基于多 Agent 的软件测试系统设计

王 伟,刘久富,娄坚波,李金奎
(南京航空航天大学 自动化学院,江苏 南京 210016)

摘 要:随着软件系统变得越来越复杂和庞大,如何对它进行快速有效的测试已经成为现在的一大热点。通过将 Agent 技术引入软件测试过程,设计了一种基于多 Agent 的软件测试系统。该系统由界面 Agent 模块、预处理 Agent 模块、程序插桩 Agent 模块和动态测试信息分析 Agent 模块组成,实现了软件语句和分支覆盖率的测试及查询覆盖率不满足要求时未覆盖目标的情况。有效地解决了传统手工测试程序运行效率低、繁琐等一些问题,保证了软件系统高效稳定的运行。

关键词:软件测试;Agent 模块;词法语法分析;程序插桩;覆盖测试
中图分类号:TP311 **文献标识码**:A **文章编号**:1673-629X(2011)04-0037-03

Design of Software Testing System Based on Multi-Agent

WANG Wei, LIU Jiu-fu, LOU Jian-bo, LI Jin-kui
(College of Automation Engineering, Nanjing University of Areonautics and Astronautics, Nanjing 210016, China)

Abstract: It is a hot topic to test software quickly and effectively as software systems become increasingly complex. Design an intelligent software testing system which brings Agent technology into software testing process. The system consists of interface Agent module, pre-treatment Agent module, program Agent instrumentation module and active testing information analysis Agent module, it mainly realized the testing of the code coverage and branch coverage of the software, besides, it can also inquire the condition of uncoveraged target when the coveragement cannot satisfy the requirement. This method has effectively solved the problems of the traditional manual testing program operation inefficient and verbose and ensured the software system of highly efficient and stable operation.

Key words: software testing; agent modules; iexical and grammar analysis; program instrumentation; coverage test

0 引 言

软件测试是保证软件系统正确性的一个重要手段,也是计算机软件工程方法和技术的一个主要组成部分。软件测试的方法和技术是多种多样的,但最基本最直观最有效的测试方法是覆盖测试。重视测试覆盖率的度量,能够减少测试的盲目性,并引导测试工作朝着提高覆盖率的方向努力,从而找出那些潜伏的程序错误[1]。

Agent 是近年来计算机科学领域中的一个重要概念,基于 Agent 的系统代表了一种新的软件开发方式和范型。Agent 是指驻留在某一环境下能持续、自主地发挥作用,具有自治性、智能性、社交性的计算实体。面向 Agent 的软件测试是一项新的研究方向,由于其自治的特点,在复杂的软件测试工作中逐渐受到重视[2]。

文中设计了一种基于 Agent 技术的软件智能测试系统,该系统将 Agent 的观念运用到测试技术中,通过预处理 Agent、程序插桩 Agent 和动态测试信息分析 Agent 等一系列 Agent 模块帮助测试者对软件的语句和分支覆盖率进行测试。在测试过程中,利用 Agent 的自治特性,在没有其它 Agent 或测试者的直接命令和干预下进行独立的判断,利用 Agent 的智能特性理解测试者的测试目标,利用 Agent 的社交能力与其它的 Agent 进行通信。该多 Agent 软件测试系统能够减少测试者的直接干预,并提供一个良好的集成测试环境,可以大大提高自动化测试的效率[3]。

1 软件测试系统框图

文中将传统的软件测试系统的结构分为三大模块:预处理模块、程序插桩模块和动态测试信息分析模块,其结构框图如图 1 所示。

(1)预处理模块。包括词法分析模块和语法分析模块,主要用来生成源程序的语法树结构。词法分析

收稿日期:2010-09-10;**修回日期**:2010-12-16
基金项目:国家自然科学基金(60674100)
作者简介:王 伟(1986-),男,山西运城人,硕士研究生,研究方向为软件测试技术;刘久富,博士,硕士生导师,研究方向为软件测试技术与软件质量工程。

模块是采用词法分析工具 lex 将被测程序分解成单独的词的表示,形成初步的符号表;语法分析模块则通过接收词法分析的结果,采用 yacc 将输入字符串识别为单词符号流,并生成源程序的语法树结构。

(2)程序插桩模块,包括语法树分析模块和自动插桩模块。其中,语法树分析模块是在语法树的基础上自动分析源程序的关键点,即插桩位置;自动插桩模块则是通过接收语法树分析模块的结果,自动往程序的关键点处插入探针函数即插桩语句,从而在程序运行到插桩点时记录下有关的运行情况[4]。

(3)动态测试信息分析模块,包括覆盖率分析模块和自动查询模块。通过输入测试用例,覆盖率分析模块可通过插桩函数库自动计算出源程序的语句和分支覆盖率;当覆盖率不满足要求时,自动查询模块可自动分析源程序中未覆盖目标的相关情况,并找出到达此目标的路径[5]。

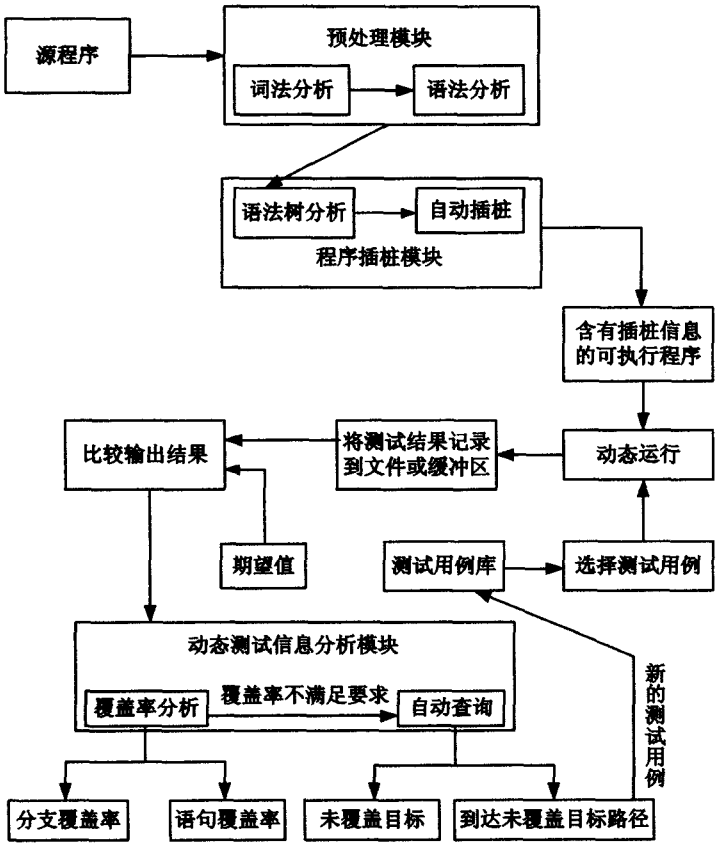

图1　软件测试系统框图

2　基于多 Agent 的软件智能测试系统

2.1　系统模型

基于 Agent 的自治性、智能性、社交性等特点,文中将它应用到软件测试中,提出了一种基于 Agent 的新型软件测试系统模型,该模型的系统结构如图2所示。

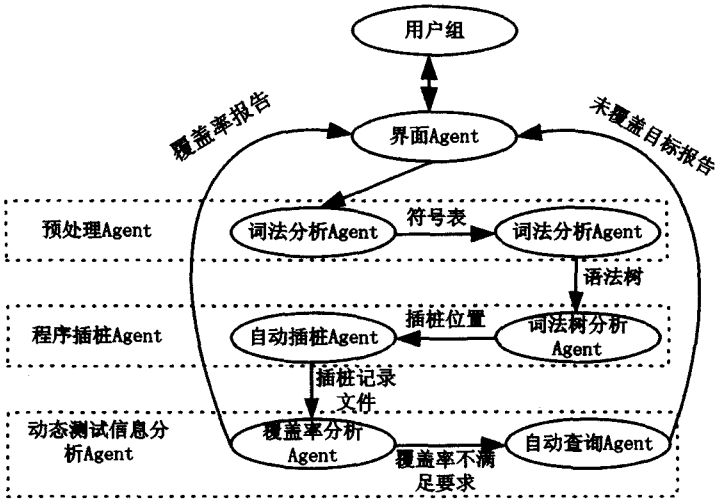

图2　基于多 Agent 的软件智能测试系统架构

2.2　Agent 组织的交互

该多 Agent 系统内部的通信采用了消息机制,其通信内容为一个四元组,即:

<通信内容>::=<发送者><接收者><时间戳><参数数据流>

从程序设计的层次来看,Agent 之间的通信是由事件激发的。当一个事件激发时,一个 Agent 就会向消息队列发送一个消息,消息队列接收到消息,就会根据消息的接收者进行转发。在这个过程中,事件是 Agent 之间通信、同步的唯一途径,通信机制和计算 Agent 内部的执行机制都是基于消息队列的性质,它们是异步的、可靠的、有序的[6]。

2.3　各 Agent 模块功能设计

对应于软件测试系统的结构框图,本多 Agent 软件测试系统模型主要由界面 Agent、预处理 Agent、程序插桩 Agent 和动态测试信息分析 Agent 组成,各模块功能及实现方法如下所述:

2.3.1　界面 Agent

界面 Agent 是整个系统与用户直接交互的部分,作为管理 Agent 组织和发起多个 Agent 的联合行动。本系统中,界面 Agent 主要实现以下三个功能:

首先,界面 Agent 为用户提供了访问软件测试系统的接口;

其次,界面 Agent 可以与服务 Agent 之间进行相互协作,将用户的意图传达给服务 Agent,并将来自服务 Agent 的信息呈现给用户;

最后,界面 Agent 使系统具有用户服务定制的能力,包括界面风格、功能、使用方式等多个方面[7]。

2.3.2　预处理 Agent

预处理 Agent 模块包括词法分析 Agent 和语法分析 Agent,用来生成源程序语法树结构,每一个语法规则对应一个处理函数,并作节点挂在语法树上,提供对外的接口。

1)词法分析 Agent 模块。

词法分析是将源程序代码分解成单独的词的表示,形成初步的符号表。表的结构主要包括标识符表、类型表、关键字表、常数表等,这些信息对于错误的查找和定位都有十分重要的作用。

2)语法分析 Agent 模块。

语法分析主要是将输入字符串识别为单词符号流,对源程序作进一步分析。语法分析的结果是生成语法树,每一个语法规则对应一个相应的处理函数,并作为树的一个节点挂在语法树上,提供对外的接口。

本系统将词法分析 Agent(lex)和语法分析 Agent(yacc)结合起来使用,其实现过程如下(流程如图3所示):

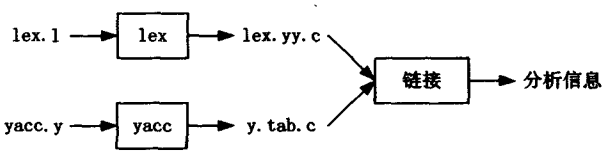

图3　词法、语法分析模块实现流程

(1)写一个给出特定词法规则的文件 lex. l,并由 lex 将其转变成一个通用宿主语言,生成 C 程序 lex. yy. c,其主函数为 yylex();

(2)根据任务需求制定功能说明,即一个语法规则文件 yacc. y,并利用 yacc 将其转换成一个处理相应输入流的程序 yy. tab. c,其主函数是 yyparse();

(3)yyparse()通过调用 yylex(),从输入流中找到"单词",再根据输入结构对这些单词加以组织。当某条规则被匹配,就调用对应于此规则的一个动作[8]。

2.3.3　程序插桩 Agent 模块

本模块包括语法树分析 Agent 和自动插桩 Agent,主要是通过找出源程序的关键点,并向其中插入探针函数,从而获取程序执行时的动态信息。

1)语法树分析 Agent 模块。

语法树分析 Agent 主要是对预处理 Agent 生成的语法树结构进行分析,得到源程序的关键点,并将分析结果传递给自动插桩 Agent。在对语法树进行分析时主要识别如下几类关键点:

(1)If 结构中, then 分支和 else 分支处;

(2)switch 语句的每个分支处;

(3)对于循环结构,考虑两个分支,即空循环和循环一次;

(4)顺序结构中,在程序入口处和出口处、子函数入口处和出口处等。

2)自动插桩 Agent 模块。

本模块采用一次插装的方式,使用探针函数作为插装库函数的调用指针,插装库作为实现函数功能的实体和测试代码的集合。其中,探针函数按如下原则来设计:

(1)语句覆盖探针:计数器插入以块为单位,所谓"块"指逻辑上连续执行相邻最大的语句集。

(2)分支覆盖探针:在分支语句的各个分支插入计数器函数统计各个分支的执行次数。

为了使程序插装更加方便,对所有函数进行编号,建立函数索引表。这样在插装过程中,就可以根据函数 ID 进行选择,从而快速植入探针[9]。

2.3.4　动态测试信息分析 Agent 模块

本模块包括覆盖率分析 Agent 和查询 Agent,主要用来实现源程序的语句和分支覆盖率测试,以及自动查询覆盖率不满足要求时程序中未覆盖目标的情况。

(1)覆盖率分析 Agent 模块。

将插桩后的源程序编译并运行,输入事先准备好的测试用例,选择需要的测试类型,覆盖率分析 Agent 会调用相应的插桩函数库,同时相应地生成一个插桩记录文件,里面记录了该测试类型中的所有函数的 ID 号,还能将测试结果记录到事先确定的缓冲区中。通过将测试结果与期望值相比较,即能判定覆盖率是否满足要求[10]。

(2)查询 Agent 模块。

当覆盖率不能满足要求,查询 Agent 便会自动开始工作,首先分析插桩记录文件,找出未覆盖目标,然后根据控制流图,找出到达此目标的路径。之后测试人员便可依据此路径上各分支节点找出要到达此目标应满足的条件,设计新的测试用例,以增加覆盖率,同时还可能找到程序隐藏的错误。最后将结果以 GUI 的形式显示[11]。

3　结束语

随着软件业的发展,人们对软件质量要求的不断提高,无论从工程上还是实验系统阶段,软件测试都会受到越来越多的关注和推广。文中提出的基于 Agent 的软件测试系统是一个基于 Agent 的具有自治性、智能性、社交性的自动化测试工具,能快速并有效地实现软件的覆盖率测试。

目前,本系统中预处理 Agent 模块中需输入源程序的词法、语法分析文件,即仍需一定的人为工作量,

(下转第43页)

$NA=0$;②temp = $M1=-7$, $M1=1$;③令牌A下一个访问的进程为$P6$;将令牌A发往下一个进程。

步骤2,如b图所示:进程$P2$接收到令牌B:① $M2=7 => NB=NB-1=-2$;② $M2=-PB=-7$;③该进程请求访问共享资源 => $PB=2$;④将令牌B发往进程$P3$。与此同时进程$P6$接收到令牌A: ① $M6=7 => NA=1$;②temp = 7, $M6=1$;③令牌A下一个访问的进程为$P5$,将令牌A发往下一个进程。

步骤3,如c图所示:进程$P3$接收到令牌B:① $M3=7 => NB=NB-1=-3$;② $M3=-PB=-2$;③将令牌B发往进程$P4$。与此同时进程$P5$接收到令牌A:① $M5=7 => NA=0$;②temp = 7, $M5=1$;③令牌A下一个访问的进程为$P4$,将令牌A发往下一个进程。

步骤4,如d图所示:假设进程$P4$先接收到令牌B: ① $Mi=7 => NB=NB-1=-4$;② $Mi=-PB=-2$;③将令牌B发往进程$P5$。这时进程接收到令牌A: ① $Mi=-2 => NA=0$;②temp = $Mi=-2$, $Mi=1$;③根据temp可知令牌A下一个访问的进程为$P2$,将令牌A发往下一个进程。

在令牌环中等负荷的时候,该改进算法可以加快A令牌沿着环传递的速度,从而提高算法的有效性,但是要求环里的每个进程知道环里的所有进程的地址。

4　结束语

文中分析了基本的令牌环算法和能够检测令牌丢失的双令牌算法,提出了一种改进的双令牌算法。该算法在延续原算法的检测令牌丢失的基础之上,特别是在中等负荷的情况下,提高了原算法的有效性。

参考文献:

[1] Ricart G, Agrawala A K. An optimal algorithm for mutual exclusion in computer networks [J]. Communications of the ACM, 1981, 24(7): 9-17.

[2] Maekawa M. A square-root(N) algorithm for mutual exclusion in decentralized systems[J]. ACM Transactions on Computer Systems, 1985, 2(6): 145-159.

[3] 邢　雁. 关于分布式系统进程互斥算法的研究[J]. 哈尔滨商业大学学报(自然科学版), 2003(1): 43-46.

[4] 王昕晔,叶慧娟,李　新. 分布式系统设计中的互斥问题[J]. 海军工程大学学报, 2003(5): 105-112.

[5] 李旭芳. 分布式系统中进程的同步与互斥算法讨论[J]. 计算机工程与设计, 2004(6): 935-937.

[6] 李美安,刘心松,王　征. 非稳定环境下基于竞争消息复杂度的分布式互斥节点容错算法[J]. 微计算机信息, 2005(26): 145-147.

[7] 石　敏,金　辉. Recart&Agrawla 同步互斥算法的改进[J]. 计算机应用研究, 20046): 32-34.

[8] 孙辰军,王翠茹. 分布式系统进程互斥算法的研究与改进[J]. 微计算机应用, 2005, 26(2): 139-141.

[9] Raymond K. A distributed algorithm for multiple entries to a critical section [J]. Information Processing Letters, 1989, 30(2): 189-193.

[10] Nishio S, Li K F, Manning E G. A resilient mutual exclusion algorithm for computer networks [J]. IEEE Transactions on Parallel and Distributed Systems, 1990, 1(7): 344-355.

[11] Lynch N A. 分布式算法[M]. 舒继武,李国东,余华山,等译. 北京:机械工业出版社, 2004.

[12] Misra J. Detecting termination of distributed computations using markers[C]//Proc. ACM Symposium on PODC. Montreal: [s. n.], 1983: 290-294.

(上接第39页)

下一步的工作将围绕这部分内容展开,考虑如何减少人工的参与,最大程度地提高自动化测试的效率[12]。

参考文献:

[1] 王　璞,张臻鉴,王玉玺. 基于覆盖的软件测试技术在实时嵌入式软件中的应用研究[J]. 计算机工程与设计, 1998, 19(6): 45-49.

[2] 严建峰,李伟华,刘　明. 多 Agent 系统任务分配的研究[J]. 计算机工程, 2009, 35(11): 221-222.

[3] 郑明辉,周慧华,马光致. 面向对象的软件测试多 Agent 系统研究[J]. 计算机应用, 2004, 24(5): 94-97.

[4] Landi W. Undecidability of Static Snalysis[J]. ACM Lett. Programming Languages Syst, 1992, 1(12): 32-46.

[5] 裘杭萍,肖登海,连向磊,等. 一种新的基于 Agent 的体系机构[J]. 计算机技术与发展, 2010, 20(1): 50-53.

[6] 周　清,林　拉. 基于 Agent 技术的在线测试系统研究与设计[J]. 计算机技术与发展, 2007, 17(10): 184-188.

[7] 樊　玮,陈增强,袁著祉. 基于 Agent 的分布式数据库设计及其在飞行计划系统中的应用[J]. 南开大学学报, 2004, 37(3): 49-54.

[8] 刘　军,刘卓军. 改进 lex 和 yacc 以开发基于事件驱动的语法分析类软件[J]. 计算机工程与设计, 1999, 20(1): 17-23.

[9] 乔文军. 嵌入式软件测试平台的研究与实现[D]. 南京:南京航空航天大学, 2007.

[10] Zhu H. A formal analysis of the subsume relation between software test adequcy criteria [J]. IEEE Trans. on Software Eng., 1996, 22(4): 248-255.

[11] Frankl P. A formal analysis of the fault-detecting ability of testing methods[J]. IEEE Trans. on Software Eng., 1993, 19(3): 202-212.

[12] 赵欣培,李明树,王　青,等. 一种基于 Agent 的自适应软件过程模型[J]. 软件学报, 2004, 15(3): 348-359.