$$G!(\{lastStep.state\}\Rightarrow e(event).$$

当比对时发生条件不一致时,由于前提条件不满足,*event* 根本不会执行,故不存在后续状态.为了表达“模型在特定条件下不可执行某个事件”,ProMiner 记为上述 LTL 公式,意为总是不能在 *lastStep.state* 条件下执行 *event*.

- 状态改变不一致

$$G(\{lastStep.state\}\&[event]\Rightarrow X\{SCode.state\}).$$

当比对时发生状态改变不一致时,模型与代码执行事件后的后续状态是不同的.为了表达“模型在执行某个事件后应该与代码执行该事件后的状态改变相一致”,ProMiner 记为上述 LTL 公式,意为总是在 *lastStep.state* 状态下且执行 *event* 后,可推测出下一个状态为 *SCode.state*.

其中,*SCode* 和 *lastStep* 出自上述比对算法的伪代码,*SCode.state* 指代代码在当前 *event* 行后的 *state* 赋值,*lastStep.,state* 指代 *event* 发生前 *state* 的赋值(由于不一致的 *event* 发生前代码与模型仍然是一致的,故上一步的状态两者是相同的).我们未考虑模型在不一致事件发生后的状态,因为生成的 LTL 要表达的是模型“应该改成什么样”而非“之前错在了哪里”.

比对器产生的每个不一致都会如上转换为一个 LTL 公式,作为最终整个表示所有不一致的 LTL 的一个子公式.当多个子公式被生成时,使用&符号将它们连接,可参见表 1 查看生成的 LTL 的示例.但仅仅通过直接转译得到的 LTL 公式由于未经过任何处理,可能导致冗余,影响它的可读性并降低模型检测时的执行效率,故实际使用时必须考虑将其优化.

1.6 LTL的合并优化

ProMiner 通过抽取系统性质反映模型与代码的不一致,但经过上一步骤转译的 LTL 公式在实际使用时往往会冗余,导致在使用模型检测工具时将花费过多的时间.事实上,LTL 模型检测的时间复杂度为 $O(|M|\times 2^f)$[8],其中,M 为模型的状态规模,f 为 LTL 的逻辑长度.显然,指数级时间复杂度在公式规模过大时将导致 ProB 的运行时间过长,因为在生成上述的 3 种 LTL 公式时,由于每个 *state* 是由多个变量构成的,另外,*event* 也可以接受多个参数,每个变量和参数都可以有一定的取值范围,故 LTL 在生成时,生成的 LTL 的规模对应于所有涉及到变量的笛卡尔积.

仍然以热水壶系统为例,假定在代码中 *water_add*/*water_pour* 的行为与模型出现不一致,无论参数取值多少,每次都会注满水或倒光水(为了更好地演示优化算法,这里先设定 *switcher=on* 时也可以倒水),属于上文所述的状态改变不一致.ProMiner 因为记录了每个变量在每次状态迁移后值的变化情况,转译出来的系统性质过长,见表 1.然而,系统中有 3 个变量,而这个不一致仅与 *water_height* 有关,加上 *water_add*/*water_pour* 操作本身要求 *cap=open*,理想状况下,最终生成的 LTL 公式只应包括 *water_height* 和 *cap* 而不包括 *switch*.然而,由于 ProMiner 从最初的测试用例生成开始就会记录下每个变量在每次事件发生后值的变化情况,最终依照上述步骤所生成的 LTL 公式仍然会带有其他实际上与该不一致无关的变量.换言之:与不一致毫无关系的变量 *switch* 的值也会被带入结果中,导致 LTL 子公式的数量加倍,并且每个子公式多带了一个无效变量.

而实际项目中随着变量的增多,无效变量也会随之增多,子公式则相应地以笛卡尔积的规模增加.为解决这一问题,ProMiner 的优化器通过以下 3 个步骤自动合并多个冗余的 LTL 公式并且删去无关的变量.

1. 获取所有变量的取值空间.

通过遍历原始测试用例得到.由于 ProMiner 测试用例生成时未必会覆盖整个模型的取值空间,可能会比原始模型的取值空间要小.

2. 根据 *Event* 将 LTL 子公式分成若干子集.

如前所述,比对器已将不一致分为了 3 类.在此基础上,由于 *Event* 是不一致产生的核心,多个 LTL 子公式可能源于同一个不一致,优化器继续根据每个 LTL 子公式对应的 *Event* 再次分类,遍历每类不一致下所有的 LTL 子公式,然后将 *Event* 名加参数作为键,将这些子公式存于哈希中.在上文 *water_add*/*water_pour* 的例子中,不一致将都被归入状态改变不一致,但分属 *water_add*(1),*water_add*(2),*water_pour*(1),*water_pour*(2)这 4 个不同子集

(当参数为 3 时,并不会发生不一致).

3. 遍历所有上一步骤中不一致 LTL 的子集以得到所有变量的取值空间.

每个未优化的 LTL 中包含的状态中每个变量的值都是有一个取值范围的,它会比步骤 1 的取值空间更小,比如例子中 *cap*∈{*open*},*switch*∈{*on*,*off*},*water_height*∈{0,3},*water_add*/*water_pour* 的参数 *height*∈{1,2}.

4. 在每个子集中,利用遍历法检查其中变量是否满足下述的条件,对满足条件的子公式进行合并并删去无效变量.

   - *State* 中含有某变量取值覆盖其整个取值空间;且若该不一致类型是上述的状态改变不一致,则该变量自身的值在迁移发生前后不发生改变.

这意味着该参数本身并不影响该 LTL 公式所描述的语义.例如,在例子中未经优化得到的 LTL 子式中,*switch* 的取值为{*on*,*off*},而且操作本身并未改变 *switch* 的值,也就是说,*switch* 并未影响该变量,故可以将{*switch*==*on* & *variable*1 & *variable*2 &…& *variable n*}和{*switch*==*off* & *variable*1 & *variable*2 &…& *variable n*}这两个子公式合并为{*variable*1 & *variable*2 &…& *variable n*},将子公式数量减半.同时,将该变量从最终结果中剔除.另外,对条件不一致而言,由于不用考虑变量在 *Event* 后的变化,其实更加简单,直接考察是否有变量的取值覆盖第 1 步中得到的整个取值空间即可;

- 利用表达式简化变量

ProMiner 目前支持赋值操作、取反操作、简单的数学运算等.比如扩展前述的不一致,假定代码中 *water_add* 仍然对参数有效,而 *water_pour* 无论参数为多少每次都会倒光水,从而无论 water_height 为 2 还是 3,在 *water_pour*(1)后,*water_height* 均会变为 0,那么可以将两个 LTL 子式 *G*({*cap*==*open*&*water_height*=2}&[*water_pour*(1)]⇒*X*{*cap*==*open*&*water_height*=0})和 *G*({*cap*==*open*&*water_height*=3}&[*water_pour*(1)]⇒*X*{*cap*==*open*&*water_height*=0})合并为

$$G(\{cap==open\&water\_height\in\{2,3\}\}\&[water\_pour(1)]\Rightarrow X\{cap==open\&water\_height=0\}).$$

值得注意的是:上述的优化过程事实上是“有损”的,因为虽然被舍弃掉的不必要的变量本身与不一致无关,但其实经过优化之后,“该变量与该不一致无关”的语义本身也会从结果的 LTL 中被剔出;而且优化过程本身也是有额外时间消耗的.故:目前 ProMiner 在实际处理 LTL 的生成和优化时,出于效率考虑,会由用户决定是否需要对结果进行优化.

ProMiner 生成的 LTL 始终是以 *event* 为核心的,故在 LTL 的生成/简化操作之后得到的最终结果正反映了代码层中表达 *event*(相当于函数或代码片段)的前后置条件的语义.3 种不一致中,条件不一致反映的是前置条件语义,初始化和状态改变不一致反映的是后置条件语义.而 LTL 简化的本质就是从松散的测试结果抽取的原始 LTL 的基础上,进一步提炼出能够准确表达 *event* 前后置条件语义的 LTL 公式.我们未来的工作中会继续优化该提炼手段,以使之更加精确.这样生成的 LTL 不但可以帮助模型检测,还可以进一步在此基础上挖掘出给用户的一些可能的修改建议.

2 系统原型

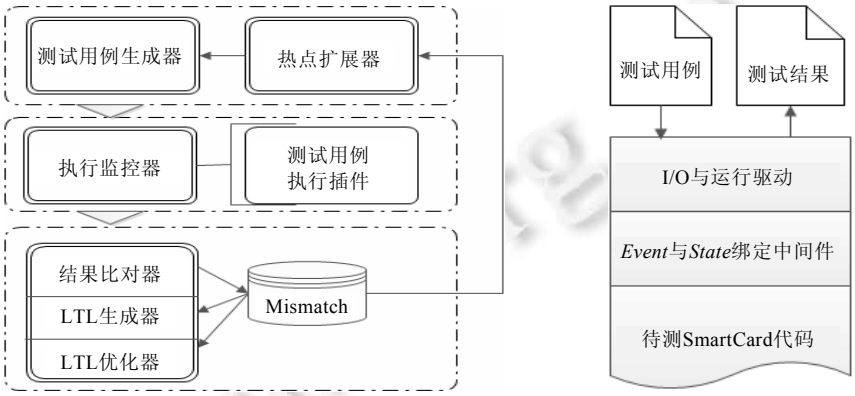
在上一节,我们介绍了 ProMiner 的组件结构以及工作流程,本节将从系统实现的角度介绍其原型,并介绍 ProMiner 在多功能智能卡项目中的应用.

图 5 是 ProMiner 总体的结构图:各个组件与图 1 所示的流程图是相对应的,但实现上并非每个流程都对应单独的一个模块,而是大致分为 3 个相互可独立工作的松散结构,各个组件之间仅通过 xml 文件(测试用例及生成结果)交互以降低耦合性并增加架构的灵活性.

- 测试用例生成器基于上文提到过的开源工具 Event-B MBT[12],输入原始模型并输出测试用例.热点扩展器是测试用例生成器的扩展,如第 1.4 节所述,它将根据结果比对器的工作结果和用户的一些自定义热点通知测试用例生成器增量生成测试用例;
- 执行监控器本身仅作为具体测试用例执行插件的一个容器运作,为其提供测试用例,接受测试结果并

输出测试结果与其他组件交互,具体的执行插件需要根据具体的待测代码编写.其关键是如何将模型中的 *Event* 和 *State* 与代码中的相关函数和变量得到有效的对应,每个 *Event* 都被注册了一个相关函数,而该函数将调用代码中改变系统状态的一些函数;而每个 *State* 中的 *Variable* 则使用一些表达式与代码中某些特定的全局或是重要对象的变量相联系,当代码中变量修改时,将捕捉到这些变化.可参见第 1.2 节;

- 结果比对器、LTL 生成器、LTL 优化器三者在实现中处于同一个模块.它们共用内存中的 Mismatch 对象集合作为数据的存储区.结果比对器主要负责写,LTL 生成器和优化器可根据每次的结果生成相应的 LTL.三者共用同一个模块,主要是出于执行效率的考虑.



(a) ProMiner 的架构 (b) 测试执行插件

Fig.5

图 5

3 实验分析

为分析基于系统性质的一致性检验方法的有效性,我们考察实际软件多功能智能卡系统,将 ProMiner 对其文件系统的模型与代码进行一致性检验.该系统符合 ISO-7816 标准,使用 Event-B 作为建模语言,C++作为编码语言.鉴于项目的安全要求,部分具体功能和实现细节不再详尽描述,部分建模信息可参考文献[14,15].除此之外,为了更好地分析 ProMiner,我们也对以下几个示例模型以及其对应的代码部分进行了属性抽取和模拟实验:上文作为例子的热水壶 Water Boiler 模型以及一个自动电梯 Lift 的模型.

3.1 多功能智能卡系统

在该多功能智能卡系统中,文件系统对其安全性和稳定性起着关键性作用.相对于其他文件系统而言,对用户的操作权限设计方面有更严格的要求.在实际开发环境中,代码编写与形式化建模分属两个不同的团队,工作时间与地点都不一样,很难保证工作流程按照预期进行.建模过程中模型经过 5 层精化,而代码则前后主要共经历了 10 次迭代过程.在整个开发过程中,有效保持模型与代码间的一致性则显得尤为重要.ProMiner 产生的 LTL 能够直接被 ProB 使用,帮助定位模型需要更新的部分,并且用于检验每次修改后的模型是否有效处理了这些不一致部分.由于模型本身在经历 5 层精化后 *event* 及其参数的数量快速增加,导致模型路径以指数级速度增加,使得测试用例生成时间超出可接受范围,因此,我们主要在模型的前两层模型的基础上进行实验.

在测试用例生成过程中,我们选取了默认的全迁移覆盖策略以及用户定义的热点扩展.由于访问权限是智能卡文件系统最重要的关键性质,包含该性质的所有 *event* 都被定义为需要扩展生成测试用例的热点,共产生抽象测试用例 761 条,覆盖了所有与访问权限参数直接相关的迁移.对每条结果,我们都使用了 LTL 优化,最终,这些 LTL 在 ProB 上的运行时间均<1s,并未对系统开发时间造成太大影响.

在上述 10 次代码迭代的过程中,ProMiner 对模型和代码进行不一致检验,共找到 55 个模型与源码发生不一致的情况.例如:

- 初始环境下,用户对文件的访问权限的不一致定义;
- 文件创建的操作结果的不一致,代码中增加了文件创建后的访问权限,使其对某类用户可见;
- 用户进行删除操作时对用户权限判断条件的不一致.

然而,与不断更新的代码相比较,模型最终实际更新部分有 60 处.ProMiner 未找到的 5 处不一致,经检查主要是受限于基于模型的测试方法本身,即,代码中实现了建模时根本没有包括在内的组件.例如:代码实现过程中增加定义了不同的用户访问权限类别,超出了模型对权限变量的取值范围,所以测试用例无法覆盖到这部分代码,进而导致不一致检验的遗漏.我们计划在后续研究中,考虑从代码层面定义需扩展热点,进而生成测试用例以覆盖这部分的不一致.

3.2 模拟实验

除了智能卡系统之外,我们主要为两个不同的示例模型编写了对应的代码,并在其上实践了 ProMiner.如后文表 2 所示,Water Boiler 即我们在上文中作为例子的热水壶模型,这里不再赘述.Lift 是一个简单的电梯系统的模型,其中定义了电梯的升/降、开/关以及相应的用户控制功能.

在模拟实验中,首先根据现有模型实现代码,并首先通过基于模型的测试以保证两者的一致性;然后,为了模拟出现不一致的情况,人为地注入不一致,再通过 ProMiner 检验其一致性并生成 LTL.为了尽量做到客观,人为注入不一致时,我们首先针对每个系统建立变化库,即,能够反映合理需求变化的代码并将若干这样的变化编号保存;其次,每次实验过程中,从变化库中选取上述会产生不一致的变化,以 10%的概率插入原代码.对每组模型与代码,我们都进行 10 次这样的不一致注入.

与智能卡一样,模拟实验中,生成的测试用例数量是恒定的.当然,在实际操作中,由于 ProMiner 可能会需要多次迭代,每次会对模型修改,故测试用例数量可能发生改变,本次实验中尚未遇到这样的情况,我们在未来的工作中会考虑更加复杂的情况及相关的应对方法.

3.3 结 论

通过上述实验,我们得到了如表 2 所示的实验结果.

Table 2 Experimental results

表 2 实验结果

实验系统	测试用例数	不一致总数	找到的不一致
SmartCard	761	60	55
Water Boiler	18	21	18
Lift	56	29	29

在实验中,我们总结出下面几点关于 ProMiner 的问题:

(1) 流程中的时间瓶颈为测试用例生成;

(2) ProMiner 可承受的模型规模与代码行数无关,与模型中 *event* 和参数的数量有关;

(3) 存在 ProMiner 找不到不一致,即代码超越模型定义的部分.

第 1 个问题的产生是因为状态图的构建和路径覆盖问题本身的复杂性;第 2 个问题则是因为模型路径的多少其实是由 *event* 和参数决定的,路径对于最终产生的测试用例数的影响是指数级的,在 *event* 中,无论经历多么复杂的操作,耗费时间仍然只是线性的;对于第 3 个问题,目前我们仍然没有特别好的解决方案,未来将考虑在代码层扩展热点以暴露出模型中根本未定义这一功能.

另外,对前两个问题造成的当模型过大导致时会使测试生成时间过长的问题,我们目前的建议是在模型规模比较庞大时使用比较高的精化层次,或者尽可能地在建模时就将整个系统切分为几个组件部分建模.当然,这样做是有代价的,前者增加了模型的颗粒度,因而无法利用 LTL 来检查精化度较高的模型层次;后者增加了建模

的复杂度.但这么做一方面可以使 ProMiner 的适用性大为增强,另一方面使得工程本身变得更加灵活和易于管理.具体的取舍视实际情况有所不同,我们将在后续的工作中尽可能地优化相关问题.

4 相关工作

基于模型的测试方法的研究目前更多地集中在诸如 UML 这样工业界更为流行的、但并非完全形式化的语言上.例如,李宣东、王林章等人对 UML 活动图提出了基于路径覆盖生成测试用例的方法.ProMiner 目前主要支持形式化语言 Event-B,他们在 UML 这种半形式化/图形化的建模语言中的研究是值得借鉴的.另外,他们在文献[16]中提出了将 UML 状态机语义用于模型检验的方法,这与我们使用 ProB 对 Event-B 进行模型检验是类似的.我们在未来的工作中会考虑在李宣东等人研究的基础上,进一步将 ProMiner 中提出的扩展的一致性检验框架用于 UML 这样被更多开发者使用的建模语言上.

对系统性质的抽取一直是业界的热点问题,但人们往往关注于系统的某个特定方面,而非模型与代码的一致性.例如:微软实验室开发的工具 Tark[17]能分析序列化的数据集合文件,例如系统执行路径和系统日志等,并将其转换为 LTL 的一个子集.该 LTL 更多地是在表达系统状态迁移序列的组合形式.而我们对 LTL 的抽取是在测试用例和其执行结果的基础上完成的,这一点有相似之处,但我们的 LTL 系统性质更加关注于两者间的迁移或迁移条件的不一致所在.

同样来自微软实验室的 Daikon[18]使用动态执行结合对关键变量监控的手段来挖掘系统的不变式.这些不变式本身可以用来作系统分析,应用于代码层本身,如单元测试用例筛选、代码重构检查等.不变式以形如 $x!=0$, $y>x$ 等方式表达.其动态执行过程与我们的测试用例执行本身存在某些相似性,而我们更多的是生成反映不一致的动态性质而非简单的不变式.

Su 和 Gabel 也在系统性质挖掘方面做过一系列研究[19,20].文献[19]介绍了 Deja Vu,可以用来检测不同软件代码版本之间的不一致所在.Su 等人使用了自动化的手段检测一系列自定义的不一致情况是否存在.文献[20]介绍了他们在反向工程中抽取时序逻辑系统性质的方法和相关项目.他们同样用了测试手段来总结 Java 程序中时序化的函数调用性质.其方法同样具有参考价值,但更多的是在代码的版本间对某些特定问题的性质进行抽取,并不主要关注模型与代码的一致性.

5 结束语

作为对传统基于模型的一致性检验方法的扩展,本文提出了一种基于系统性质的一致性检测方法,并开发了其原型框架 ProMiner.在支持传统的基于模型的自动化测试方法的同时,又能在模型需要随实际运行系统而更新的特定条件下,做到对模型之于代码的一致性检测.

ProMiner 一方面是现有的基于模型的测试方法的一个实现,另一方面也对基于模型的测试方法本身做出扩展.ProMiner 的自动化特性可以保证用户以较小的代价来完成基于模型的测试.我们的热点扩展方法可以对基于模型测试中的一些关键点进一步挖掘以产生更多的测试用例,这使得测试过程变得更加可信和有效.

ProMiner 对 LTL 性质的抽取和优化本身是一个独创的方法.ProMiner 将测试用例与测试执行的比对结果分成 3 类并且在该原始 LTL 上进一步优化提取出能反映模型与代码不一致的 LTL 公式.抽取的 LTL 公式除了可以作为后续的模型检测的输入,也可以用于对模型与代码的进一步分析.另外我们也发现:其实,类似的方法对许多诸如对模型精化过程的检测、对一些模型或代码的转换工作等也可以起到很好的效果.在未来的工作中,我们将继续探讨其可能性.另外,目前阶段,ProMiner 仍然处于原型工具阶段,尚有很多不足之处,我们也将在未来的工作中对其本身进行以下几个方面的改进:

(1) 使用更加自动化的热点扩展功能,以优化覆盖效率并减少人为工作;

(2) 引入随机性和副作用概念,以对系统中的随机或是外来要素更好地加以处理;

(3) 如第 2 节最后所述,在生成 LTL 的基础上,对用户的修改方案自动化地提出建议.

致谢 最后,感谢在我们的工作中给予帮助和提供建议的所有同行.

References:

[1] Duan YC, Gu YQ. A multi-dimensional separation of concerns approach for model driven process framework modeling. Ruan Jian Xue Bao/Journal of Software, 2006,17(8):1707−1716 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/17/1707.htm [doi: 10.1360/jos171707]

[2] Schmidt DC. Guest editor's introduction: Model-Driven engineering. Computer, 2006,39(2):25−31. [doi: 10.1109/MC.2006.58]

[3] Dias Neto AC, Subramanyan R, Vieira M, Travassos GH. A survey on model-based testing approaches: A systematic review. In: Proc. of the 1st ACM Int'l Workshop on Empirical Assessment of Software Engineering Languages and Technologies: Held in Conjunction with the 22nd IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE 2007). ACM Press, 2007. 31−36. [doi: 10.1145/1353673.1353681]

[4] Liu H, Ma ZY, Shao WZ. Progress of research on metamodeling. Ruan Jian Xue Bao/Journal of Software, 2008,19(6):1317−1327 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/19/1317.htm [doi: 10.3724/SP.J.1001.2008.01317]

[5] Wang J, Li XD. Preface to special issue on formal methods and tools. Ruan Jian Xue Bao/Journal of Software, 2011,22(6):1121−1122 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4036.htm [doi: 10.3724/SP.J.1001.2011.04036]

[6] Huth M, Ryan M. Logic in Computer Science: Modelling and Reasoning About Systems. Cambridge: Cambridge University Press, 2004. 175−186.

[7] Tao QM, Zhao C, Guo L. Proving soundness of program transformations in optimizing compilation based on temporal logic. Ruan Jian Xue Bao/Journal of Software, 2009,20(8):2074−2086 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3378.htm [doi: 10.3724/SP.J.1001.2009.03378]

[8] Clarke EM, Grumberg O, Peled D. Model Checking. Cambrige: The MIT Press, 1999. 35−50.

[9] Lin HM, Zhang WH. Model checking: Theories, techniques and applications. Acta Electronica Sinica, 2002,30(Z1):1907−1912 (in Chinese with English abstract).

[10] Abrial JR. Modeling in Event-B: System and Software Engineering. Cambrige: Cambridge University Press, 2010. 176−188.

[11] Abrial JR, Butler M, Hallerstede S, Hoang TS, Mehta F, Voisin L. Rodin: An open toolset for modelling and reasoning in Event-B. Int'l Journal on Software Tools for Technology Transfer (STTT), 2010,12(6):447−466. [doi: 10.1007/s10009-010-0145-y]

[12] Dinca I, Ipate F, Mierla L, Stefanescu A. Learn and test for Event-B—A Rodin plugin. In: Proc. of the Abstract State Machines, Alloy, B, VDM, and Z. Berlin, Heidelberg: Springer-Verlag, 2012. 361−364. [doi: 10.1007/978-3-642-30885-7_32]

[13] Leuschel M, Butler M. ProB: A model checker for B. In: Proc. of the FME 2003: Formal Methods. Berlin, Heidelberg: Springer-Verlag, 2003. 855−874. [doi: 10.1007/978-3-540-45236-2_46]

[14] Zhang Y, Guo J, Zhu XR, Wang WJ, Zhu JY, Tang JH, Chen JN. Modeling and development of multi-application smart cards based on Event-B. Computer Engineering and Science, 2014,36(10):1943−1951 (in Chinese with English abstract).

[15] Zhang Y, Guo J, Zhu XR. Application of model-based development method in multi-application smart cards. Netinfo Security, 2013,(156):75−79 (in Chinese with English abstract).

[16] Zhou Y, Zheng GL, Li XD. An operational semantics for UML state machines in model checking context. Acta Electronica Sinica, 2003,31(Z1):2091−2095 (in Chinese with English abstract).

[17] Lo D, Ramalingam G, Ranganath VP, Vaswani K. Mining quantified temporal rules: Formalism, algorithms, and evaluation. In: Proc. of the 16th Working Conf. on Reverse Engineering (WCRE 2009). IEEE, 2009. 62−71. [doi: 10.1109/WCRE.2009.42]

[18] Ernst MD, Perkins JH, Guo PJ, McCamant S, Pacheco C, Tschantz MS, Xiao C. The Daikon system for dynamic detection of likely invariants. Science of Computer Programming, 2007,69(1):35−45.

[19] Murray G, Barton ES. Scalable and systematic detection of buggy inconsistencies in source code. ACM Sigplan Notices, 2010, 45(10):175−190. [doi: 10.1145/1869459.1869475]

[20] Gabel M, Su Z. Testing mined specifications. In: Proc. of the ACM SIGSOFT 20th Int'l Symp. on the Foundations of Software Engineering. ACM Press, 2012. 4. [doi: 10.1145/2393596.2393598]

附中文参考文献:

[1] 段玉聪,顾毓清.多维关注分离的模型驱动过程框架设计方法.软件学报,2006,17(8):1707−1716. http://www.jos.org.cn/1000-9825/17/1707.htm [doi: 10.1360/jos171707]

[4] 刘辉,麻志毅,邵维忠.元建模技术研究进展.软件学报,2008,19(6):1317−1327. http://www.jos.org.cn/1000-9825/19/1317.htm [doi: 10.3724/SP.J.1001.2008.01317]

[5] 王戟,李宣东.形式化方法与工具专刊前言.软件学报,2011,22(6):1121−1122. http://www.jos.org.cn/1000-9825/4036.htm [doi: 10.3724/SP.J.1001.2011.04036]

[7] 陶秋铭,赵琛,郭亮.基于时序逻辑证明编译优化程序变换的保义性.软件学报,2009,20(8):2074−2086. http://www.jos.org.cn/1000-9825/3378.htm [doi: 10.3724/SP.J.1001.2009.03378]

[9] 林惠民,张文辉.模型检测:理论、方法与应用.电子学报,2002,30(Z1):1907−1912.

[14] 章玥,郭建,朱晓冉,王文君,朱晶洋,汤家华,陈峻念.基于 Event-B 方法的多应用智能卡的建模与开发.计算机工程与科学,2014,36(10):1943−1951.

[15] 章玥,郭建,朱晓冉.基于模型的开发方法在多应用智能卡中的应用.信息网络安全,2013,(156):75−79.

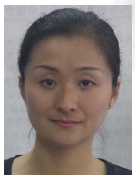
[16] 周颖,郑国梁,李宣东.面向模型检验的 UML 状态机语义.电子学报,2003,31(Z1):2091−2095.

葛徐骏(1989－),男,江苏兴华人,硕士,主要研究领域为软件测试与验证.

郭建(1969－),女,博士,副教授,CCF 会员,主要研究领域为可信计算,模型检查,程序分析与验证.

王玲(1989－),女,硕士,主要研究领域为软件测试与验证.

朱惠彪(1967－),男,博士,教授,博士生导师,CCF 会员,主要研究领域为高可信计算,形式化方法.



徐立华(1979－),女,博士,副教授,CCF 会员,主要研究领域为软件分析与测试,形式化方法.