Kitchen}[\mathbf{0}]\end{aligned}$$

4 工具实现

我们设计并实现了 ACGC 对 AL 的模型检测工具 ACGCCk. 本节首先介绍 ACGCCk 的结构, 然后介绍我们引入的一些优化.

4.1 软件结构

图 7 展示了 ACGCCk 的基本结构. 输入一个由 MLMC 转换而来的 ACGC 进程和一条 AL 公式, 进程词法分析模块和公式词法分析模块分别将表示进程和公式的字符串转换为 token 序列, 然后进程语法分析模块和公式语法分析模块分别解析进程和公式的 token 序列并构造相应的进程树和公式树, 最后递归检测模块在进程树和公式树上递归调用模型检测算法并给出检测结果.

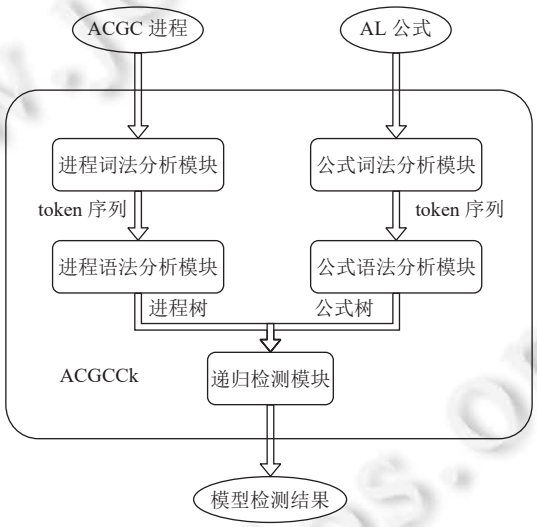
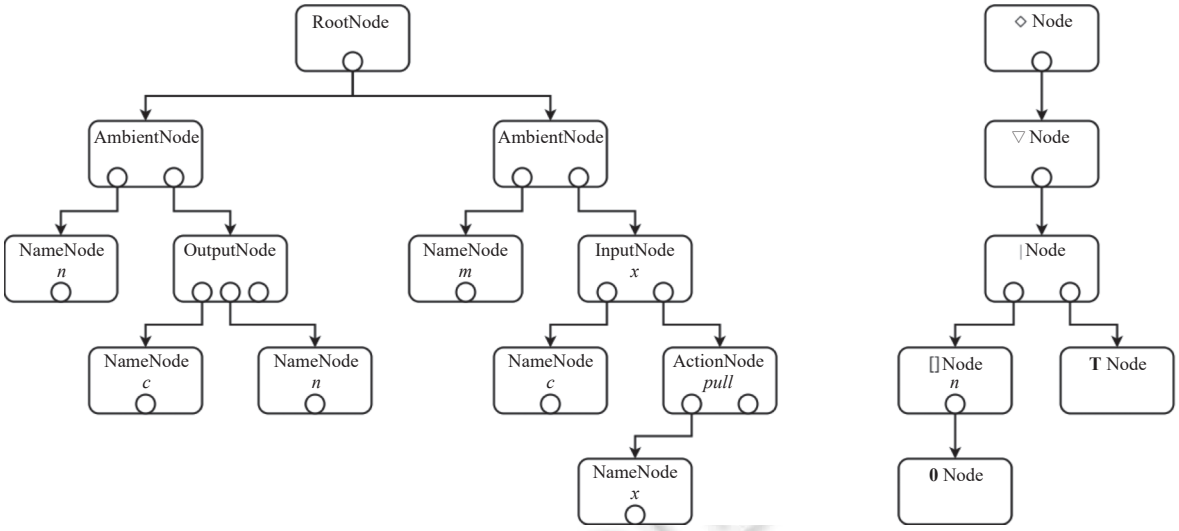


图 7 ACGCCk 的基本结构

进程树和公式树分别是表示 ACGC 进程和 AL 公式的树形数据结构. 文献 [12] 给出了一种 AC 进程的树形结构表示方法, 我们在此基础上加以扩展和改进, 设计了适用于表示 ACGC 进程的进程树和 AL 公式的公式树. 图 8 以进程 $n[c\langle n\rangle.\mathbf{0}]\mid m[c(x).\mathit{pull}\ x.\mathbf{0}]$ 和公式 $\diamond\nabla\, n[\mathbf{0}]\mid \mathbf{T}$ 为例展示了进程树和公式树的结构. 在进程树中, 除了根结点 RootNode 以外, 其他结点都表示进程中的一个能力、通信原语或者 ambient; 在公式树中, 每个结点都表示公式中的一个算子.

4.2 性能优化

通过分析算法 $\mathit{Check}(P,\mathcal{A})$ 可知, 检测过程主要的时间开销来自于对 $P\vDash \mathcal{A}\mid \mathcal{B}$ 和 $P\vDash \diamond\mathcal{A}$ 这两类问题的判定, 因此我们主要针对这两类问题进行性能优化.



(a) 进程树 (b) 公式树

图 8 进程树和公式树的结构

(1) $P \vDash \mathcal{A} \mid \mathcal{B}$

对于判定 $P \vDash \mathcal{A} \mid \mathcal{B}$ 的问题, $Check(P, \mathcal{A} \mid \mathcal{B})$ 的过程是将进程 P 分割成两个并行的进程 Q 和 R 并保证 $P \equiv Q \mid R$, 然后判定是否有 $Q \vDash \mathcal{A}$ 且 $R \vDash \mathcal{B}$. 假设 $Norm(P) = [\pi_1, \ldots, \pi_k]$ (即进程 P 由 k 个素进程并行而成), 那么一共会有 2^k 种将进程 P 一分为二的方式, 这说明 $Check(P, \mathcal{A} \mid \mathcal{B})$ 至少会具有指数级别的时间复杂度.

我们通过总结发现, 在本文研究的场景中, 大部分具有实际意义的 $\mathcal{A} \mid \mathcal{B}$ 型公式的子公式 $\mathcal{A}$、$\mathcal{B}$ 经常是 $n[C]$ 型的 (即 $\mathcal{A} \mid \mathcal{B} = n[C] \mid \mathcal{B}$ 或 $\mathcal{A} \mid n[C]$). 由引理 4 可知, 如果一个进程的正规式含有 2 个或更多的素进程, 那么该进程一定不会与任何 ambient 结构同余. 对于这种情况, 便不需要一一尝试 P 的 2^k 种二分方式, 而是只需要将进程 P 分成 1 个素进程与 $k-1$ 个素进程的并行, 易知这样的二分方式一共只有 k 种. 以判定 $P \vDash n[C] \mid \mathcal{B}$ 为例, $Check(P, n[C] \mid \mathcal{B})$ 可简化为: 令 $Norm(P) = [\pi_1, \ldots, \pi_k]$ 且 $S = \{1, \ldots, k\}$, 对任意 $S' \in 2^S$ 且 $|S'| = 1$, 令 $S'' = S - S'$, 如有 $Check(\Pi_{i \in S'} \pi_i, n[C]) =$ true 并且 $Check(\Pi_{i \in S''} \pi_i, \mathcal{B}) =$ true, 那么就有 $Check(P, n[C] \mid \mathcal{B}) =$ true.

(2) $P \vDash \diamond \mathcal{A}$

对于 $P \vDash \diamond \mathcal{A}$ 的问题, $Check(P, \diamond \mathcal{A})$ 需要用到进程 P 的可达进程集 $Reachable(P)$. 在构造 $Reachable(P)$ 的过程中, 如果不加限制地向进程数组添加数组内已有进程一步归约产生出的所有新进程, 那么就会面临状态爆炸的问题. 例如, 假设有以下归约关系: $P \to P_{11}$, $P \to P_{12}$, $P_{11} \to P_2$, $P_{12} \to P_2$, 如果不考虑优化, 那么得到的 $Reachable(P)$ 中将会存在两个 P_2.

我们的算法 $MakeReachable(P)$ 对该问题进行了一些优化. 具体地, 我们在向进程数组添加新的进程之前, 先判断所添加的进程是否和进程数组中已经存在的某个进程结构同余, 如果是则放弃添加该进程, 否则将该进程加入进程数组. 在判断两个进程是否结构同余的问题上, 我们对进程树设计了一种哈希方法, 保证只要有 $P \equiv Q$, P 和 Q 的进程树的哈希值就一定相同.

5 案例研究及实验分析

本节研究了所提方法在一个机器人送餐案例中的实际应用, 并通过实验分析了 ACGCCk 工具的性能.

5.1 案例研究

5.1.1 场景描述

图 9 描述了一个酒店利用机器人乘梯送餐的场景. 该酒店共有 9 层, 分别为 $F1$–$F9$, 并配备一部可以在这 9 层

之间往返的、内部空间只能容纳一台机器人的电梯 *Elev*. *F*2–*F*9 各自有 10 间客房, 第 *x* 层的客房分别为 *Rx*01–*Rx*10. 位于 *F*1 的两台机器人 *Rob*1 和 *Rob*2 要将各自携带的餐点 *Food*1 和 *Food*2 送往客房 *R*206 和 *R*702.

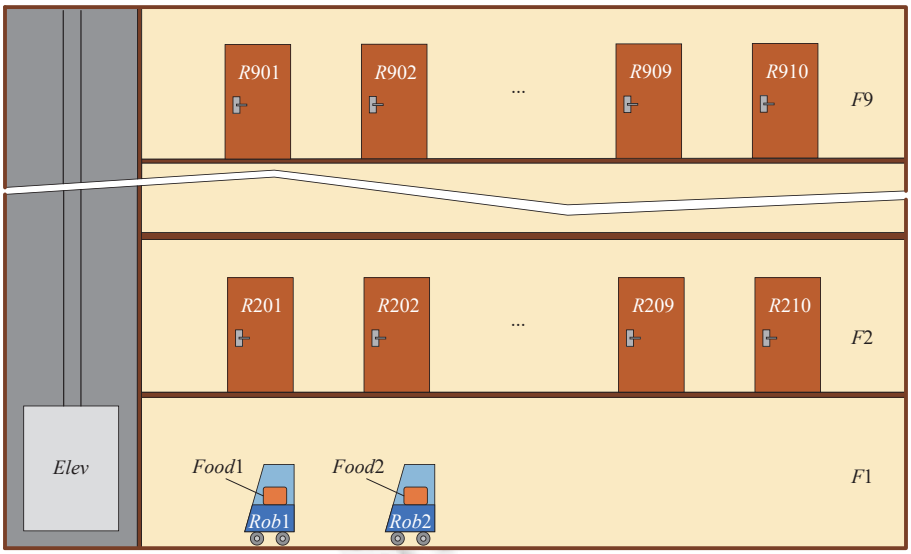


图 9　机器人送餐场景

图 10 描述了机器人乘梯的基本流程[13,14]. 机器人在出发楼层呼叫电梯, 电梯到达出发楼层并开门后向机器人发送“等待入梯”指令, 机器人成功进入电梯后告知电梯目的楼层, 电梯到达目的楼层并开门后向机器人发送“等待离梯”指令, 机器人成功离开电梯后向电梯发送“已离梯”指令, 电梯进入空闲状态.

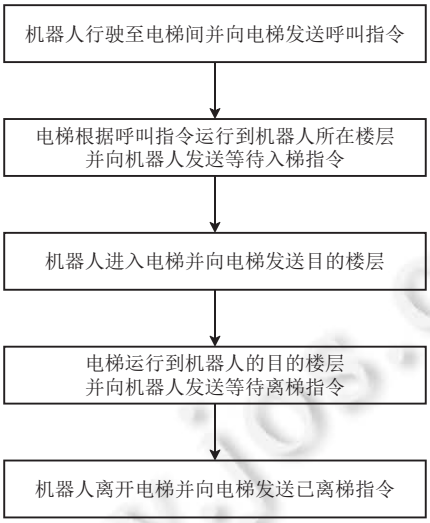


图 10　机器人乘梯的基本流程

在该场景中, 我们主要关心两条性质能否被满足: 性质 1——*Food*1 和 *Food*2 最终分别被送至 *R*206 和 *R*702, 以及性质 2——*Rob*1 和 *Rob*2 不会同时存在于 *Elev* 内.

5.1.2　场景建模及性质规约

该场景需要参与建模的实体有: 楼层 *F*1–*F*9, 客房 *R*201–*R*210、…、*R*901–*R*910, 电梯 *Elev*, 机器人 *Rob*1 和 *Rob*2, 餐点 *Food*1 和 *Food*2. 因此实体声明如图 11 所示.

根据图 8 所示的位置关系, 得到初始位置定义如图 12 所示.

根据图 10 所示的基本流程, 得到行动规则定义如图 13 所示 (我们假设电梯 *Elev* 内有两个进程分别用来处理 *Rob*1 和 *Rob*2 的请求, 并且这两个进程之间没有协调机制).

```
1  ent {
2     F1, …, F9, R201, …, R910, Elev, Rob1, Rob2, Food1, Food2;
3  }
```

图 11　机器人乘梯送餐场景的实体声明

```
1  loc {
2     F1: Rob1, Rob2;
3     F2: R201, …, R210;
4     …
5     F9: R901, …, R910;
6     Rob1: Food1;
7     Rob2: Food2;
8  }
```

图 12　机器人乘梯送餐场景的初始位置定义

```
1  act{
2     Elev {
3        {
4           recv f11 from c1;
5           enter f11;
6           send notice to c1;
7           recv f12 from c1;
8           exit f11;
9           enter f12;
10          send notice to c1;
11          recv notice from c1;
12          exit f12;
13       }
14       {
15          recv f21 from c2;
16          enter f21;
17          send notice to c2;
18          recv f22 from c2;
19          exit f21;
20          enter f22;
21          send notice to c2;
22          recv notice from c2;
23          exit f22;
24       }
25    };
26    Rob1 {
27       {
28          send { F1; } to c1;
29          recv notice from c1;
30          enter Elev;
31          send { F2; } to c1;
32          recv notice from c1;
33          exit Elev;
34          send notice to c1;
35          enter R206;
36          put Food1;
37       }
38    };
39    Rob2 {
40       {
41          send { F1; } to c2;
42          recv notice from c2;
43          enter Elev;
44          send { F7; } to c2;
45          recv notice from c2;
46          exit Elev;
47          send notice to c2;
48          enter R702;
49          put Food2;
50       }
51    };
52 }
```

图 13　机器人乘梯送餐场景的行动规则定义

将该场景的 MLMC 模型转换为 ACGC 模型, 得到 ACGC 模型 (进程) P_{fd}:

$P_{fd} \triangleq F1[$

*Rob*1[*c*1⟨*F*1⟩.*c*1(*x*11).*in Elev*.*c*1⟨*F*2⟩.*c*1(*x*12).*out Elev*.*c*1⟨*ε*⟩.*in R*206.*push Food*1.**0** | *Food*1[**0**]]

|*Rob2*[*c*2⟨*F*1⟩.*c*2(*x*21).*in Elev*.*c*2⟨*F*7⟩.*c*2(*x*22).*out Elev*.*c*2⟨*ε*⟩.*in R*702.*push Food*2.**0** | *Food*2[**0**]]

]

| *F*2[*R*201[**0**] | … | *R*210[**0**]]

…

| *F*9[*R*901[**0**] | … | *R*910[**0**]]

| *Elev*[

*c*1(*f*11).*in f*11.*c*1⟨*ε*⟩.*c*1(*f*12).*out f*11.*in f*12.*c*1⟨*ε*⟩.*c*1(*x*01).*out f*12.**0**

| *c*2(*f*21).*in f*21.*c*2⟨*ε*⟩.*c*2(*f*22).*out f*21.*in f*22.*c*2⟨*ε*⟩.*c*2(*x*02).*out f*22.**0**

]

结合上述 ACGC 模型, 将性质 1 (*Food*1 和 *Food*2 最终分别被送至 *R*206 和 *R*702) 规约为公式 $\mathcal{A}_{\text{fd1}}$:

$\mathcal{A}_{\text{fd1}} \triangleq \diamond\Box\, F2[R206[Food1[\mathbf{T}] \mid \mathbf{T}] \mid \mathbf{T}] \mid F7[R702[Food2[\mathbf{T}] \mid \mathbf{T}] \mid \mathbf{T}] \mid \mathbf{T}$

将性质 2 (*Rob*1 和 *Rob*2 不会同时存在于 *Elev* 内) 规约为 $\mathcal{A}_{\text{fd2}}$:

$\mathcal{A}_{\text{fd2}} \triangleq \neg(\diamond\nabla\, Elev[Rob1[\mathbf{T}] \mid Rob2[\mathbf{T}] \mid \mathbf{T}] \mid \mathbf{T})$

5.1.3 验证及修改

将 P_{fd} 分别与 $\mathcal{A}_{\text{fd1}}$ 及 $\mathcal{A}_{\text{fd2}}$ 输入 ACGCCk 进行验证, 得到的结果如图 14 所示.

图 14 P_{fd} 分别与 $\mathcal{A}_{\text{fd1}}$ 及 $\mathcal{A}_{\text{fd2}}$ 的验证结果

可以看到, 虽然 P_{fd} 满足 $\mathcal{A}_{\text{fd1}}$, 但并不满足 $\mathcal{A}_{\text{fd2}}$. 这说明机器人有可能在电梯里已经存在其他机器人的情况下继续进入电梯, 容易引发安全事故.

因此需要对电梯和机器人的行动规则进行修改, 使电梯同时只处理 1 个机器人的请求. 修改后的行动规则定义如图 15 所示.

将修改后的 MLMC 模型转换为 ACGC 模型, 得到 ACGC 模型 P'_{fd}.

$P_{\text{fd}}' \triangleq F1[$

*Rob*1[

cpublic⟨*c*1⟩.*c*1⟨*F*1⟩.*c*1(*x*11).*in Elev*.*c*1⟨*F*2⟩.*c*1(*x*12).*out Elev*.*c*1⟨*ε*⟩.*in R*206.*push Food*1.**0**

| *Food*1[**0**]

]

| *Rob*2[

cpublic⟨*c*2⟩.*c*2⟨*F*1⟩.*c*2(*x*21).*in Elev*.*c*2⟨*F*3⟩.*c*2(*x*22).*out Elev*.*c*2⟨*ε*⟩.*in R*702.*push Food*2.**0**

| *Food*2[**0**]

]

]

| *F*2[*R*201[**0**] | … | *R*210[**0**]]

…

| *F*9[*R*901[**0**] | … | *R*910[**0**]]

| *Elev*[

cpublic(*cx*).*cx*(*fx*1).*in fx*1.*cx*⟨*ε*⟩.*cx*(*fx*2).*out* fx1.*in fx*2.*cx*⟨*ε*⟩.*cx*(*x*01).*out fx*2

.*cpublic*(*cy*).*cy*(*fy*1).*in fy*1.*cy*⟨*ε*⟩.*cy*(*fy*2).*out fy*1.*in fy*2.*cy*⟨*ε*⟩.*cy*(*x*02).*out fy*2.**0**

]

将 P'_{fd} 分别与 $\mathcal{A}_{fd1}$ 及 $\mathcal{A}_{fd2}$ 输入 ACGCCk 进行验证, 得到的结果如图 16 所示.

```
act{
    Elev {
        {
            recv cx from cpublic;
            recv fx1 from cx;
            enter fx1;
            send notice to cx;
            recv fx2 from cx;
            exit fx1;
            enter fx2;
            send notice to cx;
            recv notice from cx;
            exit fx2;
            recv cy from cpublic;
            recv fy1 from cy;
            enter fy1;
            send notice to cy;
            recv fy2 from cy;
            exit fy1;
            enter fy2;
            send notice to cy;
            recv notice from cy;
            exit fy2;
        }
    };
    Rob1{
        {
            send { c1; } to cpublic;
            send { F1; } to c1;
            recv notice from c1;
            enter Elev;
            send { F2; } to c1;
            recv notice from c1;
            exit Elev;
            send notice to c1;
            enter R206;
            put Food1;
        }
    };
    Rob2 {
        {
            send { c2; } to cpublic;
            send { F1; } to c2;
            recv notice from c2;
            enter Elev;
            send { F7; } to c2;
            recv notice from c2;
            exit Elev;
            send notice to c2;
            enter R702;
            put Food2;
        }
    };
}
```

图 15　修改后机器人乘梯送餐场景的行动规则定义

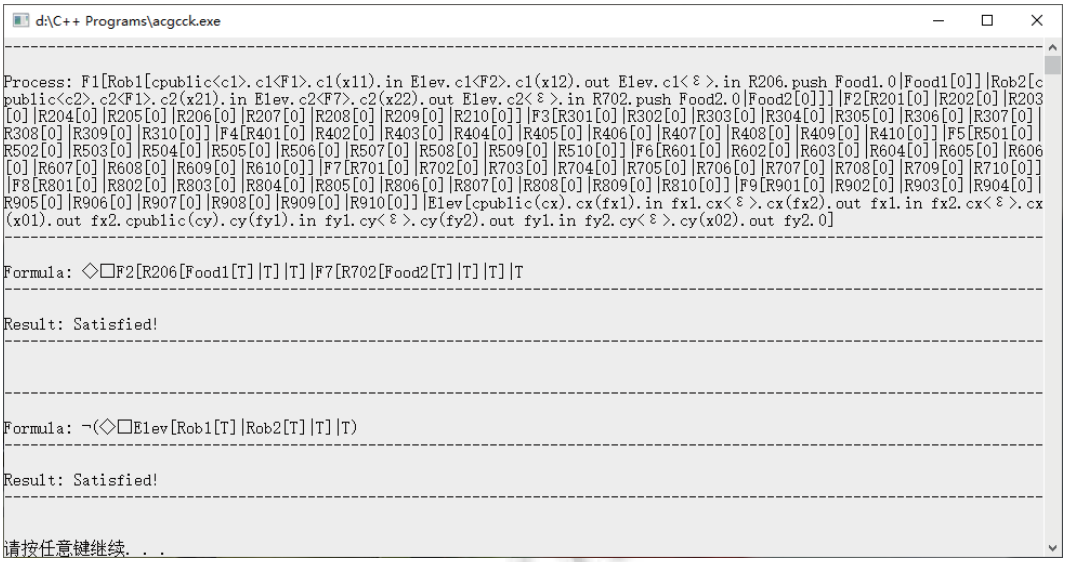


图 16 P'_{fd} 分别与 $\mathcal{A}_{\text{fd1}}$ 及 $\mathcal{A}_{\text{fd2}}$ 的验证结果

可以看到, P'_{fd} 满足了 $\mathcal{A}_{\text{fd1}}$ 和 $\mathcal{A}_{\text{fd2}}$. 这说明修改后的电梯与机器人的行动规则既可以使餐品正确交付, 又能保证电梯里最多同时只有一台机器人.

5.2 实验分析

我们设计了相关模拟实验, 以评估优化后的 ACGCCk 工具的性能. 我们研究以下两个问题.

问题 1. 进程的结构规模对检测的时间开销有何影响.

问题 2. 进程的归约规模对检测的时间开销有何影响.

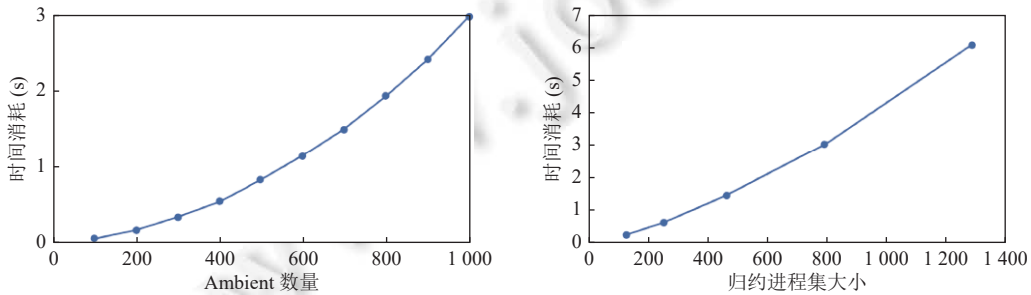
5.2.1 实验设计

对于问题 1, 我们主要以进程含有的 ambient 数量来衡量其结构规模. 结合实际情况, 我们构造了一些由若干 ambient 嵌套和并行组成的进程, 这些进程的嵌套层数均为 3, 含有 ambient 的数量从 100–1 000 不等, 并且不可归约; 对于每个进程, 尽量使其内部的嵌套关系均匀分布 (即大部分父 ambient 单层嵌套的子 ambient 的数量相近). 所验证的 AL 公式形如 $\nabla n[\mathbf{T}] \mid \mathbf{T}$ 且 n 不为进程中的任何 ambient 的名字, 以使检测时间开销接近最坏情况.

对于问题 2, 我们主要以进程的归约进程集的大小来衡量其归约规模. 结合实际情况, 我们构造了一些可以归约的进程, 这些进程的归约进程集的大小从 126–1 287 不等. 所验证的 AL 公式为 $\diamond \mathbf{0}$, 以使检测时间开销接近最坏情况.

5.2.2 实验结果与分析

图 17 展示了进程的结构规模和归约规模对检测时间开销的影响. 可以看到, 随着进程结构规模或归约规模的增加, 检测的时间开销均会快速增加, 但是 ACGCCk 工具对于一般规模的进程的检测时间开销是基本可以接受的.



(a) 进程结构规模对检测时间开销的影响 (b) 讲程归约规模对检测时间开销的影响

图 17 ACGCCk 的性能评估实验结果

6 相关工作

进程代数 (或进程演算) 是一种对并发系统进行形式化建模的代数方法. 文献 [10] 提出了 π-calculus, 该演算以名字来命名信道, 并允许名字通过信道进行传递. 文献 [5] 以 π-calculus 为基础, 引入“ambient”这一可移动、可嵌套的环境的概念, 提出了用以描述移动计算的 ambient calculus. 进一步地, 文献 [4] 提出了描述 AC 进程模型时空性质的模态逻辑 ambient logic. 文献 [11] 证明了 AC 中的$!P$ 操作和 AL 中的$\triangleright$算子均会导致 AC 对 AL 模型检测问题的不可判定, 并证明了不含 $(\nu n)P$ 和$!P$ 操作的 AC 对不含∇、 $\circledR$和$\otimes$算子的 AL 的模型检测是可判定的. 文献 [15] 在 AL 中引入新名字量词和不动点算子, 提出了基于 μ-calculus 的一阶谓词 AL. 文献 [16] 针对 AC、SA 和 BA 等的不足, 提出了 SBAP (safe boxed ambients with password), 并说明了 SBAP 在网络计算方面有一定的表现能力. 文献 [17] 针对建模需要表达 IF-THEN-ELSE 的语义以及模型需要保持有限状态的问题, 提出了 Applied-AC. 文献 [8] 使用推拉动作扩展 AC, 提出了 PAC (push and pull ambient calculus). 文献 [9] 使用全局通信机制扩展 AC, 提出了 CMC (calculus of mobility and communication). 然而上述工作大多停留于理论研究阶段, 缺乏对实际应用的探索.

在 AC/AL 支撑工具方面. 文献 [18,19] 等考察了基于 AC 的开发环境. 我们的前期工作[20]给出了动作的上下文感知应用这一特定领域的 AC 建模方法和基于 AL 的验证工具, 但是这一工具仅可用于特定领域的运行时验证, 模型只包括对于运行时路径的描述, 无法支持移动相关以及通信相关的算子, 同时, 检测算法的性能优化较为缺乏, 在物联网设备模型检测这一场景中无法得到应用.

采用模型检测这一方法提升物联网应用的可靠性与安全性已得到广泛应用. AutoTap[21]将物联网规则转换为 LTL 公式, 并通过模型检测发现潜在问题, 进行规则修复. iRuler[22]、Soteria[23]等方法用 LTL 或 CTL 等工具检测物联网规则中潜在的冲突、动作成环、动作重复等错误. Menshen[24]对于物联网应用采用混成自动机进行建模, 并进行模型检测. 这类工作较多考虑时态方面的性质, 对于物联网设备的空间相关性质考虑有所缺乏. 将 AC/AL 用于物联网应用的模型检测有一些理论方面的工作[25,26]对我们有所启发, 系统性地在物联网场景中考虑这类形式化工具的应用并进行相应的理论扩展和工具开发具有研究意义.

7 总 结

针对现有的一些模型检测方法不能很好地应用于具有移动和通信行为的物联网设备形式化验证的问题, 本文在 AC 的基础上融入了推拉动作和全局通信机制, 提出了全局通信移动环境演算 ACGC 并给出了 ACGC 对 AL 的模型检测算法, 提出了移动通信建模语言 MLMC 并给出了 MLMC 模型到 ACGC 模型的转换方法, 实现了模型检测工具 ACGCCk 并通过实验说明了 ACGCCk 具有实际应用的价值. 我们下一步的工作是在 ACGC 的基础上进行进一步扩展, 以更好地刻画物联网设备的行为; 同时继续研究优化策略, 进一步降低判定 $P \vDash \mathcal{A} \mid \mathcal{B}$ 和 $P \vDash \diamond \mathcal{A}$ 这两类问题的时间与空间开销, 实现 ACGCCk 性能的优化.

References:

[1] IoT Analytics. IoT 2021 in review: The 10 most relevant IoT developments of the year. 2021. https://iot-analytics.com/iot-2021-in-review/

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

[3] Clarke EM, Emerson EA. Design and synthesis of synchronization skeletons using branching time temporal logic. In: Proc. of the Workshop on Logic of Programs. Yorktown Heights: Springer, 1981. 52–71. [doi: 10.1007/BFb0025774]

[4] Cardelli L, Gordon AD. Anytime, anywhere: Modal logics for mobile ambients. In: Proc. of the 27th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages. Boston: ACM, 2000. 365–377. [doi: 10.1145/325694.325742]

[5] Cardelli L, Gordon AD. Mobile ambients. In: Proc. of the 1st Int'l Conf. on Foundations of Software Science and Computation Structure. Lisbon: Springer, 1998. 140–155. [doi: 10.1007/BFb0053547]

[6] Levi F, Sangiorgi D. Mobile safe ambients. ACM Trans. on Programming Languages and Systems, 2003, 25(1): 1–69. [doi: 10.1145/596980.596981]

[7] Bugliesi M, Castagna G, Crafa S. Boxed ambients. In: Proc. of the 4th Int'l Symp. on Theoretical Aspects of Computer Software. Sendai: Springer, 2001. 38–63. [doi: 10.1007/3-540-45500-0_2]

[8] Phillips I, Vigliotti MG. On reduction semantics for the push and pull ambient calculus. In: Baeza-Yates R, Montanari U, Santoro N, eds. Foundations of Information Technology in the Era of Network and Mobile Computing. Boston: Springer, 2002. 550–562. [doi: 10.1007/978-0-387-35608-2_45]

[9] Gul N. A calculus of mobility and communication for ubiquitous computing [Ph.D. Thesis]. Leicester: University of Leicester, 2015.

[10] Milner R. Communicating and Mobile Systems: the π-Calculus. Cambridge: Cambridge University Press, 1999. 75–156.

[11] Charatonik W, Dal Zilio S, Gordon AD, Mukhopadhyay S, Talbot JM. Model checking mobile ambients. Theoretical Computer Science, 2003, 308(1–3): 277–331. [doi: 10.1016/S0304-3975(02)00832-0]

[12] Nielson F, Nielson HR, Sagiv M. A kleene analysis of mobile ambients. In: Proc. of the 9th European Symp. on Programming. Berlin: Springer, 2000. 305–319. [doi: 10.1007/3-540-46425-5_20]

[13] Liu XW, Li L, Guo YP, Zhang F. Method and device for elevator and robot riding. CN108163653B. 2020-08-18 (in Chinese).

[14] Ichinose R, Takeuchi I, Teramoto T. Elevator system that autonomous mobile robot takes together with person: US, 8958910B2, 2015-02-17.

[15] Lin HM. Predicate μ-calculus for mobile ambients. Journal of Computer Science and Technology, 2005, 20(1): 95–104. [doi: 10.1007/s11390-005-0011-7]

[16] Jiang H. Model checking for mobile ambients [Ph.D. Thesis]. Guiyang: Guizhou University, 2008 (in Chinese with English abstract).

[17] Lin RD. Research on key issues of mobile ambients and model checking applications [Ph.D. Thesis]. Guangzhou: South China University of Technology, 2010 (in Chinese with English abstract).

[18] Coronato A, De Pietro G. Tools for the rapid prototyping of provably correct ambient intelligence applications. IEEE Trans. on Software Engineering, 2012, 38(4): 975–991. [doi: 10.1109/TSE.2011.67]

[19] Kato T, Miyai A, Higuchi M. IDE for the ambient calculus in distributed environments. In: Proc. of the 2014 Int'l Conf. on Industrial Automation, Information and Communications Technology. Bali: IEEE, 2014. 83–89. [doi: 10.1109/IAICT.2014.6922104]

[20] 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]

[21] Zhang LF, He WJ, Martinez J, Brackenbury N, Lu S, Ur B. AutoTap: Synthesizing and repairing trigger-action programs using LTL properties. In: Proc. of the 41st IEEE/ACM Int'l Conf. on Software Engineering. Montreal: IEEE, 2019. 281–291. [doi: 10.1109/ICSE.2019.00043]

[22] Wang Q, Datta P, Yang W, Liu S, Bates A, Gunter CA. Charting the attack surface of trigger-action IoT platforms. In: Proc. of the 2019 ACM SIGSAC Conf. on Computer and Communications Security. London: ACM, 2019. 1439–1453. [doi: 10.1145/3319535.3345662]

[23] Celik ZB, McDaniel PD, Tan G. Soteria: Automated IoT safety and security analysis. In: Proc. of the 2018 USENIX Annual Technical Conf. Boston: USENIX Association, 2018. 147–158.

[24] Bu L, Xiong W, Liang CJM, Han S, Zhang DM, Lin S, Li XD. Systematically ensuring the confidence of real-time home automation IoT systems. ACM Trans. on Cyber-physical Systems, 2018, 2(3): 22. [doi: 10.1145/3185501]

[25] Ranganathan A, Campbell RH. Provably correct pervasive computing environments. In: Proc. of the 6th IEEE Int'l Conf. on Pervasive Computing and Communications. Hong Kong: IEEE, 2008. 160–169. [doi: 10.1109/PERCOM.2008.116]

[26] Li XS, Tao XP, Lu J. Programming method and formalization for activity-oriented context-aware applications. In: Proc. of the 12th Int'l Conf. on Ubiquitous Intelligence and Computing and the 12th Int'l Conf. on Autonomic and Trusted Computing and the 15th IEEE Int'l Conf. on Scalable Computing and Communications and its Associated Workshops. Beijing: IEEE, 2015. 174–181. [doi: 10.1109/UIC-ATC-ScalCom-CBDCom-IoP.2015.48]

附中文参考文献:

[13] 刘熙旺, 李良, 郭雅萍, 章飞. 电梯及机器人乘梯的方法和装置. CN108163653B, 2020-08-18.

[16] 江华. 界程演算模型检测 [博士学位论文]. 贵阳: 贵州大学, 2008.

[17] 林荣德. 移动界程演算及模型检测应用的关键问题研究 [博士学位论文]. 广州: 华南理工大学, 2010.

[20] 李晅松, 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]

刘靖宇(1996－), 男, 硕士生, CCF 学生会员, 主要研究领域为形式化技术.

叶海波(1987－), 男, 博士, 副教授, CCF 专业会员, 主要研究领域为普适计算, 物联网.

李晅松(1985－), 男, 博士, 副教授, CCF 专业会员, 主要研究领域为软件方法学, 形式化技术, 普适计算技术, 物联网安全.

宋巍(1981－), 男, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为软件工程与方法学, 形式化方法, 服务计算.

陈芝菲(1990－), 女, 博士, 副教授, CCF 专业会员, 主要研究领域为程序分析, 软件测试, 软件维护.