

上认为,高昂的建库费用和缺乏有效的检索工具是阻碍软件重用的二个因素,但在 Internet 上这两个问题将得到缓和。首先 Internet 上有许多免费的可重用构件库,原先是由某一机构开发 RCL,RC 的分类、归档需花费昂贵的费用,当某一 RC 修改时,需重新分类和归档,这种工作极为繁琐,且易于出错,这也是软件重用难以推广的原因,而现在有许多外部的 RCL 可供使用,因此即使开发机构内部无 RCL 也可进行软件重用。其次对 RCL 的搜索也可以依赖于外部的信息提供者,Internet 上有许多方法来检索可重用构件,当在 Internet 发布了一个需求时,往往会得到许多反馈信息,这些信息往往有助于我们找到可重用的构件,其结果往往比亲自搜索还要好。

一些机构根据 Internet 的特点专门开发一些检索工具,Gertz 等人开发了 Onika 以在 Internet 上搜索软件模块^[45],Onika 能通过 Internet 或 WWW 检索远地站点的 RC,并能把这些 RC 和本地的 RC 集成在一起。Boisvert 用了 GAMS 系统在分布式计算环境,即分布在 Internet 上的多个 RCL 中寻找 RC^[46],作者给出了 GAMS 的实现原理,使用方法等。Poulin 等人认为用形式化的工具和技术来开发 RCL 需要很大的投资,而使用这些 RCL 还需要专门的训练。由于这些原因,要想从 RCL 中获得效益很难,即使这些 RCL 中包含了许多高质量的构件,因此 Poulin 等人采用 HTML 语言和 WWW 上的浏览器 Mosaic 来开发检索 RC 的工具,其开发、维护

费用仅为采用常规方法费用(80—130人年)的1%^[47]。

目前,在 Internet 上搜索可以在短时间内搜索大量的 RCL,其 RCL 的数量是以前单个开发机构的 RCL 数量无法相比的,相差达好几个数量级。当一个系统的参数值的变化达到几个数量级时,该系统的性质往往会发生根本性的变化,Internet 也一样,巨大的网上资源可以保证网上存在我们所需要的构件,但检索这些构件的方式是和传统方法不同的。

当然,由于 Internet 的迅速发展还是近几年的事,有关的网上软件电子交易的法律,商业规范还不健全,因此 RC 的质量、安全等一些问题的保证,一些安全性要求极高的系统还不宜使用这类 RC,但随着 Internet 的逐步成熟,这些问题会逐步解决,可以说,Internet 的迅速发展对于软件重用技术来说既是一个机会也是一个挑战。

结束语 软件重用就好像“站在前人的肩膀上”,能大幅度提高软件生产率、降低开发成本、保证软件质量,是解决软件危机的一个很有效的方法,但目前软件重用还没有在工业界得到全面的推广应用,其原因很多,涉及到管理、心理、经济、社会文化、法律等方面的问题。从技术角度讲,软件的重用面临着通用性与专用性、构件的分类与检索、理解、修改以及其它潜在的问题,今后,领域分析、过程重用、采用软件体系结构的重用方法、大粒度重用、构件程序设计方法学、重用和软件工程的其它领域的相互关系等将会是研究的重点。(参考文献共60篇略)

(上接第126页)

常规的合取式(假设后置断言已经化为 Skolem 范式),每一个合取式再分解成一组类似 Prolog 的 Horn 子句:

$$a_1 \wedge \dots \wedge a_n \rightarrow b_1 \vee \dots \vee b_m$$

分解成:

$$a_1 \wedge \dots \wedge a_n \rightarrow b_i (i=1, \dots, m)$$

然后对各组 Horn 子句进行排列组合,则形成一组类似 Prolog 的程序。如果对于前置条件这组程序中有(至少)一个经过执行后的结果为真,那么我们认为在这样的前置条件下,后置断言为真。在规范中,总是存在特殊的谓词,例如“a is father of b”,所有这种谓词必须在 Where 部分有结构化的定义,否则无法执行。为了更好地求值,我们规定这些谓词定义成类似 Prolog 式子的递归函数。对于问题(b),只要把每次对 PRE-POST 断言的求值看成是一次过程调用,处理好参数值的传递即可。

结论 上述方法不但可以用于逐步求精技术中检查相邻的两步之间的语义一致性,还可以认为是一种基于形式语义的速成原型的新方法,因为在这种方法中,每一规范都可以看成是一个原型。近年来,使用形式语义作为速成原型的基础被认为是可行的方法。我们的方法有如下优点:(1)这种速成原型的方法直接和产生高效的执行程序的过程相联系。(2)通过 PRE-POST 断言,这种处理方式又和验证相联系,这种把验证和速成原型相结合的结果可以看成是把动态语义和静态语义结合在一个统一的形式化框架下的产物。

但是,我们的工作还存在缺陷。对于该方法中的 PRE-POST 断言的处理,依旧存在效率的问题。分解出的 Horn 子句组很多时,对它们进行排列组合是个 NP 问题,导致搜索效率不高。如何高效地搜索解空间是我们下一步要解决的问题。(参考文献略)

125-126,124

可执行规范技术

An Executable Specification Technique

沈孝龙 龚世生

(汕头大学计算机系 汕头515063)

TP311.5

摘要 本文简要介绍了可执行规范技术。可执行规范在软件工程过程的早期就能发现错误,随着形式化规范研究的发展,它越来越受到重视。XYZ系统是唐稚松教授提出并领导实现的,该系统的核心是序列化时序逻辑语言(TTL XYZ/E语言),该语言能在同一框架之中表示动态语义(可执行命令)与静态语义(前后断言规范),并可混合在一程序中出现。用这种混合出现的程序,就能表示出由完全抽象的规范到可有效执行的程序之间平滑过渡的过程,实现该过程必然用到可执行规范技术。

关键词 可执行规范,动态语义,静态语义,XYZ系统

软件工程

一、可执行规范简介

形式化规范在很多领域里,包括要害系统(critical system),被认为是具有价值的,一个规范能否正确描述它对应的需求,得经过验证。但是,众所周知验证一个形式化或非形式化的规范是很困难的。用形式化规范,有很多人对验证作过很有价值的工作,如复查(review),证明和可执行性等,人们对形式化规范的可执行性有所争议,这种争议看上去更像是,能否把程序证明看成是一种验证程序正确性的方法。

Hoare曾对可执行规范的可行性做过一个有趣的评论,这个评论权衡了形式化规范的表达性和可执行性。他以最大公因子(GCD)函数为例,并在完全一阶谓词演算中把它形式化。开始时用与($\wedge$)、或($\vee$)和非($\neg$),表示这个函数。一般来说,所有象这样的规范只能在整个证明空间上进行盲目搜索来执行它。接着对这个规范逐步求精,首先消去了非($\neg$),得到了一个逻辑程序;然后再消去或($\vee$),得到一个函数式程序;最后消去与($\wedge$),取而代之的是顺序式的结构。这样,就得到了命令式的程序。

许多人在利用可执行规范的优点方面做了有益的工作,Knott和Krause及Dick等人为了激活(animating)形式化规范,研究了从Z到Prolog的自动转换。他们把负号(negation)当作有限失败。这意味着他们在一个有限的时间区间内得到失败的信息,

因此也限制了所有类型的子集。例如,整数被限制为一个有限子集。虽然这种限制在理论上对转换的完备性作出了妥协,但是在实际使用这种方法时,并没导致很多问题,用这种方法可以把Hayes的《Specification case studies》中的许多例子转换成可执行的。利用Prolog的可执行性带来的好处(可以用来检查逻辑一致性和不确定性)吸引了许多研究者。同样,把形式化规范转换成函数式语言也是可能的。Johnson和Sanders把Hayes的《Specification case studies》中的一些例子转换成类Miranda的表示法。但是,这些转换都不是自动完成的,因此效率都不太高,亟需提高。

对于上述看法,Hayes和Jones主张规范是不(需要)可执行的。他们证明增加任何形式的决定(determinism)到规范里都会导致不良影响,这些影响能导致对适宜规范的程序的限制。

作为直接的反驳,Fuch提出了相反的意见:规范是可执行的。他从Hayes和Jones的文章中举的例子转换成逻辑规范语言,这种语言比Prolog更有表现力。变换的方案不是自动的,他认为,任何有限的规范都可执行,虽然效率很低,但可以在可能的结果空间里完成搜索。据称这样的转换几乎在相同的抽象层次,并且基本上没有改变它们的结构。虽然这种说法遭到广泛的责备,但Fuch设法用一个简明的说明式(declarative)程序实现了Hayes和Jones的举例,这点让人印象非常深刻。Fuchs得出结论:

可执行规范的好处在于可执行的组件比传统的生命周期中的组件更容易得到,并且规范的可执行性提供了与验证类似的检查(inspection)和推理。

Damon 和 Jackson 也做了类似的工作。他们在 Fuch 的工作上继续努力,寻找在有限空间里更快找到解的方法。他们建立了一个叫 Nitpick 的系统,这个系统已经初见成效,实现了发现推导变量、消除同形和剪枝技术,加快了搜索速度。

总的说来,可执行规范有如下可能的用途:增加确信度(confidence);快速地产生原型系统;指导设计等,但是还没有哪种单一的技术能满足规范验证,只有把可执行规范技术同其他规范验证技术结合在一起,才能取得更好的效果。

二、XYZ 系统简介

在冯·诺依曼体系的计算机中,程序由数据定义部分和执行部分组成。前者的语义是静态的,而后的语义是动态的。为将两种语义统一在一个框架之中,计算机科学家和逻辑学家们进行了数十年的努力。在1950年的“Automata Studies”会议上,图灵机和有限自动机被确认为最适合冯·诺依曼计算机的数学模型。显然,这两种自动机都是基于状态转换的。此时程序的动态特征得到了更多的关注。后来,S. C. Kleene 提出了正规表达式这一更数学化的模型,并证明了它与有限自动机的等价性,然而,他试图用一阶逻辑来表达正规式的努力失败了。随着70年代 J. McCarthy 形式化程序动态语义的尝试的失败和指称语义的出现,人们的注意力逐渐转移到程序的静态语义上来。但是,静态语义讨论的是“如何做”,而不是“怎样做”,它与工程师的思维方式不一致,因而难以被工业界接受。1979年,XYZ 系统的创始人唐稚松教授发现,线性时间时序逻辑(LTTL, linear time temporal logic)既可表达动态语义(有限状态转换),又可以表达静态语义(含2前后断言的规范)。基于这种逻辑,他提出了时序逻辑语言 XYZ/E,将动态语义与静态语义统一在同一框架中。XYZ 系统还提供一组基于 XYZ/E 的 CASE 工具,使程序的整个开发过程有了一个统一的逻辑基础。

XYZ/E 中表示状态转换的基本语句是条件原子式或简称条件元(conditional element 或简写为 c. e.):

$$\begin{aligned} LB = y \wedge R \Rightarrow \$O(v_1, \dots, v_k) \\ = (e_1, \dots, e_k) \wedge \$OLB = z \end{aligned} \quad (1)$$

这里“LB”是一个控制变量,表示标号,y 和 z 是标号

值,分别称作定义标号和向前标号。R 是一个 FOL (一阶逻辑)公式,表示转移条件。 $v_1, \dots, v_k$ 是程序变量, $e_1, \dots, e_k$ 是类型与 $v_1, \dots, v_k$ 一一对应的表达式。“\$O”表示时序逻辑算子 NEXT, $\$O(v_1, \dots, v_k) = (e_1, \dots, e_k)$ 等价于 $\$Ov_1 = e_1 \wedge \dots \wedge \$Ov_k = e_k$,表示变量 $v_1, \dots, v_k$ 在下一时刻将具有 $e_1, \dots, e_k$ 在本时刻的值。“ $\Rightarrow$ ”是蕴含算子,表示“if . then”。“ $\rightarrow$ ”也表示蕴含,它们的不同之处在于前者用于指令中而后者用于逻辑表达式中。

条件元还有另一种写法:

$$LB = y \wedge R \Rightarrow \Diamond(Q \wedge LB = z) \quad (2)$$

这里“ $\Diamond$ ”表示时序逻辑算子“eventually”;Q 是一个 FOL 公式,表示动作。R 与 Q 也可以作为过程的前后置断言。这种写法用于表示规范语句。

一组用“;”(表示“与”操作)连接起来的语句和一些规则构成 XYZ 的程序单元,其形式如下:

$$[ce_1; \dots; ce_n] \text{ WHERE } [rl \wedge \dots \wedge rm] \quad (3)$$

如果所有的 $ce_i (i=1, 2, \dots, n)$ 是形式(2)的语句,则称此单元为抽象的程序规范;如果所有的 $ce_i (i=1, 2, \dots, n)$ 是形式(1)的语句,则此单元是一个可执行程序。如果既有形式(1)的语句,又有形式(2)的语句,则该单元是表示抽象程度不同的程序规范,因此 XYZ/E 可以精确地表示从抽象规范到完全可执行程序的逐步求精的整个过程。如何保证求精过程的相邻两步规范或程序之间的语义一致呢?这就是我们的工作。

三、可执行规范在 XYZ 系统中的应用

在程序开发的逐步求精技术中,总是从一个纯粹的抽象规范开始,最后到一个有效的可执行程序,不可避免地会出现抽象规范和可执行语句混合形式的程序。对于这种规范形式的序列,我们必须要有有一种方法保证相邻的两步之间的语义的一致性。否则,这种方法就失去了意义。通常来说,Hoare 逻辑没法验证这种语义的一致性,唯一的方法是用类似测试的方法,通过对它们的求值,看看在确定的输入下,是否输出值也是确定的,由此方法检查它们的一致性。这种规范是抽象部分和可执行命令的混合体。那么,问题就变成了:(a)如何对 PRE-POST 断言求值。(b)如何把这种求值的结果和可执行命令的结果相结合。对于问题(a),我们在唐稚松教授指导下采用如下的解决方法:①把前置条件(pre-condition)看成是 Prolog 中的事实。②把后置断言转换成一组

(下转第124页)