文章编号: 1001-4543(2017)01-0049-09 DOI: 10.19570/j.cnki.jsspu.2017.01.009

基于过程模拟树的 Web 软件一致性检测方法

李丽萍, 王 娜, 唐 姗

(上海第二工业大学计算机与信息工程学院, 上海 201209)

摘 要: 随着大数据、云计算的发展, Web 软件越来越复杂, 人们对其质量要求也越来越严格。研究了一种 Web 软件一致性检测的方法。针对 Web 软件的独特性质, 研究了 Web 软件创建可执行模型的方法, 动态模拟系统运行。引入文法的概念, 设计一个模拟器, 在模拟过程中动态检测可执行模型在外部事件 (文法描述的) 触发下的执行是否存在不一致现象, 整个过程以模拟树的方式呈现。该方法能在项目初期检测出模型的设计与需求的不一致, 在一定程度上保证模型的一致性。

关键词: Web 软件; 模拟树; 活序列图; 上下文无关文法; 一致性检测

中图分类号: TP3-0 **文献标志码**: A

0 引言

软件测试是保证软件质量和提高软件可靠性的一种最主要的手段。软件测试的核心问题是如何在系统无限的输入状态空间中选择最有效的测试用例以满足测试需求 [1]。Web 软件已成为国计民生、商业领域以及未来软件开发的主导软件。当前流行并具有广阔运用前景的 “云应用” 也是 Web 应用。Web 应用迅猛发展的同时, 也使人们对其质量和安全性等要求变得越来越严格。随着大数据、云计算的发展, Web 软件规模越来越大, 形态也越来越复杂, 如何保证这些 Web 软件的质量, 倍受人们的关注。但是 Web 软件内在的动态性、异构性、复杂性、交互性、组成成分和链接的多样性等使得对 Web 软件的建模和测试都十分困难。

另外, Web 软件面对的是全球范围的庞大用户群, 出于抢占市场的需要, 通常采用快速应用开发方式。鉴于目前的软件测试主要还是依赖测试工程师的直觉和经验, Web 软件的测试被认为是一个耗时的、代价昂贵的过程。许多 Web 软件在没有进行充分测试的情况下就投入运营, 质量难以保证, 投产后错误频发。据统计, 目前国内 80% Web 软件的功能测试都是手工进行的, 带有较大的盲目性和不确定性。因此, 迫切需要开发高效的自动验证和测试 Web 软件的新理论、新方法。正确、合理地实施自动化测试, 能够快速、彻底地对 Web 软件进行测试, 从而保证软件质量, 缩短产品发布周期, 节省项目经费。

现有的 Web 应用测试主要有功能测试、性能测试、可用性、兼容性以及安全性方面的测试。目前比较流行的基于模型的测试是根据被测系统的分析设计模型及其派生模型 (一般称其为测试模型) 产生测试用例 [2], 属于功能测试。

但是以往基于模型的测试中, 前提大多是假设模型是正确的, 如何确认模型的正确性, 如何在软件开发初期对系统模型进行有效地模拟和验证, 一直是业界所关注的重点。目前模拟技术主要用于硬件测试, 基于模型模拟对 Web 软件进行一致性检验及测试的工作还比较少。基于模型的 Web 应用测试中所创建的测试模型大多不可以动态地模拟系统运行, 而且很多方法对模型也没有进行一致性检验。静态建模方法对于描述软件执行动作和动态交互方面时, 特别是例如 Web 软件这种交互行为比较复杂的系统时, 表达出来的结果往往错综复杂, 难以阅读和理解 [3]。如果模型本身有错误的话, 那么基于模

收稿日期: 2016-07-21

通信作者: 李丽萍 (1976–), 女, 湖南嘉禾人, 副教授, 博士, 主要研究方向为软件工程、软件测试、形式化方法。
E-mail: liliping@sspu.edu.cn。

基金项目: 国家自然科学基金 (61272036), 上海第二工业大学校重点学科建设项目 (XXKZD1604), 上海第二工业大学校青年项目资助

型生成的测试用例也是错误的。这样生成的测试用例不一定是有效的测试用例。

本文重点研究 Web 软件可执行模型的构造、模型的一致性检验。从构造 Web 软件的可执行模型出发, 用文法表示外部触发事件, 模拟系统的运行, 在模拟系统执行时检测模型是否存在一致性问题; 整个过程以模拟树的形式呈现。若一致将生成状态迁移系统, 再结合测试准则可以生成测试用例, 这些将是下一步研究的内容。

1 相关工作

1.1 基于模型的 Web 软件测试

目前基于模型的 Web 应用测试比较常用的方法是用有限状态机、实体 - 关系图、决策表或 UML 对其进行结构和行为进行建模 [4-13], 然后研究其测试方法。Kung 和 Liu 等 [4-5] 为 Web 应用创建了一个包含对象模型、动态模型和结构模型的形式化测试模型, 提出了一种基于对象关系图、对象状态图和网页导航图等多模型的 Web 应用建模与测试生成方法。Conallen [8] 扩展 UML 对 Web 应用的体系结构进行了建模。文献 [7-8] 中采用 Statecharts 对 Web 导航、Web 元素以及这些元素之间的交互进行了建模。Offutt 等 [9] 分析了构成 Web 应用的网页和软件构件之间的 8 种连接关系, 提出了一种基于有限状态机 (FSM) 的 Web 应用建模和测试用例生成方法。意大利的 Ricca [10] 提出用决策表来对 Web 应用中每个页面的行为进行建模, 然后利用基于决策表的测试准则产生测试案例。缪淮扣等 [12] 设计并实现了一个基于模型的 Web 应用测试系统, 用 FSM 构造 Web 应用的测试模型, 集成了模型转换器、测试目标分析器、测试序列生成器、FSM 和测试序列可视化以及 Web 应用测试执行引擎等工具。Hamideh 等 [13] 提出了一个基于结构模型的 Web 应用测试方法, 先用结构模型表达 Web 应用的静态特征, 接着使用本体和映射工具自动生成测试用例揭示 Web 应用的动态特征。刘晓强等 [14] 提出了一种基于云计算自动生成 Web 应用系统功能测试的并行测试用例的方法, 该方法使用场景流图和脚本录制的并行测试脚本生成和基于搜索的测试数据生成, 产生自动化测试用例集。

这些研究对 Web 应用的形式建模和基于模型的测试有较大的指导作用。但是, 每一种方法都有不同的目的, 所关注的 Web 特性也不一样。随着云计算、大数据的蓬勃发展, Web 软件越来越复杂, 过去的研究有的已经不能适应新的需求和变化, 因而急需探讨 Web 软件的新的测试方法。

1.2 有关活序列图的研究

本文拟用活序列图 (Live Sequence Charts, LSCs) 结合形式化方法为 Web 软件创建可执行模型, 模拟系统的运行。LSCs 是一种基于场景可视化的规约语言, 已成功应用于许多软硬件系统的建模 [21-22]。LSCs 构建的模型的优势是可以动态地模拟系统的执行, 然而用它描述系统规约的一个最大缺点是规约之间可能存在不一致性, 特别是对于包含了许多 LSCs 的复杂系统。文献 [15-20] 中使用模型检验 (model-checking) 的方法将 LSCs 转换为等价的时序逻辑以检测 LSCs 之间的一致性。吴宏 [18] 使用时序逻辑表达 LSC 模型, 研究了基于 LSC 的模型检验。戴雨婷等 [20] 用 LSC 描述基于场景的需求规格说明, 为了能够验证设计模型是否满足需求规格说明, 抽取 LSC 中描述的待验证的 CTL 性质, 与由设计模型生成的 SMV 程序合并到模型检验工具进行验证。Sibay 等 [21] 提出了一种基于场景的语言来支持 LSCs 的全局图和存在图以便更好地建模基于条件的场景。针对一般的标签迁移系统 (Labeled Transition System，LTS) 不能表达既有全局图又有存在图的场景, 他们提出了一种模型迁移系统 (Model Transition System, MTS), 将形式模型转换为 MTS, 然后从 MTS 可以抽取出精化的行为场景。Guo 等 [21-22] 提出了用 LSCs 为交互系统创建可执行模型, 用上下文无关文法 (Context Free Grammar) 表示外部触发事件, 检测模型的运行在外部事件的触发下是否存在一致性问题。他们用一种 PLAY-tree 模拟系统的整个运行过程。模拟结果最终将产生状态迁移系统, 然后基于状态迁移系统生成测试用例。目前, 工作主要还停留在前半部分, 基于状态迁移系统生成测试用例还在摸索阶段。由于他们的研究并不是针对 Web 软件, 没有考虑到 Web 软件的特点, 因而本文参考他们的工作研究了 Web 软件模型的一致性检验及测试用例生成。从构造 Web 软件的可执行模型出发, 对其测试理论和方法展开研究。

2　相关知识点

2.1　上下文无关文法

根据软件工程的观点, 文法既是一种用于规定语言结构的 “规约”, 又是一种能够被语法分析程序自动生成器所识别和转换的 “程序”。文法的句子生成在软件测试数据自动生成方面一直有着广泛的应用 [23-28]。选用合适的编译器能将精化后的符号文法直接转换为测试脚本, 便于测试用例的自动执行。本文基于需求规约用上下文无关文法 (Context-Free Grammar, CFG) 表示外部触发事件, 检测模型在外部事件触发下的执行是否符合定义的需求。

定义 1　上下文无关文法 G 是一个四元组 (V_N, V_T, W, P), 其中:

(1) V_N 是一个非终结符有限集, 每个元素 $v \in V_T$ 被称为一个非终结符或者一个变量, 通常用大写字母表示。

(2) V_T 是一个包含所有终结符的有限集, 终结符通常用小写字母表示, 在本文对应一系列外部事件。

(3) W 是一个开始变量, 用于表示整个系统的开始, 它必须是 V 的一个成员。

(4) P 是一个产生式 (规则) 的有限集合, 每个产生式的形式是 $A \to \alpha$, 其中 $A \in V_N$, $\alpha \in (V_T \cup V_N)$*。

本文扩展了上下文无关文法, 用其表示系统的外部触发事件。

2.2　活序列图

传统活序列图 (LSCs) [29] 是一种可视化的基于场景的规约语言, 非常适合于描述如 Web 软件这样的交互式系统。用 LSCs 为系统建模的好处在于模型可以动态地模拟系统的运行。LSCs 是基于 MSCs (Message Sequence Charts) 的扩展, 也是 UML 顺序图的变形。但是比起后两者来, LSCs 表达力更强而且语义更丰富 [17,29]。LSCs 在扩展 MSCs 的基础上, 主要增加了能表达临时行为的功能, 它可区分系统的强制性行为、可能性行为和禁止行为。

LSCs 主要有 2 种场景图: 全局图和存在图, 其中全局图适用于系统的所有运行, 存在图通常描述系统至少有一次运行所能满足的场景。全局图语义精确地描述系统的强制性行为。全局图包含一个前置图 (prechart) 和一个主图 (main chart), 图 1 所示为一个全局图 [20], 其中虚线六边形表示前置图, 下面的实线矩形框是主图。如果前置图中的条件满足的话, 那么系统将一定满足或运行主图中定义的场景。该强制性行为在系统运行时能明确地告诉用户什么是期望的行为。存在图通常描述系统至少有一次运行所能满足的场景 (没有前置图)。

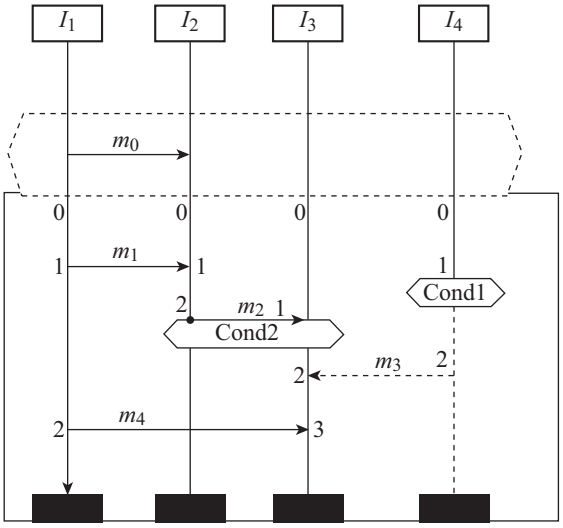

图 1　LSC 全局图

Fig. 1　Universal chart of LSC

LSCs 的对象/实例变量用一个带生命线的矩形框表示, 生命线从上到下表示事件发生的先后顺序。生命线上的数字, 表示事件的发生或者条件, 称为位置。根据位置号, 可以得到事件发生的先后次序。在图 1 中有 4 个对象: I_1, I_2, I_3, I_4, 对象 I_1 有 3 个位置点 $\langle I_1, 0\rangle$, $\langle I_1, 1\rangle$, $\langle I_1, 2\rangle$。对象之间的交互用带箭头的消息表示, 箭头表示方向。条件用六边形表示, 图 1 有 2 个判定条件 Cond1 和 Cond2。本文假设在 LSC 中的所有消息都是同步消息 (满箭头)。LSC 还有许多其他重要的特性, 如 cut, temperatures 等。温度值可取两个值 hot 和 cold, hot 表示元素必须发生, cold 表示元素也许发生。LSC 通过给消息、条件和位置点分配温度值来描述它们的强制性程度。例如, 对象 I_1 在位置 l_1 发送消息 m_1 给 I_2, 它在 l_2 接收到消息, 如果消息和位置都标识为 hot, 那意味着 I_1 必须发送消息 m_1, 消息必须到达并且 I_2 必须接受该消息。如果消息和位置 l_1 被标识为 hot, 但是位置 l_2 被标识为 cold, 那意味着 I_1 必须发送消息 m_1, 消息必须到达, 但是 I_2 可以决定是否接受该消息。对于 l_1、m_1 和 l_2, 可以有 8 种不同的组合方式。有关 LSC 的详细介绍可以参考文献 [29]。

3 模型构造

针对以往基于模型的 Web 软件测试创建的测试模型不可以动态执行, 而且没有进行一致性检验的问题, 本文主要研究 Web 软件可执行模型的构造、基于文法的句子生成、模型模拟时一致性检验、测试准则的设计、测试用例的生成与测试集的优化。整个项目的技术路线图如图 2 所示。本文重点研究如何为 Web 软件创建可执行模型, 模型模拟时一致性检验。测试准则的设计、测试用例的生成及优化等是后续研究的内容。

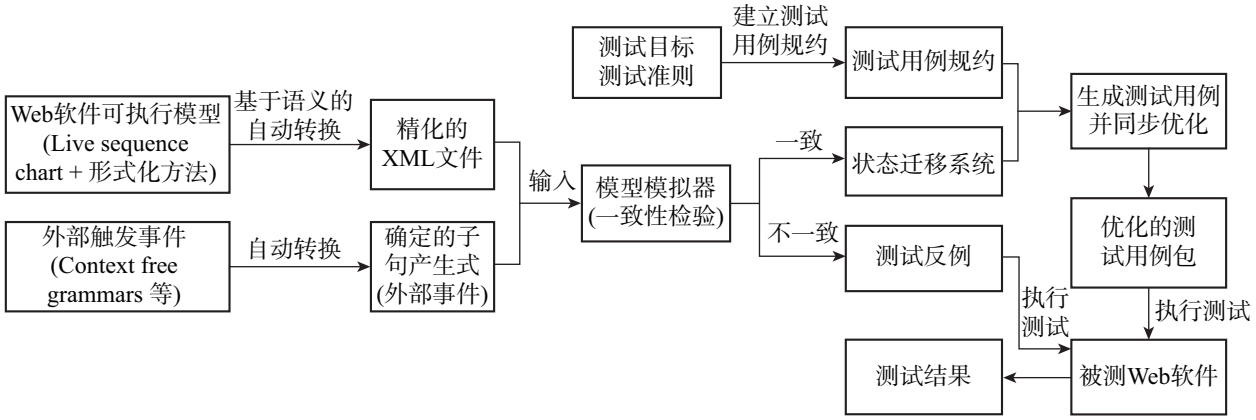

图 2 Web 软件测试技术路线图

Fig. 2 The Roadmap of Web software testing

本文研究的具体内容包括:

(1) 可执行模型的构造, 研究如何将可视化建模与形式化相结合的方法为 Web 软件创建可执行模型, 使其在外部事件的触发下能够模拟系统的运行。因为 LSCs 的表达能力和形式语义比 UML 更加丰富, 所以选用 LSCs 为 Web 软件构建可执行模型。

(2) 考虑如何基于需求规约用上下文无关文法 (CFG) 表示外部触发事件, 解析其递归特性, 研究基于文法的句子生成算法。主要采用了基于文法覆盖准则的生成算法, 每次有选择性地选取产生式来替换非终端符, 使得文法的每个产生式至少被用到 1 次, 句子的生成采用编译原理的最左推导, 即从开始符号出发, 总是选择每个句子的最左非终结符进行替换。

(3) 同时将可执行模型 (LSCs 构建的模型) 和外部触发事件 (文法模型) 输入到模拟器, 研究进行基于场景的模型一致性检验的方法。边模拟边进行一致性检验。

本文基于 Web 的机票预订系统 (Web-based Flight Reservation System, WFRS) 为例来展示可执行模型的建立, 主要考虑 LSCs 的全局图模型。WFRS 的需求模型图如图 3 所示, 包括客户 (Customer)、代理 (Agency) 和航空公司 (Airline) 3 个对象。Customer 是外部对象, 用挥手的云表示, 它将外部或者用户的输入发送给系统。Agency 和 Airline 都有一个状态变量 conf, conf 有 3 个取值 {false, true, abort}, 分别表示订单是否已经启动, 确认或者终止。本系统有 2 种类型的机票: 可退票和不可退票, 用参数 fNo 表示。fNo = 0 表示购买的是不可退换的机票, fNo = 1 表示购买的是可退换的机票。如果旅客购买的不是打折票, 并且航空公司还没有确认该订单, 顾客可以退票, 否则, 顾客不能退票。

图 3 所描述的 WFRS 包含几个不同的场景。图 3(a) 、(b)、(c) 描述了成功购票的场景行为。图 3(a) 显示了一个顾客想要订票, 顾客发送订票消息给机票代理商 (Agency) , 代理商将自己的状态变量 conf 设置为 false, 即 Agency.conf = false, 并且发送订购信息给航空公司 (Airline) , 等待航空公司的回复; 图 3(b) 显示了航空公司接到了订票请求, 也将自己的状态变量 conf 设置为 false, 即 Airline.conf = false, 将确认消息发送回代理商。图 3(c) 显示了代理商收到航空公司发的确认信息, 将状态变量 conf 修改为 true, 即 Airline.conf = true。图 3(d)、(e)、(f) 描述了取消订单的场景。图 3(d) 描述了客户想取消订单, 如果这时订单还没有被确认, 即状态变量 conf 的值为 false, 则取消信息将发送到对象 Airline; 否则, 订单将不能取消。如果状态变量 conf 的值为 abort, 则将订单取消信息发给顾客。图 3(e) 描述了如果航空

公司收到一个取消订单请求, 同时状态变量 conf 的值仍为 false 并且所定机票不是打折票, 则修改状态变量 conf 的值为 abort, 并且发送取消确认消息给对象 Agency。图 3(f) 描述了航空公司 Airline 发送取消拒绝消息给代理商 Agency, Agency 将该消息发送给客户; 图 3(g) 描述了航空公司 Airline 发送取消确认消息给代理商 Agency, Agency 将该消息发送给客户。因为本文采用的是 LSCs 的全局图模型, 所以只要前置图中的条件满足的话, 则系统将一定满足或运行主图中定义的场景。

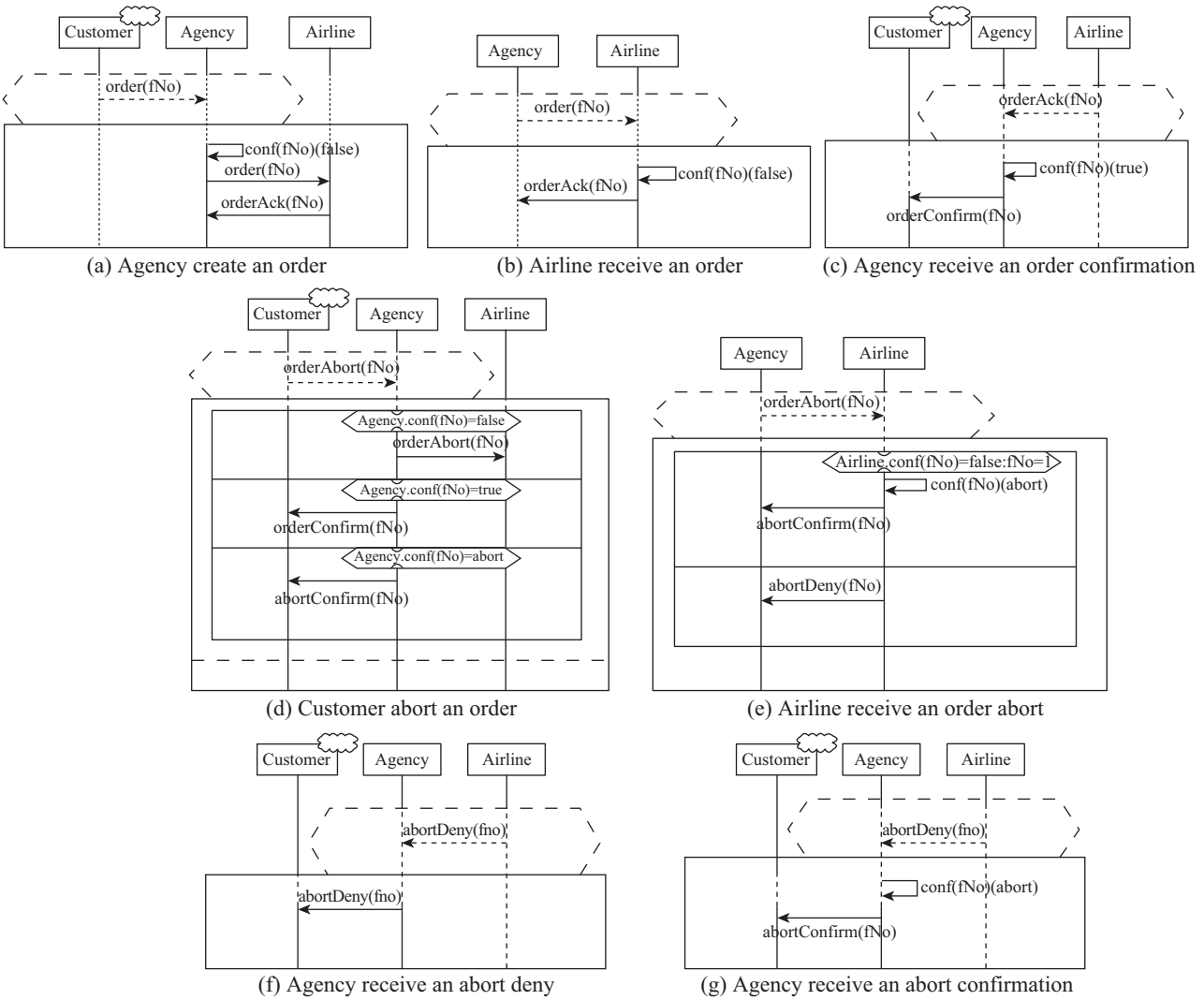

图 3　基于 Web 的机票预订系统

Fig. 3　Web-based flight reservation system

为了构造 Web 软件的可执行模型, 本文从整体功能层面和交互行为层面为 LSCs 添加了形式化的操作语义。

定义 2　(Ls 的操作语义) LSCs 描述的规约 Ls 的操作语义可以被定义为一个迁移系统 $Ls = (S, s_0, E, \Delta)$。

(1) S 是系统一系列可能的对象状态集合, Web 应用的页面和构件都可以看成实例对象。状态 $s \in S$ 可被定义为(RL, ϕ), 其中 RL 是一系列 LSCs 当前正在运行状态的拷贝版, ϕ 用 True 或者 False 表示这个状态是否冲突。

(2) $s_0 = (\phi, \text{False})$ 是系统的初始对象状态, 表示 Web 应用的主页。

(3) E 是一系列系统事件, 包括外部事件、系统对象之间的内部消息, 或者定义在 LSCs 中的隐藏事件, 对应于 Web 应用中的各种链接关系, 如 link, redirect, call, build 等。

(4) 函数 $\Delta \subseteq S \times (\varepsilon \cup E) \times S$ 是一组迁移集合。给定当前状态 $s \in S$ 和一个系统事件 $e \in E$, 迁移 $\Delta(s, e)$ 将返回一个更新的状态, 表示 Web 页和软件

构件之间的关系。

4 模型一致性的检验

基于文法和符号执行的测试数据生成是目前非常流行的测试方法 [23-28]。Godefroid 等 [23] 利用符号文法作为测试输入, 提高了基于搜索的路径覆盖度, 发现了系统更多的错误。万勇兵等 [24] 提出了一种符号化执行的实时系统一致性测试生成方法, 用符号值代替具体的数值, 充分体现了符号化模型的优势, 避免了由于数据枚举所造成的空间爆炸。本文扩展了上下文无关文法, 用符号值代替具体的数值。

本文基于系统需求用扩展的上下文无关文法 (CFG) 设计外部触发事件; 在模拟器中输入 Web 软件的 LSCs 模型和触发事件, 检测模型在外部事件触发下的执行是否符合定义的需求, 即是否存在设计与需求不一致的问题 [30]。用模拟树展示了整个模拟过程, 模拟树的一个分支对应于模型运行的一个场景。在外部事件 (文法描述) 的触发下检测模型的执行是否满足定义运行时的性质。若满足, 表示 Web 软件的设计模型与需求规约一致, 模拟结果可以生成状态迁移系统; 若不满足, 表示存在不一致问题, 可以生成测试反例。

对于本文采用的实例, 基于 Web 的机票预订系统 (WFRS), 其符号文法 $G = (V_N, V_T, W, P)$ 表示如下: 非终结符集 $\boldsymbol{V}_N = \{W, X, Y, A\}$; 终结符集 $\boldsymbol{V}_T$ = {order(fNo), orderConfirm(fNo), orderAbort(fNo)}; $W \in V_N$ 是一个开始变量; P 是一组产生式规则, 定义如下:

$W \rightarrow XW|\varepsilon$

$X \rightarrow$ order(fNo) Y

$Y \rightarrow$ orderConfirm(fNo)

$Y \rightarrow$ orderAbort(fNo)A

$A \rightarrow$ abortDeny(fNo) | abortConfirm(fNo)

fNo$\rightarrow$ 0 |1

其中, "|" 是一个可选操作符。比如, 产生式规则 $W \rightarrow XW|\ \varepsilon$ 表示 $W \rightarrow XW$ 和 $W \rightarrow \varepsilon$, 其中 "$\varepsilon$" 代表空字。

将 Web 软件的可执行模型 Ls 和文法表示的触发事件 G 输入到模拟器, 模拟系统的运行。检测模型运行在外部事件 $w \in L(G)$ 的触发下是否存在一致性问题。整个过程将以模拟树的形式呈现, 可执行模型主要用于动态模拟系统的执行, 一致性检验通过文法句子的解析及可执行模型如何响应外部事件实现。

定义 3 (模拟树) 假定 Ls 是用 LSCs 描述的规约模型, G 是一个扩展的上下文无关文法(Ls, G):

(1) 树中的每一个节点都是一个 ID。

(2) 根节点是 $(Q_0, V_0, \phi,$ False), 其中 Q_0 是对象一系列初始状态的集合, V_0 是 G 中的始元变量, 空集 ϕ 表示在初始状态下 LSCs 还没有运行;

(3) (Q, W, RL, ϕ) 是模拟树的一个节点, 该节点能迁移到它的每个分支的孩子节点, $(Q, eW, RL, \phi) \rightarrow (Q_1, W_1, RL_1, \phi_1)$ 当且仅当以下两个条件之一满足:

① e 是一个外部或者终端事件, 对于 $e \in V_T$, $W \in \{V_T \cup V_N\}^*$, 当条件 $(Q_2, RL_2, \phi_2) \in \Delta$ $((Q_1, RL_1\phi_1), e)$ 满足时, $(Q_1, eW, RL_1, \phi_1) \rightarrow (Q_2, W, RL_2, \phi_2)$ 成立, 称该迁移为终端迁移。

② E 是 V 中的非终端变量, $(Q, EW, RL, \phi) \rightarrow (Q, D_1 \cdots D_n\ W, RL, \phi)$ 成立当且仅当 $E \rightarrow D_1 \cdots D_n$ 满足上下文无关文法 (CFG) 中的产生式规则, 这种迁移为非终端迁移。

(4) 如下的两种节点是叶节点:

① $(Q, W, RL,$ True) 表示一个有冲突的叶节点;

② $(Q, W, RL,$ False) 表示一个成功 (没有冲突) 的叶节点, W 可以取值 ε。

在定义 3 中, 四元组 (Q, W, RL, ϕ) 被引入描述 LSC 模拟器的一个运行状态, 其中 Q 是一系列系统中当前的对象状态, W 是 CFG 句型中未加工部分, RL 是 LSCs 中一系列正在运行的版本, 布尔变量 ϕ True 或者 False 表示该状态是否是一个冲突状态。这里只考虑 LSCs 模拟期间的 super-step 模式和 super-step 状态。在 super-step 模式, LSCs 执行尽可能多的事件, 直到运行的 LSC 到达一个稳定状态, 期间系统除了等待用户的另一个外部事件触发什么都不做。

定义 4 (模型的一致性检查) 一个 Web 软件的设计模型 Ls 与需求规约一致当且仅当模拟树的所有分支都是成功的分支。模拟树的一个分支对应于模型运行的一个场景。在外部事件 (基于规约描述的文法) $L(G)$ 的触发下检测模型的执行是否满足定义的运行时性质。若满足, 表示 Web 软件的设计模型与需求规约一致, 所有分支运行到达成功叶节

点 (ϕ = False); 若不满足, 表示存在不一致性, 运行将到达模拟树的叶节点 (ϕ = True)。

考虑图 3 所描述的在线机票预订系统的一致性测试, 当输入上下文无关文法表示的事件时, 其 LSC 运行的模拟树如图 4 所示。本文采用的是最左解析法, 即在解析过程中始终对最左面的非终结符进行替换。模拟树中实线椭圆表示的状态代表内部节点, 虚线椭圆表示叶节点, ϕ 值为 true 的实线椭圆表示有冲突的叶节点。用 Agency.conf(fNo) 和 Airline.conf(fNo) 的值描述在线机票预订系统的对象状态, fNo = 0 表示购买的机票是不可退换的, fNo = 1 表示购买的是可退换的机票。缩写 Q1∼ Q3 表示模拟树中 3 个不同的对象状态。

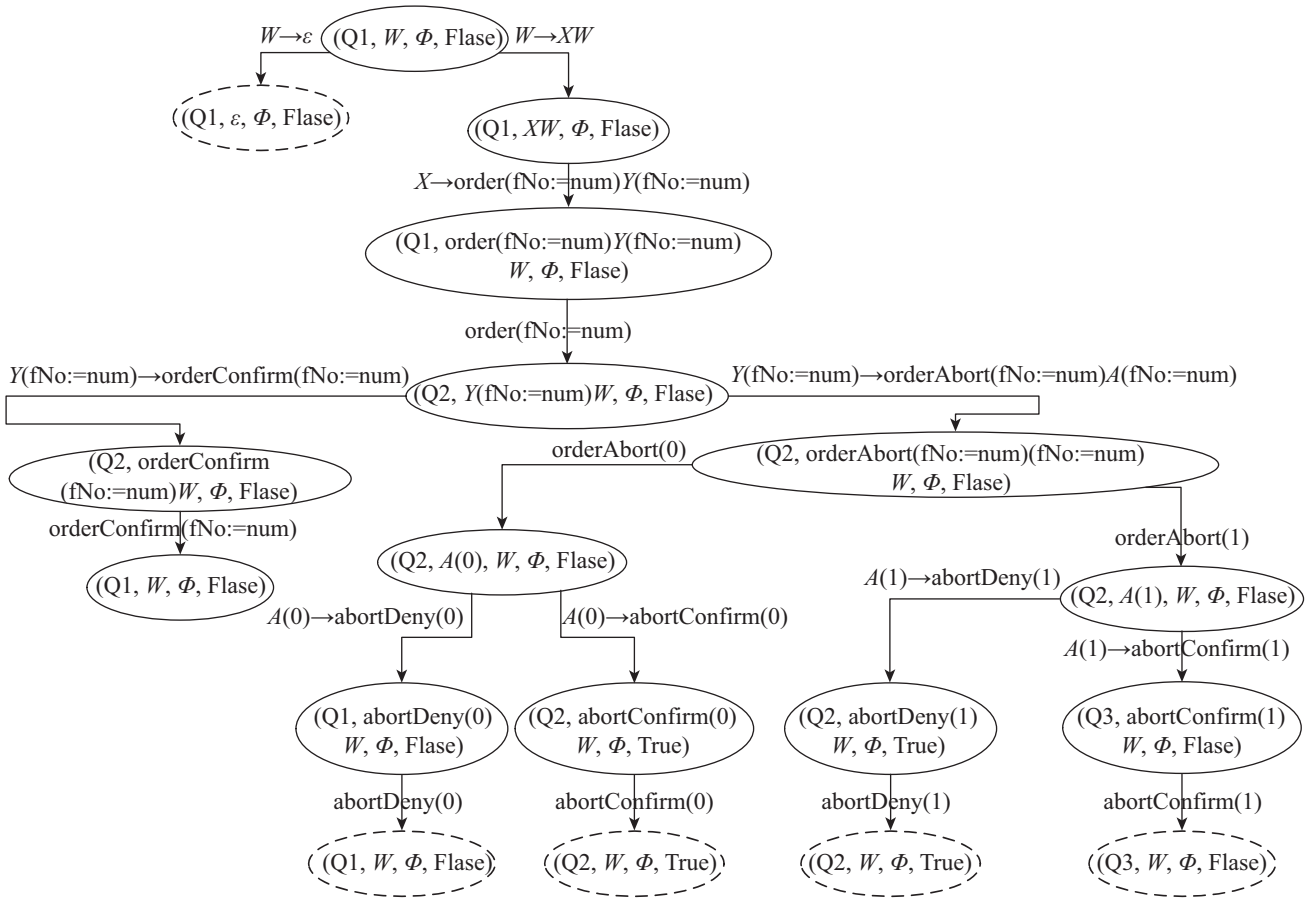

图 4　模拟树

Fig. 4　The simulation tree

Q1: Agency.conf(fNo) = Airline.conf(fNo) = true

Q2: Agency.conf(fNo) = Airline.conf(fNo) = false

Q3: Agency.conf(fNo) = Airline.conf(fNo) = abort

根据需求规定, 如果 Agency.conf(0) = Airline.conf(0) 的值均为 true, 顾客不能退票; 如果此时模拟器收到触发事件 $A(0) \rightarrow$ abortConfirm(0), 那么它将到达一个冲突状态, 模拟树中 ϕ 值为 true 的节点。类似地, 当 Agency.conf(1) = Airline.conf(1) = abort 时, 机票代理商不能拒绝退票, 如果此时事件 $A(1) \rightarrow$ abortDeny(1) 执行, 那么模拟树同样将迁移到一个冲突状态。在 WFRS 例子中, 所有节点的 RL (当前运行的 LSC) 的取值都为 ϕ, 原因有 2 个: ① 初始时没有 LSC 被激活; ② 主要关注 LSCs 模拟期间的 super-step 模式和 super-step 状态。super-step 模式表示当 LSC 接收到一个外部事件并且执行了所有可能的内部事件后, 所有运行的 LSC 将完成执行, 直到收到下一个外部事件触发 [29]。

在模拟树中, 对于每一个有限序列 $W \in L(G)$ 都存在一个对应的分支。LSCs 模型是否与需求规约一致取决于模拟树中所有的分支是否都是成功的分支, 即没有节点的 ϕ 值为真。相反, 如果模拟树中有的分支存在有冲突的节点, 那么表示 LSC 模型与需求规约不一致。

判断 LSCs 模型与需求规约一致是遍历模拟树,

发现树中是否有节点的 ϕ 值被置为 true。在 WFRS 中, 因为模拟树中有 ϕ 值为 true 的节点分支, 因而可以判断它的 LSCs 模型与需求规约不一致。

5 结论与展望

有效测试是保证 Web 软件质量的一种重要途径。本文用活序列图为 Web 软件构造可执行模型, 用扩展的上下文无关文法表示外部事件模拟系统的执行, 在模型模拟的过程中对模型进行了一致性检测, 若模型一致的话将生成状态迁移系统。根据生成的状态迁移系统, 再结合各种测试覆盖准则, 可以生成有效的测试用例。该方法的优点是可在项目初期检测出模型的设计错误, 使得在需求分析和设计阶段发生的错误能在早期被发现, 提前错误检测的进程, 从而可降低开发成本。用文法表示测试用例, 其好处在于将以一种统一的方式表达测试用例。基于一致性模型生成测试用例及测试用例约简将是下一步研究的重点。

参考文献:

[1] GELPERIN D, HETZEL B. The growth of software testing [J]. Communications of the ACM, 1988, 31(6): 687-695.

[2] 颜炯, 王戟, 陈火旺. 基于模型的软件测试综述 [J]. 计算机科学, 2004, 31(2): 184-187.

[3] 李琳, 毋国庆, 黄勃, 等. 基于行为模型的需求可视化研究 [J]. 计算机学报, 2013, 36(6): 1313-1324.

[4] KUNG D C, LIU C H, HSIA P. An object-oriented Web test model for testing Web applications [C]//Proceedings First Asia-Pacific Conference on Quality Software. Washington, DC, USA: IEEE, 2000: 537-542.

[5] LIU C H. A formal object-oriented test model for testing Web applications [D]. Arlington: University of Texas, 2002.

[6] CONALLEN J. Modeling web application architectures with UML [J]. Communications of the ACM, 1992, 42(10): 63-70.

[7] LEUNG K R P H, HUI L C K, YIU S M, et, al. Modeling web navigation by statechart [C]//Proceedings of the 24th International Computer Software and Applications Conference. Washington, DC, USA: IEEE, 2000: 41-47.

[8] OLIVEIRA D M C F, TURINE M A S, MASIERO P C. A statechart based model for hypermedia applications [J]. ACM Transactions on Information Systems, 2001, 19(1): 28-52.

[9] ANDREWS A A, OFFUTT J, ALEXANDER R T. Testing web applications by modeling with FSMs [J] . Software Systems and Modeling, 2005, 4(3): 326-345.

[10] RICCA F, TONELLA P. Analysis and testing of web applications [C]//Proceedings of the 23rd International Conference on Software Engineering. Toronto, Ontario, Canada: IEEE, 2001: 25-34.

[11] UTTING M, PRETSCHNER A, LEGEARD B. A taxonomy of model-based testing approaches [J]. Software Testing Verification and Reliability, 2012, 22(5): 297-312.

[12] 缪淮扣, 陈圣波, 曾红卫. 基于模型的 Web 应用测试 [J]. 计算机学报, 2011, 34(6): 1012-1028.

[13] HAJIABADI H, KAHANI M. An automated model based approach to test web application using ontology [C]//2011 IEEE Conference on Open Systems. New York, NY, USA: IEEE, 2011: 348-353.

[14] 刘晓强, 谢筱梦, 杜明, 等. 面向云测试的并行测试试用例自动生成方法 [J]. 计算机应用, 2015, 35(4): 1159-1163.

[15] BONTEMPS Y, HEYMANS P, KUGLER H. Applying LSCs to the specification of an air traffic control system [C]// Workshop on Scenarios and State Machines: Models, Algorithms and Tools. USA: IEEE, 2003.

[16] BUNKER A, GOPALAKRISHNAN G, SLIND K. Live sequence charts applied to hardware requirements specification and verification: A VCI bus interface model [J]. Software Tools for Technology Transfer, 2005, 7(4): 341-350.

[17] TOBEN T, WESTPHAL B. On the expressive power of LSCs [C]//The 32nd Conf. on Current Trends in Theory and Practice of Computer Science. Prague: SOFSEM, 2006: 33-43.

[18] 吴宏. 基于 LSC 的模型检验研究与实现 [D]. 长沙: 国防科技大学, 2004.

[19] 戴雨婷, 缪淮扣, 梅佳, 等. 基于 LSC 模型检验的性质抽取 [J]. 上海大学学报, 2012, 18(2): 156-162.

[20] SIBAY G E, BRABERMAN V, UCHITEL S, el al. Synthesizing modal transition systems from triggered scenarios [J]. IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, 2013, 39(7): 975-1001.

[21] GUO H F, ZHENG W, SUBRAMANIAM M. L2c2: Logic-based lsc consistency checking [C]//Proceeding PPDP '09 proceeding of 11th International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming. New York, USA: ACM, 2009: 183-194.

[22] GUO H F, ZHENG W, SUBRAMANIAM M. Consistency checking for lsc specification [C]//2009 3rd IEEE International Symposium on Theoretical Aspects of Software Engineering. Washington, DC, USA: IEEE, 2009: 119-126.

[23] GODEFROID P, KIEZUN A, LEVIN M Y. Grammar-

based whitebox fuzzing [C]//PLDI '08 Proceedings of the 29th ACM SIGPLAN Conference on Programming Language Design and Implementation. Tucson, Arizona, USA: ACM, 2008: 206-215

[24] 万勇兵, 徐中伟, 梅萌. 一种符号化执行的实时系统一致性测试生成方法 [J]. 电子学报, 2013, 41(11): 2276-2284.

[25] SOBOTKIEWICZ L P. A new tool for grammar-based test case generation [D]. Melbourne: University of Victoria, 2008.

[26] 郑黎晓, 许智武, 陈海明. 基于文法分支覆盖的短句子生成算法 [J]. 软件学报, 2011, 22(11): 2564-2576.

[27] LIU S Q, LI L P, GUO H F. Generating test cases via model-based simulation [C]//2012 IEEE 13th International Conference on Information Reuse and Integration (IRI). Las Vegas, NV, USA: IEEE, 2012:17-24.

[28] 李虎, 金茂忠, 高仲仪, 等. 上下文无关文法测试充分性 [J]. 北京航空航天大学学报, 2003, 29(10): 869-872.

[29] HAREL D, MARELLY R. Come, let's play: Scenario-based programming using LSCs and the play-engine [M]. Secaucus, NJ, USA: Springer-Verlag, 2003.

[30] LI L P, HONG H G, SHAN T. An executable model and testing for Web software based on live sequence charts [C]//15th IEEE/ACIS International Conference on Computer and Information Science (ICIS 2016). Okayama Japan: IEEE, 2016.

Research on Consistency Checking Method of Web Software Based on Process Simulation Tree

LI Liping,　WANG Na,　TANG Shan

(School of Computer and Information Engineering, Shanghai Polytechnic University, Shanghai 201209, China)

Abstract: Along with the development of big data, cloud computing, Web software has become more and more complexity and people have more strict demand for its quality. The theory and approaches of Web software consistency checking was mainly focused on. Aiming at the unique characteristics of Web software, how to create an executable model to simulate system running was studied. By importing the concept of grammar, a simulator was designed to check if the running of the executable model which triggered by the external events (grammar expressed) was whether consistent or not. The whole process was presented by simulation tree. The method can detect the inconsistent of the model's design and the requirement at the beginning of the project, and guarantee the consistency of the model to a certain extent.

Keywords: Web software; simulation tree; live sequence chart; context-free grammar; consistency checking