中国科学: 信息科学 2019 年 第 49 卷 第 11 期: 1428–1450
SCIENTIA SINICA Informationis
软件技术前沿专题 • 进展

面向环境非确定性的信息物理融合系统测试技术研究

秦逸, 许畅*, 陈紫琦, 吕建

南京大学计算机科学与技术系计算机软件新技术国家重点实验室, 南京 210093
* 通信作者. E-mail: changxu@nju.edu.cn

收稿日期: 2018–11–12; 修回日期: 2019–05–20; 接受日期: 2019–08–20; 网络出版日期: 2019–11–08

国家重点研发计划 (批准号: 2017YFB1001801)、国家自然科学基金 (批准号: 61932021, 61902173) 和软件新技术与产业化协同创新中心 (江苏) 资助项目

摘要 信息物理融合系统 (cyber-physical system, CPS) 是 “人 – 机 – 物” 三元融合模式背景下新型软件基础设施的重要一环. 为了处理不断涌现的新环境、新模式和新平台, CPS 软件必须具备一定质量保障水平的可成长和可持续演化能力. 然而 CPS 软件在与环境交互过程中所必然面对的环境非确定性给 CPS 软件的质量保障带来了新的挑战. 本文深入分析了环境非确定性中的感知非确定性和规约非确定性分别给软件测试研究带来的挑战. 为了应对这些挑战, 提出了一个迭代式的研究框架, 用于有效测试受环境非确定性影响的 CPS 软件. 基于该研究框架, 介绍了现有 CPS 软件测试技术研究进展, 并重点介绍了 3 种考虑环境非确定性的软件测试技术, 分别解决 CPS 软件测试中的测试输入生成、测试预言生成和环境适应性评估问题. 本文基于一个自适应自控小车的 CPS 软件实例, 初步验证了所介绍方法测试 CPS 软件的有效性.

关键词 信息物理融合系统, 软件测试, 环境非确定性, 测试输入生成, 测试预言生成, 环境适应性评估

1 引言

信息技术的蓬勃发展不断加速着社会的信息化进程, 同时也刺激并驱使着软件工程研究者开始关注计算机软件及其运行平台的不断演化. 面向互联网/物联网平台的软件范型研究方兴未艾 [1~3], 基于云模式、数据驱动模式和 “人 – 机 – 物” 三元融合模式的新型软件方法学研究已经开始出现[1)]. 不

1) 科技部. 国家重点研发计划 “云计算和大数据” 重点专项 2017 年度申报指南.

引用格式: 秦逸, 许畅, 陈紫琦, 等. 面向环境非确定性的信息物理融合系统测试技术研究. 中国科学: 信息科学, 2019, 49: 1428–1450, doi: 10.1360/N112018-00305
Qin Y, Xu C, Chen Z Q, et al. Software testing for cyber-physical systems suffering uncertainty (in Chinese). Sci Sin Inform, 2019, 49: 1428–1450, doi: 10.1360/N112018-00305

断涌现的新环境、新模式和新平台, 要求软件本身具备一定的可成长和可持续演化能力[2)3)], 以应对环境资源不断变迁的挑战.

信息物理融合系统 (cyber-physical systems, CPS) 是由一组高度融合的物理和软件组件构成的工程系统, 也是云计算、大数据和 "人 – 机 – 物" 三元融合模式背景下新型软件基础设施的重要一环. CPS 软件将计算设备、网络设备、动作控制设备和物理世界感知设备紧密地整合在一个系统中. 自信息物理融合系统这一概念于 21 世纪初被首次提出起, 材料技术、通讯技术和自动化技术的飞速发展使得 CPS 软件时刻面临着新环境 (如上下文感知环境) 和新平台 (如自动驾驶汽车、无人飞行器) 的挑战. 然而由于现有 CPS 软件可成长和可持续演化能力缺乏有效的质量保障手段, 使得 CPS 软件难以持续、高质量地支撑信息物理融合系统与其运行环境、平台的交互过程. 软件缺陷导致的系统失效情况时有发生, 并造成过灾难性的后果 (例如, 印度尼西亚狮子航空 JT610 航班和埃塞俄比亚航空 ET302 航班两起坠毁事故共造成 346 名乘客和机组人员遇难).

软件测试是以查找出错误为目的的执行程序的过程, 也是被使用最为广泛的软件质量保障方法. 从信息物理融合系统刚刚出现开始, 就有软件工程研究者展开了对 CPS 软件测试工作的研究[4~7]. 相比于传统软件, CPS 软件最主要的特点就是需要额外考虑软件与物理世界的交互逻辑 (即对动作控制设备和物理世界感知设备的控制和使用). 与运行于信息空间的传统软件不同, 软件与物理世界的交互会受到环境非确定性的影响. 而上述已有工作都忽视了环境非确定性对 CPS 软件测试带来的挑战.

在对信息物理融合系统的研究中, 环境非确定性是无法被回避的一个基本要素. ACM 会士、美国 Carnegie Mellon University 计算机系教授 Garlan 指出, "对于与环境交互的系统, 非确定性是固有存在的"[8]. 一些信息物理融合系统的相关案例也印证了环境非确定性对于确保 CPS 软件正常运行的重要性. 例如, 前述的两起航班坠毁事故, 就是由于波音公司 B737MAX 型客机的自动驾驶系统未能正确处理错误的仰角传感器读数信息所导致的, 造成了重大的人员生命和财产损失. 因此在测试这类系统的过程中, 环境非确定性应当被作为第一等要素加以对待. ACM SIGSOFT 前主席, National University of Singapore 计算机系教授 Rosenblum 亦指出, "在测试过程中, 软件所面临的非确定性问题, 是未来软件系统研究的基础性挑战之一"[9].

在已有工作中, 不少研究者都提出了他们各自对于环境非确定性的定义[10~12]. 综合他们的观点, 本文认为, CPS 软件所面临的环境非确定性是一种由不完整信息导致的、系统无法确定在其所处实际物理环境的系统状态. 环境非确定性描述了 CPS 软件所获得环境信息处于无知和确定性之间的一种中间状态.

本文针对面向环境非确定性的 CPS 软件测试这一研究问题, 首先深入分析了环境非确定性对于测试 CPS 软件的技术挑战. 在此基础上, 本文提出了使用一个迭代式的研究框架来测试受环境非确定性影响的 CPS 软件. 该框架将原研究问题进一步细化为已知环境中的测试输入和测试预言生成, 以及未知环境中环境适应性评估 3 个子问题. 针对这 3 个子问题, 本文介绍了现有 CPS 软件测试研究和进展, 并重点介绍了 3 种测试技术, 分别解决 CPS 软件测试中的测试输入生成、测试预言生成和环境适应性评估问题. 本文通过一个示例性的自适应自控小车系统, 实地验证了所介绍技术在测试 CPS 软件时的有效性.

本文的主要工作和创新性贡献如下:

• 深入分析了环境非确定性对于 CPS 软件测试过程的挑战, 细化了面向环境非确定性的 CPS 软件测试问题.

2) https://www.darpa.mil/news-events/2015-04-08.

3) https://www.darpa.mil/program/building-resource-adaptive-software-systems.

• 提出了一个迭代式的研究框架来测试受环境非确定性影响的 CPS 软件, 将研究问题细化为非确定性影响下的测试输入、测试预言生成, 以及环境适应性评估 3 个子问题.

• 在所提出的研究框架介绍了现有研究进展, 并重点介绍了 3 种考虑环境非确定性的软件测试技术.

• 通过一个示例性的自适应自控小车系统, 验证了所介绍的 3 种软件测试技术在测试受环境非确定性影响的 CPS 软件时的有效性.

本文后续部分的内容如下. 第 2 节介绍了 CPS 软件、环境非确定性, 以及软件测试的研究背景. 第 3 节分析了环境非确定性对测试 CPS 软件的挑战, 定义了本文的研究问题. 第 4 节提出了一个迭代式的面向环境非确定性的 CPS 软件测试研究框架, 并将面向环境非确定性的 CPS 软件测试问题细化为测试输入生成、测试预言生成和环境适应性评估 3 个子问题. 基于该研究框架, 第 5~7 节分别介绍了现有 CPS 软件测试工作在上述 3 个子问题上的研究进展, 并重点介绍了 3 种信息物理融合系统测试技术, 即基于白盒采样的 CPS 软件测试输入生成技术 [13], 基于多不变式的 CPS 软件测试预言生成技术 [14] 和基于自动程序生成的环境适应性评估技术 [15]. 第 8 节通过一个自适应自控小车系统初步验证了所介绍的信息物理融合系统测试技术的有效性. 第 9 节总结全文并给出若干进一步研究的问题与方向.

2 研究背景

本节介绍了本文的研究背景, 包括信息物理融合系统软件、信息物理融合系统软件所面临的环境非确定性, 以及面向传统软件的软件测试问题.

2.1 信息物理融合系统软件

信息物理融合系统 (CPS) 是由一组高度融合的物理和软件组件构成的工程系统 [16]. 在 CPS 中, 物理组件和软件组件深度融合, 各自在不同时空维度执行自身任务的同时, 通过多种方式的交互以适应环境的变化. 作为信息物理融合系统的重要组成部分, 信息物理融合系统中的软件 (简称CPS 软件) 也受到了软件工程研究者的广泛关注. 与传统软件相比, CPS 软件除了功能性任务之外, 还承担了连接信息空间与物理空间的交互任务. 这一特点使得 CPS 软件在其生存周期中时刻面临着新环境、新平台和新模式的挑战.

2.2 环境非确定性

信息物理融合系统软件所面临的环境非确定性是一种由不完整信息导致的、系统无法确定在其所处实际物理环境的系统状态. 环境非确定性描述了 CPS 软件所获得环境信息处于无知和确定性之间的一种中间状态. 一方面, 与没有获得任何信息的无知不同, 受环境非确定性影响的 CPS 软件是可以获得其所处物理环境的部分信息 (如自动驾驶汽车测距仪报告的距离以及测距仪测量误差的范围); 另一方面, 与获得全部信息的确定性不同, 受环境非确定性影响的 CPS 软件无法得到精确完整的物理环境信息 (如上述测距仪报告的距离所受误差的具体值). 本文重点关注环境非确定性在软件角度对于 CPS 软件的影响并将环境非确定性分为环境感知非确定性和环境规约非确定性两类.

环境感知非确定性. 环境感知非确定性是涉及到环境感知的 CPS 软件都要面临的一种非确定性. 环境感知非确定性通常表现为软件无法获得所需环境参数的精确值, 而只能获得带误差的环境参数值. 造成这种不精确环境感知的原因主要有: (1) 环境感知设备的物理性能制约所导致的测量误差;

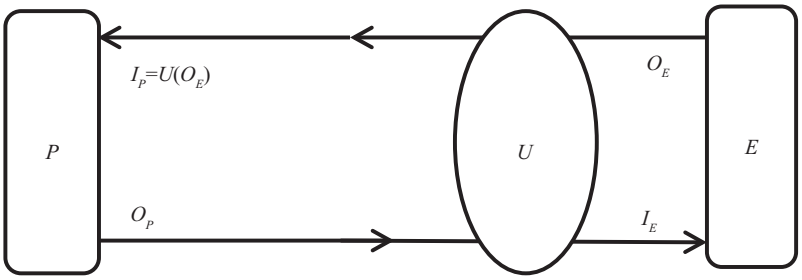


图 1 受环境非确定性影响的 CPS 软件的程序与其所属的物理环境的交互过程

Figure 1 Interaction between CPS software and its physical environment suffering uncertainty

(2) 将连续的物理空间中的值转化为离散的信息空间中的值时所产生的误差. 由于前者通常在数量级上大于后者, 因此本文主要关注前者, 即环境感知设备的测量误差.

环境规约非确定性. 环境规约非确定性是运行于动态、开放和难控的物理空间内的 CPS 软件所必然面临的一种非确定性. 环境规约非确定性通常表现为软件无法处理没有被软件规约描述的某种环境状况. 软件规约是传统软件开发过程中必不可少的产物, 但是 CPS 软件的运行环境导致了开发人员难以为其提供完整的软件规约. 造成完整软件规约不可获得的原因主要有: (1) 人类认知、描述物理世界的困难性; (2) 开发者为控制软件的复杂度而对物理环境进行的简化和假设. 前者提高了编写完整软件规约的代价和难度; 后者虽然能够获得运行于特定环境中的 CPS 软件的规约, 但是物理世界的连续性使得人们无法严格界定 CPS 软件当前的运行环境是否符合其软件规约的限定, 即无法准确、快速地发现软件规约被违反的情况.

图 1 简要描述了受环境非确定性影响的 CPS 软件的程序与其所属的物理环境的交互过程. 我们用 P 表示所关注的 CPS 软件的程序, 用 E 表示其运行所处的物理环境, 用非确定性模型 U 描述环境非确定性对于程序 – 环境交互过程的影响.

• CPS 软件的程序 P. 一个输入为 I_P, 输出为 O_P 的计算机程序. 本文所讨论的 CPS 软件包含一个信息物理融合系统内部所必要的网络协同组件, 我们暂不关注外部网络环境对于 CPS 软件的作用, 将注意力集中于程序 – 物理环境交互 (简称 “程序 – 环境交互”) 对于信息物理融合系统软件以及其测试的影响.

• 物理环境 E. 除了物理环境本身之外, 还包含 CPS 的动作控制设备和物理世界感知设备. 我们将物理环境抽象为一个黑盒程序, 其中: (1) 动作控制设备可以认为是该程序的输入端, 负责接收 P 发送的指令 (I_E); (2) 物理世界感知设备可以认为是该程序的输出端, 负责传递 P 所需要的环境感知信息 (O_E); (3) 物理环境本身的环境约束可以认为是该程序的业务逻辑, 即根据动作控制设备的运行结果调整自身环境参数从而产生物理世界感知设备所需的环境数据.

• 非确定性模型 U. 将环境非确定性抽象为从物理环境 E 的输出 O_E 到程序 P 的输入 I_P 之间的一个函数. 环境非确定性阻止了 P 获得精确、完整的环境信息 (即 $I_P = O_E$), 而将不精确、不完整的环境信息传递给 P (即 $I_P = U(O_E)$).

基于 P, E 和 U, CPS 软件的程序与物理环境的交互可以被描述为一个迭代的过程: (1) P 从 E 中获得自己的输入 I_P; (2) 执行 P 以获得其输出 O_P; (3) E 将 O_P 作为自己的输入 I_E, 执行自己的业务逻辑, 并产生对应的输出 O_E; (4) U 作用于 O_E ($I_P = U(O_E)$), 产生 P 在下一个迭代的输入 I_P. 步骤 (1)∼(4) 即构成了一个完整的程序与物理环境的交互循环.

2.3 软件测试

软件测试是以查找出错误为目的的执行程序的过程 [17], 也是被使用最为广泛的软件质量保障方法. 软件质量保障的根本目的在于保证编写出的程序的正确性, 即如下面的公式所述, 程序 P 满足软件规约 S:

$$P \models S.$$

然而在实际工程中, 程序正确性的证明往往是难以实现的. 因此软件测试希望通过找反例的方式来检验软件的质量. 软件测试一般涉及如下 3 个子问题:

- 测试输入 (test input): 软件测试应该如何向被测试程序提供输入以执行被测试程序?
- 测试预言(test oracle): 软件测试应该如何评价被测试程序的执行是否满足软件规约?
- 测试标准(test criteria): 已完成的软件测试过程在何种程度上保障了被测试程序的质量?

我们使用如下的公式从概念上描述软件测试的基本过程, 即根据测试输入执行被测试程序, 以期望发现违反测试预言的程序执行.

$$P \parallel I \not\models O.$$

3 环境非确定性与 CPS 软件测试

本节从软件测试的 3 个问题, 测试输入、测试预言和测试标准出发, 分析环境非确定性给 CPS 软件测试带来的挑战, 并从概念上定义面向环境非确定性的 CPS 软件测试问题. 整体上, 环境感知非确定性和规约非确定性从不同角度增加了 CPS 软件质量保障的困难程度. 一方面, 感知非确定性体现了新环境和新平台对于 CPS 软件的挑战. 相比于传统软件所处的环境和平台, CPS 软件即无法准确地获取准确的输入信息, 也无法确保其产生的输出信息被精确执行. 另一方面, 规约非确定性体现了新模式对 CPS 软件的挑战. 为了应对规约非确定性而引入的数据驱动的开发模式 (下文会详细讨论) 使得实现完美版本的 CPS 软件这一目标在理论上是无法实现的.

3.1 环境非确定性对测试输入带来的挑战

环境非确定性中的感知不确定行使得 CPS 软件难以精确地感知其所运行的物理环境. 这种不精确的感知导致了在生成测试输入时, 必须要考虑环境非确定性对于程序输入空间的拓展. 对于传统程序而言, 在生成测试输入的过程中, 我们只需要选取单个精确的输入值传递给被测试程序, 就能够获得该程序在该输入值下的执行. 但是对于受环境感知非确定性影响的信息物理融合系统软件而言, 无法准确地控制环境以让被测试程序获得上述精确输入值. 因此为了观察被测试程序在该输入之下可能的执行, 需要探索一个由精确输入值为中心, 由误差范围决定边界的输入空间.

由于环境感知非确定性极大地拓展了测试输入生成技术需要探索的输入空间, 因此势必会降低测试输入生成技术探索程序输入空间的效率. 在有效性方面, 输入空间的拓展导致引导失效执行的程序输入被稀释, 增大了软件测试方法发现失效的难度. 在高效性方面, 输入空间的拓展增加了测试单个程序所需要的时间开销.

3.2 环境非确定性对测试预言带来的挑战

环境感知不确定行同样导致了在生成和使用测试预言时, 必须考虑环境非确定性对于被测试程序正确/异常行为界线的模糊. 对于传统程序而言, 我们可以精确地观察、描述程序的正确行为, 更重要的是, 确定性的运行环境使得失效成为程序出现异常行为的唯一原因. 但是对于受环境感知不确定影

响的信息物理融合系统软件而言, 我们不仅无法准确地观察、描述程序的正确行为, 而且失效不再是程序出现异常行为的唯一原因.

由于环境非确定性模糊了被测试程序正确/异常行为的界线, 因此势必会降低测试预言生成技术描述、检测程序正确行为的效率. 在有效性方面, 被测试程序正确/异常行为的界线的模糊使得软件测试方法难以准确地判断程序的正确行为, 从而将程序的正确行为误报为错误行为, 或将程序的异常行为漏报为正确行为. 在高效性方面, 被测试程序正确/异常行为界线的模糊使得软件测试方法需要采集额外的信息用于辅助判断程序的正确/异常行为, 从而增加了测试单个程序所需要的时间开销.

3.3 环境非确定性对测试标准带来的挑战

环境非确定性中的规约非确定性使得开发人员无法准确完整地建模 CPS 软件的运行环境. 对软件运行环境的局限性直接导致了开发人员很难为信息物理融合系统软件给出适用于各种可能的运行环境的软件规约. 为了填补不完整的软件规约, CPS 软件开发人员大量采用了基于数据驱动的开发模式来设计、实现 CPS 软件. 例如在 CPS 的杀手级应用自动驾驶汽车中, 基于数据驱动开发的机器学习软件被广泛用于支持自动驾驶汽车的路面状况识别、驾驶策略判断和车辆控制等重要功能 [18~20]. 下面以机器学习软件为例, 讨论基于数据驱动的开发模式对于测试标准的影响.

机器学习软件动摇了以查找错误为目的的传统软件测试方法. 美国计算机学家 Macready 于 1997 年提出的 “没有免费午餐定理 (no free lunch theorem)” [21] 指出, 没有一个学习算法可以在任何领域总是产生最准确的学习器. NFL 定理使得机器学习软件在理论上不可能在一切环境中都达到 100% 的正确率, 即机器学习程序在一般环境中运行时发生偏差甚至失效是必然的. 这使得软件开发人员不可能获得一个在一般环境中均表现正常的完美版本的机器学习软件.

在不存在完美版本软件的前提下, 传统软件测试方法中以查找错误为目的测试标准难以继续适用于机器学习软件. 因此本文认为, 与其花费大量精力去寻找机器学习程序运行中必然存在的失效, 不如在整体上评估机器学习程序软件在当前环境中运行状态, 以便为软件开发人员提供被测试机器学习软件的质量指标.

3.4 面向环境非确定性的 CPS 软件测试问题

综合前述的研究挑战, 我们用如下的两个公式从概念上描述面向环境非确定性的信息物理融合系统软件测试的研究问题. 用花体表示相对于传统软件测试方法需要额外考虑的因素.

$$P \,\|\, I \,\|\, \mathscr{E} \,\|\, \mathscr{U} \nvDash O \,\|\, \mathscr{E} \,\|\, \mathscr{U}, \tag{1}$$

$$\mathscr{P} \,\|\, I \,\|\, \mathscr{E} \,\|\, \mathscr{U} \sum \models O \,\|\, \mathscr{E} \,\|\, \mathscr{U}. \tag{2}$$

式 (1) 描述了在传统软件测试框架 (即以查找错误为目的的执行程序过程) 内, 如何测试受环境非确定性影响的信息物理融合系统软件. 公式 “$\nvDash$” 的左侧指出, 在执行程序的过程中, 相比于传统软件测试方法, 还必须考虑到环境和非确定性对于测试输入的影响. 公式 “$\nvDash$” 的右侧指出, 测试受环境非确定性影响的信息物理融合系统软件, 在判断程序运行是否正确的过程中, 相比于传统软件测试方法, 还必须考虑到环境和非确定性对于测试预言的影响.

式 (2) 描述了如何从整体上评估受环境非确定性影响的信息物理融合系统软件的质量. 与式 (1) 相比, 式 (2) 关注的软件由传统软件转变为基于数据驱动开发的软件, 特别是在信息物理融合系统中被广泛使用的机器学习软件. 正是由于前文所分析的机器学习的软件与传统软件之间存在的差异性, 使得对其质量保障需要跳出传统软件测试框架. 我们认为测试 CPS 中的机器学习软件, 应当更加关

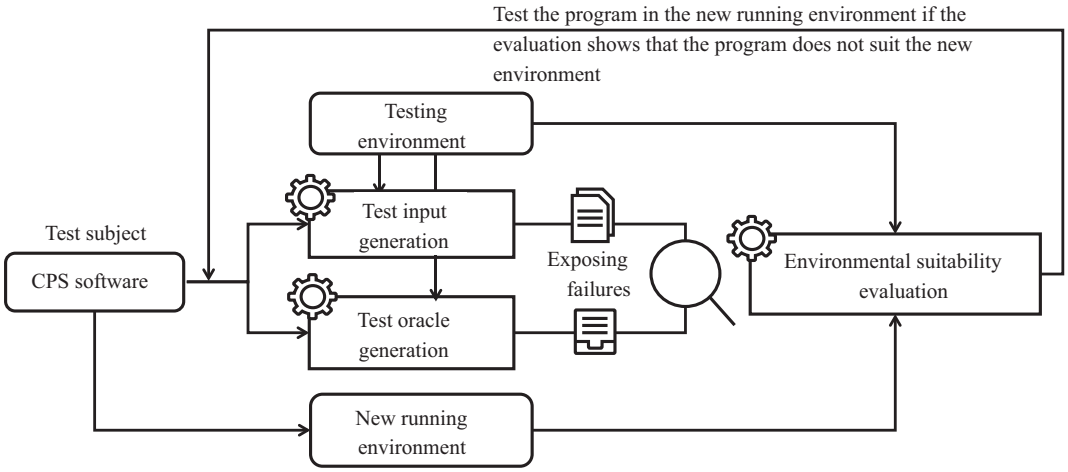


图 2　研究框架: 面向环境非确定性的信息物理融合系统软件测试

Figure 2　Research framework: testing CPS software suffering uncertainty

注评估被测试软件在整体上能够以多少程度满足需求, 而不是只关注寻找机器学习程序在执行过程中必然存在的失效 (即式 (2) 中用 "$\sum \models$" 代替式 (1) 中的 "$\not\models$").

4　一个迭代式的 CPS 软件测试研究框架

针对面向环境非确定性的 CPS 软件测试的研究问题 (即式 (1) 和 (2)), 我们使用一个如图 2 所示研究框架来快速、有效地测试受环境非确定性影响的 CPS 软件. 首先对于运行于特定环境 (如软件的调试环境) 中的 CPS 软件, 仍采用传统软件测试框架对其进行测试, 即分别对被测试的 CPS 软件进行适用于当前运行环境的、考虑环境非确定性影响的 (i) 测试输入生成和 (ii) 测试预言生成. 通过生成的测试输入和测试预言, 在传统软件测试框架内对运行于特定环境中的 CPS 软件进行测试 (即回答了研究问题式 (1)). 在完成了运行于特定环境中的 CPS 软件测试后, 通过使用 (iii) 环境适应性评估的方法来监控运行于新的非特定环境 (如软件的实际运行环境) 中的 CPS 软件. 环境适应性评估旨在回答如下问题, 即在特定环境中完成测试、有了一定质量基础的 CPS 软件, 能否以同样或者相近的质量运行于新的环境, 即适应于新的环境. 如果被测试软件在新环境中的环境适应性较好, 则认为被测试程序在新环境中运行的质量已经有了保障. 反之, 如果被测试软件在新环境中的环境适应性不佳, 则认为被测试程序在新环境中运行的质量存在缺陷, 需要测试框架生成适应于新环境的测试输入和测试预言来进行新的测试. 我们所使用的环境适应性评估能够从整体上评估运行于一般环境中的 CPS 的质量 (即回答了研究问题式 (2)).

在技术实现上, 本文研究框架中需要解决的信息物理融合系统软件测试的关键问题如下.

• **高效探索程序输入空间.** CPS 软件所面对的环境非确定性极大地拓展了程序的输入空间, 降低了传统测试输入生成技术应用于 CPS 软件测试时的效率. 我们认为, 高效的程序输入空间探索应包括以下两点: (1) 单位时间内能够生成尽可能多的测试输入; (2) 能够发现尽可能多的、能够引导程序失效执行的测试输入. 前一点要求测试输入生成技术能够以较低的时间成本生成单条测试输入, 后一点要求测试输入生成技术不遗漏会引导程序失效执行的测试输入.

• **准确检测程序异常状态.** CPS 软件所面对的环境非确定性模糊了程序正确/异常行为的界线,

降低了传统测试预言生成技术应用于 CPS 软件测试时的准确率. 我们认为, 准确的程序异常状态检测应包括以下两点: (1) 异常状态检测的漏报率低; (2) 异常状态检测的误报率低. 前一点要求测试预言生成技术尽可能少地将程序的异常状态检测为正确状态, 后一点要求测试预言生成技术尽可能少地将程序的正确状态检测为异常状态.

• **有效评估软件的环境适应性.** CPS 软件为了应对环境非确定性而引入的基于数据驱动开发模式动摇了传统的测试标准保障 CPS 软件质量时的有效性. 机器学习软件即是基于数据驱动开发模式的一种代表性软件, 在众多 CPS 中有着广泛的应用. 我们认为, 在整体上评估机器学习程序软件在当前环境中运行状态, 能够更加有效地为软件开发人员提供关于被测试软件在当前环境中的质量信息.

在后续章节中, 我们将简要介绍已有 CPS 软件测试工作在这 3 个关键问题上的研究进展, 并分别针对 3 个问题, 重点介绍 3 种测试技术.

5 CPS 软件测试输入生成问题

CPS 软件测试输入生成技术是解决本文的第 1 个研究问题的重要支撑技术, 即在传统软件测试框架内对特定环境中运行的 CPS 软件进行测试 (如式 (1) 所示).

CPS 软件测试输入生成技术负责生成被测试 CPS 软件程序的输入, 用于探索其受到环境交互约束 (即 E) 以及环境非确定性影响 (即 U) 的输入空间, 以发现被测试程序执行过程中可能存在的失效 (即式 (1) 中 $\nvDash$ 左侧部分).

5.1 现有工作进展

在软件测试中, 测试输入生成技术负责产生用于驱动被测试软件的程序的输入. 测试输入的产生并非是任意的, 而是遵循一定的生成策略, 其目的在于通过不同的测试输入, 尽可能地探索程序的输入空间以覆盖程序的状态空间, 从而达到发现程序执行过程中可能发生的失效. 在这一过程中, 测试输入的生成策略对于测试输入生成技术的有效性和高效性起着决定性的作用. 本小节以测试输入的生成策略为主线, 分别介绍基于随机测试和基于程序分析的 CPS 软件测试输入生成技术, 并分析其在有效性和高效性上的优势和不足.

5.1.1 基于随机测试的测试输入生成

随机测试是一种黑盒软件测试技术, 通过生成随机且独立的测试输入来探索软件的输入空间. CPS 研究者很早就将随机测试引入 CPS 软件测试工作中. 如 Fredericks 等 [22] 提出了一个基于效用函数的方法 Veritas, 用于设计并生成信息物理融合系统软件的测试输入. Veritas 通过检测被测试信息物理融合系统软件所处环境条件的改变, 实时地调整所产生的测试输入以确保被测试的信息物理融合系统软件的安全运行. Amalfitano 等 [23] 通过分析信息物理融合系统软件所处环境的上下文变化规律, 总结出了一组环境上下文事件, 并为该组事件随机地产生环境上下文序列, 用于测试信息物理融合系统软件. Jang 等 [24] 提出了一个用于模拟环境上下文数据和服务执行命令的信息物理融合系统软件测试框架. 除了纯随机生成策略, 该框架还可根据开发人员给出的规约, 有针对性地生成受特定环境因素、设备因素和人类活动因素影响的被测试软件的环境感知数据. Lu 等 [25] 提出了一组基于被测试程序和其所处环境交互的测试准则, 用于指导测试输入生成技术以覆盖特定的程序 – 环境交互状态下的数据流. 随后 Lu 等 [26] 又将该工作拓展到环境信息一致性检测方面, 提出了针对具备解决外部环境不一致能力的信息物理融合系统软件的测试准则.

除了学术界的研究外, 随机测试也是工业界使用最为广泛的信息物理融合系统软件测试方法. 如 Google 公司为 Android 平台开发的自动随机测试工具 Monkey, 已成为使用最为广泛的智能手机应用测试工具. 而在信息物理融合系统软件的杀手级应用自动驾驶汽车中, 各大公司 [18~20] 在测试其自动驾驶软件时, 无不采用过 “真实路面状况 + 人类司机监督” 的测试模式, 通过自动驾驶汽车在实际道路上有/无目的的行驶来丰富对自动驾驶软件的测试.

随机测试策略的优势在于其产生测试输入生成的高效性. 由于随机测试策略通过随机生成的方法来产生独立的测试输入, 因此相比于其他测试输入生成策略, 在单位时间内能够产生的测试输入的数量是最多的. 在执行被测试程序效率相同的情况下, 随机测试策略无疑能够在单位之间内探索更多的被测试程序的执行. 随机测试策略的劣势在于其产生的测试输入在暴露程序失效运行时的低有效性. 由于随机测试对于测试输入控件的探索缺乏系统性, 因此相比于其他测试输入生成策略, 随机测试策略可能需要使用更多的测试输入, 才能够暴露出被测试程序可能存在的失效运行.

5.1.2 基于程序分析的测试输入生成

程序分析技术是软件工程领域的关键技术之一, 其也被广泛应用于软件质量保障领域. 程序分析中的静态分析技术能够在不执行具体程序的情况下, 对被测试软件的代码或模型进行分析, 以检验某些特定的指标 (如被分析软件代码的规范性、被扫描软件模型的安全性等). 静态分析技术被大量应用于从模型检测的角度对信息物理融合系统软件的可靠性、安全性、稳定性和一致性进行验证 [11, 27, 28]. 同时, 不少研究者亦将静态程序分析技术用于支持软件测试输入生成. Wang 等 [29] 分析了被测试程序中能够引起程序行为发生改变的环境感知数据变化节点, 并通过使用这样的环境感知数据变化节点来指导测试输入的生成. Ramirez 等 [12] 通过分析被测试软件的设计模型, 发现可能引起程序异常行为的特定环境条件组合, 并在所发现的环境条件组合下执行被测试程序, 以发现可能存在的失效. Greibe 等 [30] 在被测试软件的设计模型中融入环境感知信息, 并通过模型变换的方法探索被测试软件的设计模型在各种环境条件下可能的行为. Esfahani 等 [31] 提出了一种基于模糊数学的方法, 用于在被测试软件的体系结构层面推演环境非确定性可能造成的影响, 并发现其中可发生的程序异常运行状态.

与静态分析技术相对应, 动态分析技术需要通过执行对象程序, 记录下程序执行信息, 并以此为基础分析程序执行过程中所满足的性质. Adamsen 等 [32] 提出了一种基于对抗策略的测试输入生成框架, 用于系统性地探索可能引起信息物理融合系统软件异常行为的非常见时间序列. Anand 等 [33] 提出了一种基于动态符号执行的测试输入生成技术, 用于系统地探索 Android 应用的状态空间. Matinnejad 等 [34] 提出了一种基于多向性分析的输入空间搜索策略, 用于为信息物理融合系统软件产生高覆盖率的测试输入.

基于程序分析的测试输入生成的优势在于暴露被测试软件失效运行的高有效性. 由于这类方法通过对软件的代码、模型, 以及执行信息进行分析来辅助测试输入的生成, 因此从理论上能够准确覆盖被测试程序的整个输入空间, 也能够不漏掉任何会引导程序执行失效的测试输入. 基于程序分析的测试输入生成的劣势在于产生测试输入过程的低效性. 由于程序分析相比于随机采样需要花费大量时间, 因此这类方法的效率大大落后于基于随机测试的测试输入生成技术. 此外, 由于基于程序分析的测试输入生成方法往往需要获得被测试程序所处环境的模型, 而对于 CPS 软件而言, 其环境模型通常时不可得, 因此这也限制了基于程序分析的测试输入生成方法在工业级 CPS 软件测试中的应用.

5.2 SIT 方法概述

通过分析基于随机测试的测试输入生成方法和基于程序分析的测试输入生成方法两者的优劣, 本

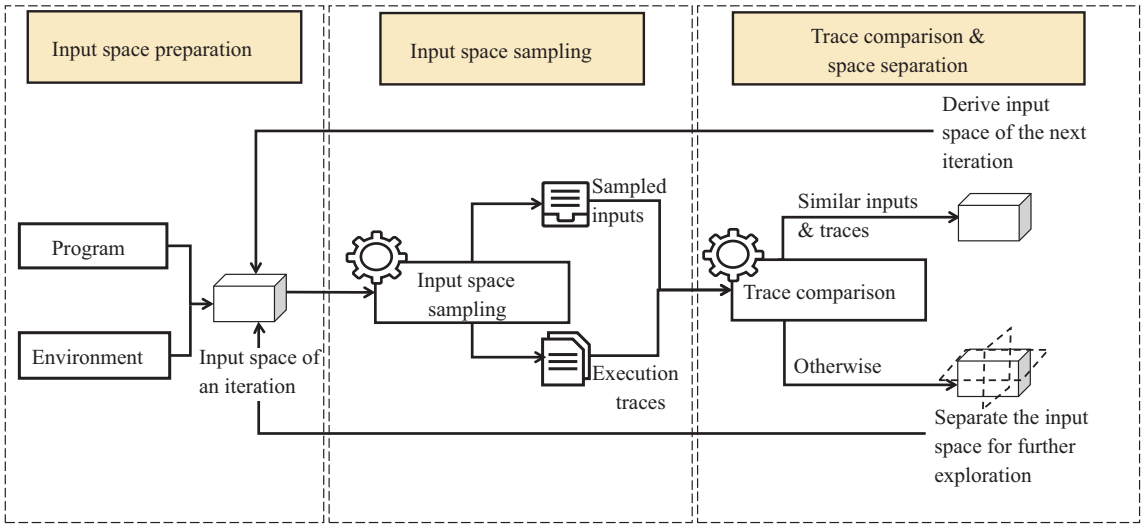


图 3 (网络版彩图) SIT: 基于采样的交互式测试输入生成概览

Figure 3 (Color online) Overview of SIT: sample-based interactive testing

文会重点介绍一种基于采样的交互式测试方法 (sampling-based interactive testing, SIT), 用于测试受环境非确定性影响的 CPS 软件. SIT 方法综合了随机测试方法和基于程序分析的测试方法的优点, 通过自适应采样高效、系统地探索被测试 CPS 程序的输入空间. 具体而言, SIT 方法通过系统性地采样被测试程序 P 的某个输入空间 is, 来探索可能导致 P 的不同执行的不同输入值. SIT 通过比较 P 的执行序列来判断两个执行是 “不同” 的还是 “相近” 的. 对于给定的待探索输入空间 is, SIT 方法首先选取其边界上的所有顶点, 并以这些顶点为输入值执行被测试程序 P, 观察并记录下 P 在这些执行中的执行序列. 这些基于被采样点的被测试程序 P 的执行称为采样执行. 然后基于这些采样执行相似性, SIT 方法通过划分的方法对 is 进行进一步的探索. 对 is 的划分将产生一组新的、更小的输入空间. SIT 方法对一个输入空间 is 进行划分时, 在每一个维度上都将 is 所对应的区间一分为二. 对于一个 n 维的输入空间, 划分后产生 2^n 个较小的输入空间. SIT 方法重复对输入空间的探索 – 划分的过程, 直到 (1) 被探索的输入空间本身已足够小, 或者 (2) 被探索的输入空间中所有的采样执行均非常相似. 在这一过程中, P 的失效执行被添加至失效执行序列集合, 而正常执行则被添加至正常执行序列集合, 以供 SIT 方法探索其后续的迭代.

鉴于 CPS 软件在执行过程中会与外部环境进行迭代式的交互, 并从交互过程中获得所需的程序输入, 因此 SIT 方法使用树结构, 用于组织被测试程序在不同迭代中所对应的输入空间. SIT 方法使用广度优先搜索的策略探索输入空间树, 并通过自适应采样探索程序面对单个输入空间时不同的行为. 当 SIT 完成对于某一个输入空间的探索后, 会通过程序 – 环境交互过程, 将探索到的被测试程序在当前输入空间内的不同行为, 转化为其在下一次程序 – 环境交互迭代后所对应的输入空间. SIT 方法在当前迭代采样的输入和执行数量对应了其在下一次迭代中需要探索的输入空间的数量. SIT 方法终止于发现第 1 个被测试程序的失效执行, 或者终止于执行时间到达测试时间上限. 图 3 描述了 SIT 方法的整体执行过程.

6 CPS 软件测试预言生成问题

与测试输入生成技术一样, CPS 软件测试预言生成技术是解决本文的第 1 个研究问题的重要支

撑技术, 即在传统软件测试框架内对特定环境中运行的 CPS 软件进行测试 (如式 (1) 所示).

CPS 软件测试预言生成技术负责在排除环境非确定性影响 U 的前提下, 判断被测试程序 P 在特定环境 E 中的运行过程是否出现了异常状态 (即式 (1) 中 $\nvDash$ 右侧部分).

6.1 现有工作进展

在软件测试中, 测试预言技术负责产生用于判断被测试程序执行是否正确的预言. 不同于被测试程序严格相关的测试输入, 测试预言有多种表现形式, 通常可分为以下 4 类 [35]: 基于软件规约的测试预言、基于执行状态归纳的测试预言、基于隐形假设的测试预言, 以及基于人类决策的测试预言. 本小节将以测试预言分类主线, 分别介绍上述 4 种测试预言在信息物理融合系统软件中的应用, 分析其优势和不足.

6.1.1 基于软件规约的测试预言

如第 2 节所述, 测试预言的作用在于决定 “如何评价被测试程序的执行是否满足软件规约”, 因此软件规约自然成为信息物理融合系统软件测试预言的重要来源之一. Sama 等 [6] 基于有限状态状态机的运行特性提出了信息物理融合系统软件需要遵循的稳定性和一致性准则. Xu 等 [7] 通过观察信息物理融合系统软件的行为模式和失效类型, 总结出了可用于检测信息物理融合系统软件缺陷的错误模式. Tse 等 [36] 提出可使用程序运行环境特有的蜕变关系, 用于检测程序和环境交互过程中存在的异常状态.

前面所列举的基于软件规约的测试预言虽能准确识别信息物理融合系统软件的异常运行状态, 但是其测试预言的生成都需要软件测试工程师的参与. 由于人类能力的有限性, 这些有软件测试工程师参与的测试预言生成过程通常是低效的, 因此本文认为这些基于软件规约的测试预言并不适用于工程化的信息物理融合系统软件测试要求.

6.1.2 基于执行状态归纳的测试预言

基于执行状态归纳的测试预言的基本逻辑是通过记录、分析被测试程序少量的正确执行, 由计算机自动地归纳总结出被测试程序正确执行应满足的条件 (如变量的取值范围、方法参数的类型等), 并将这些自动生成的条件作为测试预言, 用于后续的测试工作 [37~40].

在信息物理融合系统软件领域, 基于执行状态归纳的测试预言能够捕捉到一般方法难以捕捉的被测试程序的异常执行状态, 因此也被广泛应用于为信息物理融合系统软件生成测试预言. Jiang 等 [41] 通过记录分析无人飞行器的控制信息, 提出了一个基于时间序列和多项式约束的不变式生成技术, 用于监控并修正无人飞行器在飞行过程中存在的异常状况. Aliabadi 等 [42] 通过同时考虑信息物理融合系统软件在时间、事件和数据维度的状态信息, 提出了一个基于数据挖掘的多维不变式生成算法. Golombek 等 [43] 为机器人控制程序建立了一个数据驱动的异常识别检测机制, 并使用熵和概率论模型来评估程序中包含的误差的影响. Steinbauer 等 [44] 提出了一个基于模型的信息物理融合系统软件检测与诊断框架, 通过记录分析被测试程序中的事件关系、方法调用, 以及进程树, 采用基于规则的推理手段对被测试程序的异常行为进行检测. Gillula 等 [45] 将数据挖掘技术引入信息物理融合系统软件的异常状态识别中, 通过挖掘被测试程序的执行状态序列, 以确定其处于安全运行状态的边界.

基于执行状态归纳的测试预言生成技术既能够高效地生成被测试程序所需的测试预言, 其生成的测试预言又能够保持较高的异常状态识别准确率. 但是这些工作都没有考虑到环境非确定性对于测试预言生成以及异常状态检测的影响, 使得受环境非确定性影响的信息物理融合系统软件生成的测试预

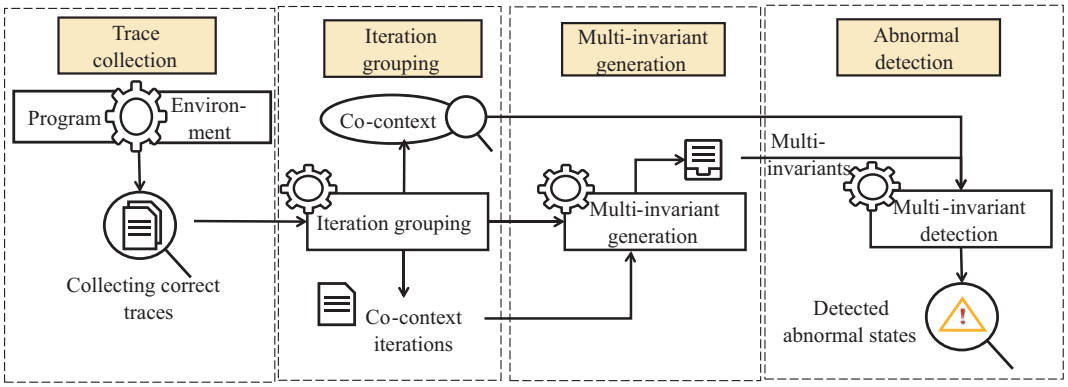


图 4 (网络版彩图) CoMID: 基于上下文的多不变式异常状态检测概览

Figure 4 (Color online) Overview of CoMID: context-based multi-invariant detection

言无法保持较高的有效性.

6.1.3 基于隐形假设和人类决策的测试预言

基于隐形假设的测试预言通常使用程序异常或崩溃作为表征被测试程序失效的标志 [35]. 研究者采用这种测试预言验证其测试方法的有效性 (如 2.2 小节所述的大部分信息物理融合系统软件测试输入生成均采用程序崩溃作为测试预言). 与前两种类型的程序预言相比, 基于隐形假设的测试预言具有检测成本低、异常状态误报率低的特点. 但是由于基于隐形假设的测试预言多关注显示化的程序异常运行状态, 如程序异常和程序崩溃, 因此并不适用于真实环境中对信息物理融合系统软件的测试 (如自动驾驶软件即使在测试阶段发生崩溃, 也有损害车辆甚至危及乘坐人员的生命的可能性).

在工业界对于信息物理融合系统软件的测试, 通常使用基于人类决策的测试预言, 即让软件测试工程师判断被测试程序是否失效. 如在自动驾驶汽车的测试过程中, 要求人类驾驶员实时观察路面情况和判断行驶情况, 在必要时直接控制被测试车辆以免因自动驾驶软件失效而发生交通事故. 虽然这种基于人类决策的测试预言在实际使用中能够保持较高的对异常状态判断的准确率, 但是由于引入了人类的参与, 因此在成本和效率上是远低于其他类型的测试预言. 另一方面, 对于人类不熟悉的信息物理融合系统任务, 如机器人控制、无人飞行器控制等, 人类的判断能力并不足以准确捕捉其中的程序异常运行状态.

6.2 CoMID 方法概述

本小节介绍一种基于上下文的多不变式检测方法 (context-based multi-invariant detection, CoMID) [14], 用于为 CPS 软件生成其测试所需的不变式, 同时基于生成的不变式, 在被测试程序运行时检测其异常状态. 图 4 描述了 CoMID 方法的整体执行过程. CoMID 方法主要由两个关键技术组成: 基于上下文的迭代执行分组和多不变式生成和检测. 前者通过考虑被测试程序在程序 – 环境交互迭代过程中的上下文信息, 细化了不变式在生成和检测过程中的适用范围, 从而平衡了由于环境交互的引入而失衡的不变式的泛化性和特定性. 后者通过综合多个不变式的检测结果, 平衡了环境非确定性在不变式生成和检测过程中干扰, 从而减小了环境非确定性对于异常状态检测准确性的影响.

在基于上下文的迭代执行分组中, 除了传统自动测试预言生成技术中收集的被测试程序执行状态信息 (如各个执行方法的参数值以及返回值) 外, CoMID 方法还额外收集了程序 P 的执行中每个迭代的程序和环境上下文信息. 对于程序上下文信息, CoMID 记录了 P 在当前迭代中所执行过的语句 id.

对于环境上下文信息, CoMID 记录了环境 E 在当前迭代开始时的属性值. CoMID 通过分析程序 P 中与环境感知相关的系统调用, 并以这些系统调用所获取的环境感知参数值作为 E 的属性值. CoMID 方法使用程序上下文区分 P 在不同迭代的执行中所承担的任务和执行的业务逻辑, 而使用环境上下文区分 P 在不同迭代中所面临的不同环境状态.

在基于多不变式生成和检测中, CoMID 通过已有的不变式推理引擎 (如 Daikon [37]) 生成单个不变式. 为了尽可能地减少环境非确定性的影响, CoMID 在生成不变式时并不只使用每个共上下文迭代分组中的程序执行状态信息的全集, 而是同时使用该全集的不同大小的子集作为 Daikon 引擎的输入, 以生成额外的不变式. 对于 Daikon 引擎中所使用的每个不变式模板, CoMID 除了使用一个共上下文迭代分组中的全部程序执行状态信息生成的不变式之外, 还从所采样的子集中额外生成了 4 个不变式. 这 5 个不变式被称为一个多不变式. 对于一个多不变式, 用于生成共上下文迭代分组的环境和程序上下文信息被称为该多不变式的上下文. 在使用多不变式时, CoMID 需要决定不变式的违反是由于程序实际的异常状态, 还是由于环境非确定性的影响所导致的. CoMID 通过一个评估函数综合考虑不变式簇中各个不变式在多个连续迭代中的检测结果, 从而尽量减小环境非确定性对于不变式检测的影响. CoMID 设计评估函数基于下述观察, 即 “一个不变式所使用的程序执行信息越少, 则非确定性导致该不变式被违反的可能性就越大; 如果一个不变式所使用的程序执行信息越多, 则非确定性导致该不变式被满足的可能性就越小”.

7 CPS 软件环境适应性评估问题

CPS 软件为了处理环境非确定性而采用的基于数据驱动的开发模式, 对 CPS 软件测试的测试标准提出了新的挑战和研究问题. 测试方法需要在以查找错误为目的的传统软件测试框架外, 从整体上评估受环境非确定性影响的 CPS 软件的质量, 如式 (2) 所示.

与传统软件开发模式不同, 在基于数据驱动的开发模式中, 开发者并不直接编写软件的业务逻辑, 而是使用机器学习算法从已有数据中提取软件所需的业务逻辑. 这种开发模式虽然能够有效处理因环境规约非确定性而导致的软件规约不完整的问题, 但是亦对这类软件的质量保障问题提出了新的挑战. 机器学习软件是数据驱动软件中具有代表性的一类软件, 被广泛应用于信息物理融合系统软件的环境感知和适应决策中 [19, 46, 47]. 本节以机器学习软件为例, 讨论当软件面临规约非确定性时, 如何通过软件测试方法有效保障软件质量.

7.1 现有工作进展

与传统软件不同, 机器学习软件的运行是两阶段式的: 在第 1 阶段, 机器学习软件从现有环境的数据中进行学习, 以提取其业务逻辑所必须的知识. 通常将这一阶段称为机器学习软件的训练阶段, 而其运行环境则称为机器学习软件的训练环境. 在第 2 阶段, 机器学习软件使用在训练阶段提取的知识, 在新的环境中完成所承担的机器学习任务. 这一阶段也被称为机器学习软件的预测阶段, 而其运行环境则被称为机器学习软件的预测环境. 在训练阶段, 机器学习软件的训练环境通常是可控并且已知的, 开发人员可以通过充分的调试, 使得机器学习软件在训练阶段的性能达到或接近令人满意的水平; 而在预测阶段, 对象软件的预测环境是不可控且部分未知的, 开发人员亦不再有能力对软件的行为做出任何改变. 由于机器学习软件在训练阶段的行为是可控且可调试的, 因此本小节关注机器学习软件在预测环境中整体运行质量的评估, 即是对机器学习软件在该环境中的运行效果进行评估和预测.

一般而言, 机器学习软件期望预测环境中的数据与其训练环境中的数据具有相同的特性, 以使得从后者中提取出来的知识能够更适应于在前者中使用. 训练环境和预测环境的数据同分布, 也是机器学习算法的重要前提假设之一. 然而在机器学习软件的实际使用过程中, 其预测环境是很难与其训练环境保持相对的统一. 当训练环境和预测环境出现了本质上的差异时, 机器学习软件从前者中提取出来的知识将不再适用于后者, 从而导致机器学习软件失效的发生 [48]. 对于评估机器学习软件在预测环境中的运行效果而言, 一个重要的问题是 "如何判断机器学习软件从训练环境中提取出的知识是否能够适应于在预测环境中的使用". 对于该问题的回答, 将有助于开发人员判断机器学习软件是否能够在当前的预测环境中以令人满意的效果运行, 亦或需要在当前预测环境中进行额外的训练和调试, 以获取新的知识.

在已有工作中, 解决上述适应性评估问题主要有两个思路, 一是在预测环境中对机器学习软件进行新一轮的测试工作; 二是对机器学习软件所使用的模型本身进行质量检测.

7.1.1 测试机器学习软件

相较于训练环境而言, 预测环境给机器学习软件测试带来的最大挑战在于机器学习软件在预测环境中的正确行为是难以定义和描述的. 通常将这类无法定义、描述软件正确行为的问题称为 "无测试预言 (non-oracle)" 问题. 需要注意的是, 这里的 "无测试预言" 和前文所讨论的受环境非确定性影响的 CPS 软件测试预言问题并不相同. 在受环境非确定性影响的 CPS 软件测试预言问题中, CPS 软件的正确行为是可被定义、描述的, 只是由于环境非确定性阻碍了对于软件行为的观察, 使得软件工程师难以准确地描述 CPS 软件的正确行为. 而在机器学习软件测试中, 软件工程师难以用形式化的方法严格定义何为机器学习软件的正确行为. 例如, 对于一个典型的基于分类的行为识别机器学习软件, 由于机器学习算法的分类的准确率天然低于 100%, 因此当该机器学习软件出现识别错误时, 软件工程师将无法判断该错误是由于软件失效导致的, 还是由于算法本身局限性所导致的.

为了应对无测试预言问题, 在机器学习的实际使用中, 通常采用人工标注 + 随机采样数据的方法用于判断机器学习软件的行为是否正确 [49]. 如 Google 在测试其无人驾驶汽车时就同时采用了随机模拟数据和人类监督下的路面实验两种方式 [50].

为了减少人力成本消耗, 有研究者提出可以使用差分测试的方法来测试机器学习程序. 差分测试是指通过使用与被测试程序具备相同功能, 但是不同编码的程序的行为作为交叉验证准则, 用以判断被测试程序的行为是否存在异常. 差分测试被广泛应用于测试面临无测试预言问题的其他类型软件, 如 C 预言编译器、Java 虚拟机、SSL/TLS 协议认证等 [51]. 在机器学习软件测试中, Pei 等 [48] 提出使用差分测试和梯度下降的方法, 用于快速构造能够让深度神经网络失效的反例数据.

除此之外, 还有研究者提出将蜕变测试用于测试机器学习程序的方法. 与差分测试一样, 蜕变测试亦被广泛应用于测试面临无测试预言问题的软件, 如服务计算、图像处理和网络搜索引擎等 [52]. 蜕变测试通过寻找被测试程序在若干个测试输入下的行为间应满足的蜕变关系, 用于验证被测试程序在一组测试输入下的正确行为. Tian 等 [53] 提出了一组基于模型转化的蜕变关系, 用于系统性地检测使用了深度神经网络的自动驾驶车辆可能存在的异常行为.

虽然差分测试和蜕变测试被证明能够有效测试机器学习软件, 但是这两种方法在实际使用中均存在一定的局限性. 对于差分测试, 要求能够找到与被测试程序功能相同、但实现不同的多版本程序, 用于交叉验证被测试程序的行为正确性. 对于蜕变测试, 要求能够发现被测试程序在一组测试输入下行为间必然满足的蜕变关系, 用于检验被测试程序的行为正确性. 无论是同功能、不同实现的多版本程序, 还是描述程序在不同输入下行为间特性蜕变关系, 都是较难获得的, 亦无自动化工具可以生成.

7.1.2 机器学习模型质量检测

由于机器学习算法有严格的理论模型, 因此不少研究者从机器学习的模型出发, 对机器学习软件的质量加以检验. 其中, 大量的研究者关注如何寻找机器学习模型的反例. Kurakin 等 [54] 提出了一种将机器学习模型的线性行为映射到高维空间中的方法, 用于寻找机器学习模型可能存在的反例. Nguyen 等 [55] 给出了一个对抗性算法, 用于生成人类无法识别其差别, 但是机器学习模型将其判断为内容不同的图片. Fredrikson 等 [56] 提出了两个机器学习模型攻击方法, 分别从决策树和神经网络中, 通过使用对抗性数据的方法, 泄露出相关的加密信息.

除了上述以寻找机器学习模型反例为目的的对抗性测试方法, 还有一些研究者关注如何通过模型验证的方法检测机器学习模型是否满足不同的安全性质. Katz 等 [57] 提出了一种基于单纯型法的验证方法, 用于检测神经网络模型中常使用到的非凸线性修正单元函数是否满足所需的安全性质. Huang 等 [58] 提出了一种基于可满足模理论的自动模型验证框架, 用于检测前向反馈多层神经网络的安全性. Pulina 等 [59] 提出了一种基于线性算数约束的 Boolean 型组合的验证方法, 用于检测神经网络的安全性.

7.2 SynEva 方法概述

本小节介绍一种不涉及人工参与、不涉及多版本程序和蜕变关系的, 基于程序生成的机器学习软件环境适应性评估方法 (synthesis-based evaluation, SynEva) [15], 可以自动地评判给定的机器学习软件从其训练环境中提取出的知识能否适应于在预测环境中的使用. SynEva 根据给定的机器学习软件从其训练环境中提取出的知识, 构造出一个类似于测试预言的结构 (称为镜像). 和机器学习软件提出的知识 (称为知识) 相比, 镜像能够在训练环境的预测中提供和知识几乎完全一样的效果, 而在预测环境中却无类似的保障. 因此当镜像和知识在预测环境中具有相类似的效果时, 则可认为机器学习软件的知识能够适应于新的环境, 反之, 则意味着原知识不适应于新的环境.

图 5 描述了 SynEva 方法的执行过程. 具体而言, 其执行可分为 3 个阶段, 即 (1) 准备阶段, (2) 生成阶段和 (3) 度量阶段. 首先在准备阶段中, 机器学习程序从训练环境中提取出了知识, 并使用一个驱动程序将其封装以便于在预测环境中的使用, 启动程序和知识的组合 (称为知识程序). 随后在生成阶段中, SynEva 方法通过归纳式程序生成, 根据知识程序在训练环境中的执行, 构造出能够以较高的精确度模仿知识程序的镜像. 最后在度量阶段中, SynEva 通过比较镜像和知识程序在预测环境上的预测行为的异同, 产生用于评估知识程序对预测环境适应能力的评估值.

SynEva 通过归纳式程序生成的方法构造一个镜像. 镜像与被测试的机器学习软件的知识程序在训练环境中的预测行为需要极为相似. 一个机器学习软件的知识程序包含了从训练环境中提取出的知识信息. 为了模仿知识程序的预测行为, 镜像拥有和被测试机器学习软件的知识程序完全相同的结构, 镜像程序尽可能地模仿知识程序在训练集上进行预测的行为. 在构造镜像的过程中, SynEva 首先对知识程序在预测过程中涉及到结构进行复制; 然后对知识程序中各个节点预测逻辑, 并将其应用到镜像节点中的预测逻辑学习器, 以模仿知识程序的预测行为. SynEva 使用机器学习中的回归算法, 基于知识程序的节点在训练集上的预测行为, 合成镜像程序节点的预测逻辑.

基于知识程序和 SynEva 构造的镜像在预测环境中的预测行为, SynEva 方法量化地给出被测试机器学习软件在预测环境中的适应度. 对于一个预测数据实例, SynEva 首先计算知识程序和镜像对于该实例的预测行为的相似度. 该相似度的计算主要考虑了预测过程中知识程序和镜像分别使用了哪些边, 以及这些边被激活的顺序. 该相似度的值域为 [0, 1] 的实数. 当该值接近 1 时, 则表明被测试机器

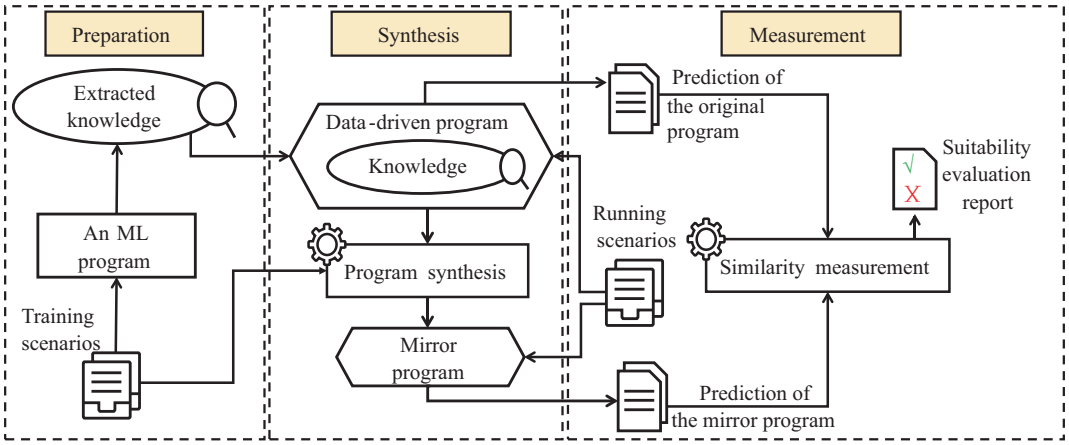
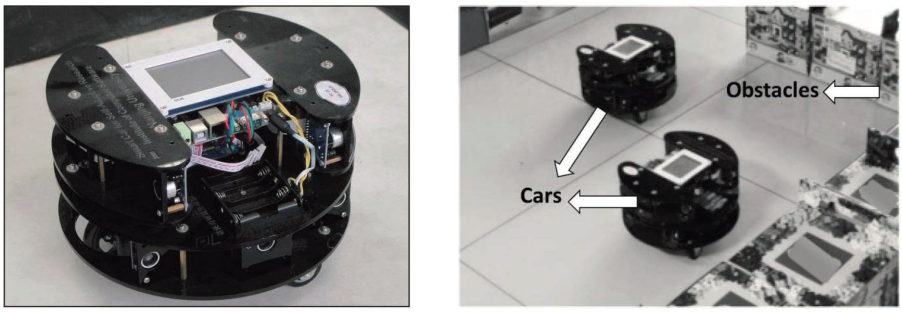


图 5 (网络版彩图) SynEva: 基于程序生成的机器学习软件环境适应性评估概览

Figure 5 (Color online) Overview of SynEva: program-synthesis-based ML programs evaluation



(a) (b)

图 6 (网络版彩图) (a) 自适应自控小车及 (b) 其运行环境

Figure 6 (Color online) (a) Robot car and (b) its running environment

学习软件在训练环境中提取的知识适应于当前的预测环境, 预示着被测试机器学习软件在预测环境中的性能能够接近其在训练环境中的性能. 当该值接近 0 时, 则表明被测试机器学习软件在训练环境中提取的知识不适应于当前的预测环境, 预示着被测试机器学习软件在预测环境中的性能将与在训练环境中的性能存在较大的差距.

8 基于自适应自控小车的实验评估

本节通过一个自适应自动小车 (Robot-car, 如图 6 所示) [11] 程序, 初步验证本文介绍的 SIT 方法、CoMID 方法, 以及 SynEva 方法在测试受环境非确定性影响的 CPS 软件时的有效性. 实验意在回答的研究问题如下:

- **RQ1.** SIT 方法为 CPS 软件生成的测试输入在暴露程序失效行为的有效性如何.
- **RQ2.** CoMID 方法为 CPS 软件生成的测试预言在检测程序运行异常状态的有效性如何.
- **RQ3.** SynEva 方法在评估 CPS 软件中的机器学习模块对环境适应程度的有效性如何.

对于 RQ1, 实验选取了基于随机测试的测试输入生成方法 (RT) 和基于动态符号执行的测试输入生成方法 (DSE) 作为比较方法. 对于 RQ2, 实验选取了朴素的不变式生成方法 (naïve, 基于 Daikon 不

变式生成引擎实现) 和基于程序上下文的不变式生成方法 (p-context, 基于 ZoomIn 方法 [39] 实现). 对于 RQ3, 实验采用了一个面向神经网络特化的 SynEva 方法 [60].

8.1 实验准备与设计

实验对象为一个控制自适应自控小车在室内环境内进行探索的 CPS 软件. 该程序由作者所在单位的多名研究者开发. 该程序能够使用超声波传感器和摄像头对周围环境进行感知, 并以此调节自身行为 (运行方向、速度等), 实现避免撞击、绕过障碍物、绘制室内环境地图等功能.

RQ1. 使用了变异测试工具 MuJava [61] 为对象程序生成了 45 个可执行的变异体. 由于 MuJava 是针对一般 Java 程序设计的, 实验暂不考虑 CPS 软件和一般软件对于变异操作敏感性的差别. 这些变异体为所评估的测试方法提供程序失效现象. 同时实验准备了 4 种不同的环境配置 (如改变环境的边界、环境中障碍物的数量和未知等). 将一个被测试程序或其变异体 P 和一个初始配置 C 的组合称为一个测试实例 (P, C). 实验一共使用了 180 个测试实例.

RQ1 使用被比较方法所生成的测试输入能够发现的程序失效行为的数量来评价测试输入生成方法的有效性. 在对象程序中内置人工编写的断言作为其测试预言. 当这些断言被违反时, 即意味着正在运行的程序出现了失效. 断言主要检测了所控制的小车是否会与障碍物产生碰撞, 以及小车是否离周围的障碍物过远 (影响小车对于环境地图的绘制). 一个被测试方法发现的失效也被定义为一个测试实例在给定时间限制内发现的被测试程序中测试断言的违反.

RQ2. 使用实现收集的对象程序执行序列, 离线地进行被比较方法检测 CPS 软件异常行为的实验. 实验同样使用内置的人工编写的断言来判断收集到的对象程序的执行时否为安全序列 (不安全序列即包含了异常行为). 实验总共收集了对象程序在不同环境配置下的 100 条执行序列, 其中包括 73 条安全序列和 27 条不安全序列. 3 种比较方法 (CoMID, naïve 和 p-context) 先使用收集到的部分安全序列, 生成各自的用于检测程序异常状态的不变式, 随后再对不变式生成中未使用到的安全序列和所有不安全序列进行异常状态检测. 实验采用 10 倍交叉验证的方法以平衡不变式生成所使用的安全序列对于方法有效性的影响.

RQ2 使用检测异常行为时的准确率和误报率来评价测试预言生成方法的有效性. 准确性 (true positive, TP) 是指被正确检测出异常状态的不安全执行序列占全部受检测不安全执行序列的比例. 误报率 (false positive, FP) 是指被错误检测出异常状态的安全执行序列占全部受检测安全执行序列的比例.

RQ3. 主要针对对象程序中用于识别周围信息的图像识别模块. 实验使用了 Udacity 无人驾驶汽车角度预测比赛的数据集[4)] 作为图像识别模块的预测和训练数据. 该数据集包含 107010 张汽车挡风玻璃后的摄像头捕捉到的图像, 每张图像均标注有其对应的方向盘角度数据. 采用人工标注的方法, 以光线明暗为标准, 为以下 4 个不同环境挑选出各自符合的图片: (1) 清晰: 即道路清晰可见; (2) 阴暗: 即场景阴暗, 基本看不清道路; (3) 阴影: 即路上阴影较为明显; (4) 阳光直射: 即光线强时阳光直射镜头, 导致画面非常亮或出现反光看不清道路.

表 1 给出了分类后的图片集情况, 以及用于训练和测试的数据集大小. 基于平衡各个数据集结果的考虑, 同时设计了一个均衡的测试集, 用于验证 SynEva 方法在评估对象程序与环境适应性时的效果.

RQ3 使用 SynEva 方法报告的环境适应程度与实际的环境适应情况两者间的准确率评价 SynEva 方法的有效性. 首先基于 4 个不同的场景, 训练被测试图像识别程序. 训练好的程序即被认为适应于

4) Udacity-challenge. https://github.com/udacity/self-driving-car.

表 1 RQ3 中图片的环境分类情况

Table 1 Environment classification for image sets in RQ3

Environment	Total images	Images in the training set	Images in the testing set	Images in the balanced testing
Clear	16703	15033	1670	194
Dark	4353	3918	435	194
Shadow	3514	3163	351	194
Sunlight	1948	1754	194	194

表 2 SIT, RT 和 DSE 发现的失效个数的比较

Table 2 Comparison of detected bugs for SIT, RT and DSE

Testing approach	Detected bugs (percentage of test instances)
SIT	154 (85.6%)
RT	65 (36.1%)
DSE	112 (62.2%)

训练场景所处的分类, 而不适应于其他分类. 随后会在不同场景的测试集上运行训练好的程序, 并用 SynEva 方法评估其对于当前场景的适应情况.

8.2 实验结果

8.2.1 RQ1: SIT 方法的有效性

表 2 比较了 SIT, RT 和 DSE 3 种方法在 180 个测试实例上发现的程序失效的数量. 其结果表明 SIT 方法能够发现的程序失效的数量是最多的, 相比于 RT 提高了 49.5%, 相比于 DSE 提高了 23.4%. 和预期的一样, DSE 方法能够比 RT 方法发现更多的失效数量, 这是因为 DSE 方法对于实验对象程序空间的探索更为系统和有效. 由于 RT 采用的随机测试方法难以有效地探索对象程序巨大的输入空间, 因此 SIT 方法相比于 RT 的领先是可以理解的. 对于 DSE 方法, 其在发现失效行为方面的效果不如 SIT, 主要原因有两点: 首先, DSE 方法需要借助约束求解器 “计算出” 待探索的测试输入值, 而在 CPS 软件常常会使用到底层平台的诸多库函数, 对于这些库函数的调用会影响到约束求解过程的有效性. 其次, CPS 软件的失效现象往往 “埋藏得很深”, 如在被测试的小车程序中, 通常需要经过 15 次程序 – 环境交互迭代后, 程序的失效现象才会出现. 而 DSE 方法在处理长程序序列时的效率是十分低下的, 往往需要很长的时间才能计算出符合要求的测试输入值. 而 SIT 方法在处理长程序序列时仍采用采样的方式, 其时间效率并不会如 DSE 一般出现指数爆炸的情况, 因此能够在有限的测试时间内发现更多的程序失效现象.

综合上述实验结果, 对实验问题 RQ1 的回答如下: SIT 方法能够有效地测试 CPS 软件. 与已有方法 (RT 和 DSE) 相比, SIT 方法能够发现更多数量的被测试程序的失效现象.

8.2.2 RQ2: CoMID 方法的有效性

表 3 给出了 3 种实验方法所生成的不变式及其在检测实验对象异常状态时有效性的概览. 实验结果表明, CoMID 方法在在检测实验对象异常状态时获得了更高的准确率和更低的误报率. 例如, CoMID 获得的准确率比 naïve 高 16.7%, 比 p-context 高 10.2%, 同时, CoMID 的误报率比 naïve 低 18.7%, 比 p-context 低 5.8%. 更高的准确率意味着 CoMID 能够捕捉到实验对象的执行过程中出现的更多的异

表 3　3 种实验方法生成的不变式及其异常状态检测效果概览

Table 3　Overview of the generated invariants by the three approaches

CoMID		naïve		p-context	
TP (%)	FP (%)	TP (%)	FP (%)	TP (%)	FP (%)
78.3	15.6	61.6	34.3	68.1	21.4

表 4　SynEva 方法在不同环境下的适应性预测准确率

Table 4　SynEva's prediction on ML program's suitability to different envrionments

Training environment	Prediction accuracy (%)	Average prediction accuracy (%)
Clear	72.8	78.0
Dark	88.0	
Shadow	76.1	
Sunlight	75.0	

常状态, 而更低的误报率则意味着对于异常状态捕获数量的提升并不是以牺牲检测精度为代价的.

综合上述实验结果, 对本小节实验研究问题的回答如下: CoMID 方法能够有效地生成不变式以检测 CPS 软件在执行过程中的异常状态. 相比于 naïve 和 p-context 两种方法, CoMID 方法检测的准确率更高, 误报率更低.

8.2.3　RQ3: SynEva 方法的有效性

表 4 给出了在不同环境下, SynEvaD 方法对于对象程序的环境适应能力的评估结果. 由于 SynEva 可采用多种算法学习为其镜像程序构造预测逻辑, 在实验中分别使用了基于线性回归算法和基于线性支持向量回归算法的 SynEva, 并以其平均值为方法报告的预测准确率值. 实验结果表明, SynEva 能够有效鉴别机器学习软件对于不同环境的适应能力, 其对于预测场景适应情况的平均预测准确率达到了 78% (72.8%∼88.0%). 我们注意到 SynEva 方法对于训练于阴暗场景中机器学习程序的场景适应性的评估准确率要显著优于训练于其他场景中的机器学习程序, 这可能是由阴暗场景和其他 3 个场景的差别导致的.

综合上述实验结果, 对本小节实验研究问题的回答如下: SynEva 方法能够准确预测机器学习程序对于环境的适应能力. 对于不同的场景, SynEva 方法的环境适应性预测准确率达到了 78% (72.8%∼88.0%).

9　讨论与展望

信息物理融合系统是云计算、大数据技术和 “人 – 机 – 物” 三元融合模式背景下新型软件基础设施的重要一环. 为了处理不断涌现的新环境、新模式和新平台, CPS 软件必须具备有一定质量保障水平的可成长和可持续演化能力. 然而 CPS 软件在与环境交互过程中所必然面对的环境非确定性给 CPS 软件的质量保障带来了新的挑战. 本文通过分析环境非确定性对于 CPS 软件的影响, 提出了迭代式的研究框架, 分别从寻找被测试软件的错误执行, 以及整体评估被测试软件的运行质量两个方面对 CPS 软件进行测试. 基于这一研究框架, 本文综述了已有 CPS 软件测试工作, 并介绍了 3 项技术, 即基于白盒采样的 CPS 软件测试输入生成技术 SIT、基于多不变式的 CPS 软件测试预言生成技术 CoMID, 以及基于程序自动生成的机器学习软件环境适应性评估技术 SynEva. 基于一个自适应自控

小车的示例程序, 验证了 3 种技术在测试受环境非确定性影响的 CPS 软件时的有效性.

纵观本文所述的 CPS 软件测试技术, 并对照当前各种平台 (如无人飞行器、自适应自控小车、人形机器人等) 上的 CPS 软件实践, 可以发现, 一方面在新环境、新模式和新平台的引领下, 各种新的测试方法、技术层出不穷, 以其为质量保障手段的某些先驱性应用已开始融入人们的日常生活; 另一方面, 对于环境非确定性影响下的 CPS 软件行为和质量保障的认识尚不够全面, 使得各项研究常局限于 CPS 软件的特定层面, 技术成果难以有效支撑大型、复杂 CPS 软件 (如自动驾驶系统) 的质量保障. 我们乐观地认为这种状况说明新一代的 CPS 软件测试方法正在孕育之中, 其最终的成功还需要进一步的研究工作.

在测试输入方面, 目前的主要问题在于现有测试方法难以有效支持在实际运行环境中的 CPS 软件测试. 对于运行于信息和物理空间中的 CPS 软件, 由于缺乏观察其内部运行状态以及控制其外部输入数据的手段, 因此多数 CPS 软件测试工作均局限于实验室场景或是模拟场景. 这使得其测试结果往往难以保障实际运行的 CPS 软件质量. 解决这一问题的一个途径是设计并实现一个高可观察性、高可操控性的测试基础设施. 在该测试基础设施的设计中, 需要综合考虑 CPS 软件实际运行平台和环境的物理特性, 使得其产生的环境数据能够有效还原 CPS 软件实际运行所处的信息物理融合环境.

在测试预言方面, 已有工作通常只考虑环境和程序状态对于程序正常行为的表征, 而不考虑人类活动对于程序行为的影响. 人类活动能够独立于 CPS 软件之外而对环境产生影响, 这对描述、判断 CPS 软件的正常行为又提出了更高的挑战. 通过考虑受人类活动所带来的干扰, CPS 软件的测试预言能够更加贴近于被测试软件实际的运行场景.

在环境适应性评估方面, 目前较为成功工作多集中于数据驱动软件算法设计层面的质量评估, 而忽视了对于这类软件在代码实现层面的质量评估问题. 事实上, 对于数据驱动软件的算法设计质量的保障应由人工智能研究者集中关注, 而软件工程和软件测试研究者则更应关注于对这类软件在代码实现过程中的质量保障问题. 对于上述两个因素的区分和细化, 能够使得人工智能和软件工程研究者专注于自己的擅长领域, 通力协作以共同提高 CPS 软件中质量.

参考文献

1 Lü J, Ma X X, Tao X P, et al. A survey of Internetware: research and progress. Sci China Ser E-Inf Sci, 2006, 36: 1037–1080 [吕建, 马晓星, 陶先平, 等. 网构软件的研究与进展. 中国科学 E 辑: 信息科学, 2006, 36: 1037–1080]

2 Lü J, Ma X X, Huang Y, et al. Internetware: a shift of software paradigm. In: Proceedings of the 1st Asia-Pacific Symposium on Internetware (Internetware 2009), Beijing, 2009

3 Mei H, Huang G, Xie T. Internetware: a software paradigm for internet computing. Computer, 2012, 45: 26–31

4 Dobson S, Denazis S, Fernández A, et al. A survey of autonomic communications. ACM Trans Auton Adapt Syst, 2006, 1: 223–259

5 Zhang J, Cheng B H. Model-based development of dynamically adaptive software. In: Proceedings of the 28th International Conference on Software Engineering, Shanghai, 2006. 371–380

6 Sama M, Elbaum S, Raimondi F, et al. Context-aware adaptive applications: fault patterns and their automated identification. IEEE Trans Softw Eng, 2010, 36: 644–661

7 Xu C, Cheung S C, Ma X, et al. Adam: identifying defects in context-aware adaptation. J Syst Softw, 2012, 85: 2812–2828

8 Garlan D. Software engineering in an uncertain world. In: Proceedings of FSE/SDP Workshop on Future of Software Engineering Research, 2010. 125–128

9 Elbaum S, Rosenblum D S. Known unknowns: testing in the presence of uncertainty. In: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2014. 833–836

10 Esfahani N, Kouroshafar E, Malek S. Taming uncertainty in self-adaptive software. In: Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, Szeged, 2011.

234–244
11 Yang W H, Xu C, Liu Y P, et al. Verifying self-adaptive applications suffering uncertainty. In: Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering, Vasteras, 2014. 199–210
12 Ramirez A J, Jensen A C, Cheng B H, et al. Automatically exploring how uncertainty impacts behavior of dynamically adaptive systems. In: Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering, 2011. 568–571
13 Qin Y, Xu C, Yu P, et al. SIT: sampling-based interactive testing for self-adaptive apps. J Syst Softw, 2016, 120: 70–88
14 Qin Y, Xie T, Xu C, et al. CoMID: context-based multi-invariant detection for monitoring cyber-physical software. 2018. ArXiv:1807.02282
15 Qin Y, Wang H Y, Xu C, et al. SynEva: evaluating ML programs by mirror program synthesis. In: Proceedings of IEEE International Conference on Software Quality, Reliability and Security, Lisbon, 2018. 171–182
16 National Science Foundation. Cyber-physical systems. 2018. https://www.nsf.gov/pubs/2019/nsf19553/nsf19553.htm
17 Myers G J, Sandler C, Badgett T. The Art of Software Testing. Hoboken: John Wiley and Sons, 2011
18 Tesla. Tesla self-driving car. https://www.tesla.com/
19 Google. Google self-driving car. https://waymo.com/
20 Uber. Uber self-driving car. https://venturebeat.com/2019/04/26/5-companies-are-testing-55-self-driving-cars-in-pittsburgh/
21 Wolpert D H, Macready W G. No free lunch theorems for optimization. IEEE Trans Evol Comput, 1997, 1: 67–82
22 Fredericks E M, DeVries B, Cheng B H. Towards run-time adaptation of test cases for self-adaptive systems in the face of uncertainty. In: Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, Hyderabad, 2014. 17–26
23 Amalfitano D, Fasolino A R, Tramontana P, et al. Considering context events in event-based testing of mobile applications. In: Proceedings of the 6th International Conference on Software Testing, Verification and Validation Workshops, Luxembourg, 2013. 126–133
24 Jang M, Kim J, Sohn J C. Simulation framework for testing context-aware ubiquitous applications. In: Proceedings of the 7th International Conference on Advanced Communication Technology, 2005. 1337–1340
25 Lu H, Chan W K, Tse T H. Testing context-aware middleware-centric programs: a data flow approach and an RFID-based experimentation. In: Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2006. 242–252
26 Lu H, Chan W K, Tse T H. Testing pervasive software in the presence of context inconsistency resolution services. In: Proceedings of the 30th International Conference on Software Engineering, 2008. 61–70
27 Filieri A, Ghezzi C, Tamburrelli G. Run-time efficient probabilistic model checking. In: Proceedings of the 33rd International Conference on Software Engineering, 2011. 341–350
28 Sama M, Rosenblum D S, Wang Z, et al. Model-based fault detection in context-aware adaptive applications. In: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2008. 261–271
29 Wang Z, Elbaum S, Rosenblum D S. Automated generation of context-aware tests. In: Proceedings of the 29th International Conference on Software Engineering, 2007. 406–415
30 Griebe T, Gruhn V. A model-based approach to test automation for context-aware mobile applications. In: Proceedings of the 29th Annual ACM Symposium on Applied Computing, 2014. 420–427
31 Esfahani N, Malek S, Razavi K. GuideArch: guiding the exploration of architectural solution space under uncertainty. In: Proceedings of International Conference on Software Engineering, 2013. 43–52
32 Adamsen C Q, Mezzetti G, Moller A. Systematic execution of android test suites in adverse conditions. In: Proceedings of International Symposium on Software Testing and Analysis, Baltimore, 2015. 83–93
33 Anand S, Naik M, Harrold M J, et al. Automated concolic testing of smartphone apps. In: Proceedings of the 20th International Symposium on the Foundations of Software Engineering, Cary, 2012
34 Matinnejad R, Nejati S, Briand L C, et al. Automated test suite generation for time-continuous simulink models. In: Proceedings of the 38th International Conference on Software Engineering, Austin, 2016. 595–606
35 Barr E T, Harman M, McMinn P, et al. The oracle problem in software testing: a survey. IEEE Trans Softw Eng, 2015, 41: 507–525

36 Tse T H, Yau S S. Testing context-sensitive middleware-based software applications. In: Proceedings of the 28th Annual International Computer Software and Applications Conference, Hong Kong, 2004. 458–466

37 Ernst M D, Perkins J H, Guo P J, et al. The Daikon system for dynamic detection of likely invariants. Sci Comput Program, 2007, 69: 35–45

38 Csallner C, Tillmann N, Smaragdakis Y. DySy: dynamic symbolic execution for invariant inference. In: Proceedings of the 30th International Conference on Software Engineering, Leipzig, 2008. 281–290

39 Pastore F, Mariani L. ZoomIn: discovering failures by detecting wrong assertions. In: Proceedings of the 37th IEEE International Conference on Software Engineering, 2015. 66–76

40 Pacheco C, Ernst M D. Eclat: automatic generation and classification of test inputs. In: Proceedings of European Conference on Object-Oriented Programming, 2005. 504–527

41 Jiang H, Elbaum S, Detweiler C. Reducing failure rates of robotic systems though inferred invariants monitoring. In: Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2013. 1899–1906

42 Aliabadi M R, Kamath A A, Gascon-Samson J, et al. ARTINALI: dynamic invariant detection for cyber-physical system security. In: Proceedings of the 11th Joint Meeting on Foundations of Software Engineering, Paderborn, 2017. 349–361

43 Golombek R, Wrede S, Hanheide M, et al. Online data-driven fault detection for robotic systems. In: Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), San Francisco, 2013. 3011–3016

44 Steinbauer G, Mörth M, Wotawa F. Real-time diagnosis and repair of faults of robot control software. In: Proceedings of Robot Soccer World Cup, 2005. 13–23

45 Gillula J H, Tomlin C J. Reducing conservativeness in safety guarantees by learning disturbances online: iterated guaranteed safe online learning. In: Robotics: Science and Systems. Cambridge: MIT Press, 2013

46 Bojarski M, Del Testa D, Dworakowski D, et al. End to end learning for self-driving cars. 2016. ArXiv:1604.07316

47 Julian K D, Lopez J, Brush J S, et al. Policy compression for aircraft collision avoidance systems. In: Proceedings of the 35th Digital Avionics Systems Conference, 2016

48 Pei K X, Cao Y Z, Yang J F, et al. Deepxplore: automated whitebox testing of deep learning systems. In: Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, 2017

49 Witten I H, Frank E, Hall M A, et al. Data Mining: Practical Machine Learning Tools and Techniques. San Francisco: Morgan Kaufmann, 2016

50 California Department of Motor Vehicles. Autonomous Vehicle Disengagement Reports 2018. https://www.dmv.ca.gov/portal/dmv/detail/vr/autonomous/disengagement_report_2018

51 McKeeman W M. Differential testing for software. Digit Tech J, 1998, 10: 100–107

52 Chen T Y, Cheung S C, Yiu S M. Metamorphic Testing: A New Approach for Generating Next Test Cases. Technical Report HKUST-CS98-01. 1998

53 Tian Y, Pei K, Jana S, et al. DeepTest: automated testing of deep-neural-network-driven autonomous cars. In: Proceedings of the 40th International Conference on Software Engineering, Gothenburg, 2018. 303–314

54 Kurakin A, Goodfellow I, Bengio S. Adversarial examples in the physical world. 2016. ArXiv:1607.02533

55 Nguyen A, Yosinski J, Clune J. Deep neural networks are easily fooled: high confidence predictions for unrecognizable images. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2015. 427–436

56 Fredrikson M, Jha S, Ristenpart T. Model inversion attacks that exploit confidence information and basic countermeasures. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, 2015. 1322–1333

57 Katz G, Barrett C, Dill D L, et al. Reluplex: an efficient SMT solver for verifying deep neural networks. In: Proceedings of International Conference on Computer Aided Verification, 2017. 97–117

58 Huang X, Kwiatkowska M, Wang S, et al. Safety verification of deep neural networks. In: Proceedings of International Conference on Computer Aided Verification, 2017. 3–29

59 Pulina L, Tacchella A. An abstraction-refinement approach to verification of artificial neural networks. In: Proceedings of International Conference on Computer Aided Verification, 2010. 243–257

60 Chen Z Q. Design and implementation of system of deep learning scenario adaptability analysis based on program synthesis. Dissertation for Bachelor Degree. Nanjing: Nanjing University, 2019 [陈紫琦. 基于程序合成的深度学习场景适应性分析系统的设计与实现. 学士学位论文. 南京: 南京大学, 2019]

61 Ma Y S, Offutt J, Kwon Y R. MuJava: an automated class mutation system. Softw Test Verif Reliab, 2005, 15: 97–133

Software testing for cyber-physical systems suffering uncertainty

Yi QIN, Chang XU*, Ziqi CHEN & Jian LÜ

State Key Laboratory for Novel Software Technology, Department of Computer Sicence and Technology, Nanjing University, Nanjing 210093*, China*
* Corresponding author. E-mail: changxu@nju.edu.cn

Abstract Cyber-physical systems (CPS) represent an important part of the software infrastructure in the ternary human-cyber-physical universe. In this new era, the CPS software should be continually adapting and evolving. Such constantly-growing CPS software is capable of handling emerging software environments, developing models, and executing platforms. However, environmental uncertainty poses challenges to CPS testing. In this article, the challenges of CPS testing caused by environmental uncertainty are studied. A comprehensive analysis of the impact of environmental uncertainty on CPS testing is conducted, and a research framework for effective and efficient testing CPS is proposed. Based on the proposed framework, the state-of-the-art testing CPS software is discussed, and three testing techniques that address environmental uncertainty are introduced, including a test input generation approach (SIT), a test oracle generation approach (CoMID), and an environmental suitability evaluation approach (SynEva). The experiments are conducted using an illustrative self-adaptive robot car, and the obtained experimental results show that the three proposed approaches can provide effective CPS testing.

Keywords cyber-physical systems, software testing, environmental uncertainty, test input generation, test oracle generation, environmental suitability evaluation

Yi QIN received his doctoral degree in computer science and technology from Nanjing University, China. He is an assistant professor at the State Key Laboratory for Novel Software Technology and Department of Computer Science and Technology, Nanjing University. His research interests include software testing and adaptive software systems.

Chang XU received his doctoral degree in computer science and engineering from the Hong Kong University of Science and Technology, Hong Kong, China. He is a full professor at the State Key Laboratory for Novel Software Technology and Department of Computer Science and Technology, Nanjing University. His research interests include big data software engineering, intelligent software testing and analysis, and adaptive and autonomous software systems.

Ziqi CHEN received her bachelor's degree in computer science and technology from Nanjing University, China. She is currently pursuing her master's degree at the Department of Computer Science and Technology at Nanjing University, China. Her research interests include software testing and program analysis.

Jian LÜ received his doctoral degree in computer science and technology from Nanjing University, China. He is the director of the State Key Laboratory for Novel Software Technology. He is also a full professor at the Department of Computer Science and Technology at Nanjing University. He has been a vice-chairman of the China Computer Federation since 2011. His research interests include software methodologies, automated software engineering, software agents, and middleware systems.