基于程序切片技术的回归测试方法研究

陈永郑[1],李龙澍[1,2]
(1.安徽大学 计算机科学与技术学院,安徽 合肥 230039;
2.安徽大学 计算智能与信号处理教育部重点实验室,安徽 合肥 230039)

摘 要:软件测试是软件开发过程的一个重要组成部分,是进行软件有效性检查、提高软件质量的重要手段。随着软件规模的不断增大、复杂度的不断提高,传统的软件测试技术在处理大规模复杂软件系统时会出现许多问题。程序切片是一种程序分解术,主要是通过寻找程序内部的相关性来分解程序,从而达到快速错误定位或理解程序的目的。主要探讨将程序切片技术引入到软件测试中,尤其是分析在回归测试中切片方法是如何提高效率的。

关键词:程序切片;软件维护;回归测试
中图分类号:TP311.5 **文献标识码**:A **文章编号**:1673-629X(2007)12-0113-03

Regression Testing Based on Program Slicing

CHEN Yong-zheng[1], LI Long-shu[1,2]
(1. School of Computer Science and Technology, Anhui University, Hefei 230039, China; 2. Ministry of Education Key Lab. of Intelligence Computing and Signal Processing at Anhui University, Hefei 230039, China)

Abstract: Software testing is an important phase during the software development process, it is used to check the validity of software and improve its quality. With the increasing complexity of software, traditional testing technology met more and more problems. Program slicing is a method for decomposing automatic technique for determining which code in a program is relevant to a particular computation. Put focus on how to use program slicing during the regression testing, and on how to improve the testing efficiency.

Key words: program slicing; software maintenance; regression testing

0 引 言

程序切片(program slice)是一种程序分解术[1]。最初的概念是由Weiser教授在其1979年的博士论文中提出的,他不仅建立了程序切片的概念,并提出了基于控制流图的计算程序切片的算法。此后出现了略有不同的定义以及计算切片的算法。大体上说,程序切片技术的发展经历了从静态到动态、从前向到后向、从单一切片过程到多个切片过程、从面向过程切片到面向对象的切片等等。20世纪70年代初期,随着计算机软件技术的发展,特别是计算机软件规模的扩大,相当一部分学者都在致力于研究大规模计算机程序的调试和维护问题。采用一些分解术将复杂的程序分解成一个个相对较小的片段来进行分析和维护是一种很适用的方法。程序切片技术经过20多年的发展,在软件维护、程序调试、测试、代码理解以及逆向工程等方面都有许多的应用。比如在程序调试中,当错误特征出现时,通过切片技术剔除程序中不可能产生该错误的部分,从而把错误限定在一个较小的范围中。

在软件测试和维护过程中,一旦发现软件系统中存在的错误,则应立即排除它们,由于程序中各元素往往是相关的,为了保证改正错误后没有引入新的错误,必须对其进行的测试,称之为回归测试。回归测试意义重大,目前已有很多种回归测试的策略,可以大大减少回归测试的时间和人力开销。文中将论述把程序切片技术引入到回归测试中的合理性和可用性,以及它给回归测试效率带来的提高。

收稿日期:2007-03-18
基金项目:国家自然科学基金(60273043);安徽省自然科学基金(050420204);安徽省教育厅自然科学研究项目(2006KJ098B)
作者简介:陈永郑(1984-),女,安徽寿县人,硕士研究生,研究方向为计算机语言分析、软件测试;李龙澍,教授,博士生导师,研究方向为智能软件和知识工程。

1 程序切片的定义

M.Weiser程序切片是Weiser教授最初提出的程序切片的定义[2],它必须满足下面两个条件:

(1)一个程序切片对应一个特定的切片准则(slicing criterion)(n, V),其中n表示程序中某条语句或

信息点，V 表示在语句点 n 使用的变量集合。

(2) 程序 P 的切片 S 可以通过从 P 中删除 M 条语句得到(其中 $M \geqslant 0$)，但要保证程序 P 和切片 S 的行为轨迹相同。可见，Weiser 切片是一种可执行的程序。

此后经过推广，由多位学者提出的程序切片的典型定义方法是：给定某个程序 P，程序切片准则(n，V)，那么程序 P 关于切片准则(n，V)的程序切片是由程序 P 中所有在 n 点对变量 $v \in V$ 的值有影响的语句和控制谓词构成(这里 n，V 的意思参见前面 M.Weiser 程序切片定义中的第一点)。显然，这种程序切片不一定可执行，但在语义投影上与原程序是一致的。

下面给出一个简单的程序切片的实例。

```
1 scanf("%d",&n);
2 i=1;
3 sum=0;
4 price=1;
5 while( i <=n )
  {
6   sum=sum + i ;
7   price=price * i ;
8   i++;
  }
9 printf("%d",sum);
10 printf("%d",price);
```

那么关于切片准则(9，sum)的程序切片就是下面这个程序片段，由语句 1，2，3，5，6，8，9 构成。

```
1 scanf("%d",&n);
2 i=1;
3 sum=0;
5 while( i <=n )
  {
6   sum=sum + i ;
8   i++;
  }
9 printf("%d",sum);
```

同样道理关于切片准则(10，price)的切片就是由语句 1，2，4，5，7，8，10 组成的程序片段。

可见，切片方法主要运用的是一种剔除不相关保留相关的思想[3]，可以很有效地减小程序规模，便于对关键部分进行分析和理解等操作。正因为切片方法的这种“将复杂问题进行分解和简化，去除不相关的细枝末节，抓住问题的主要矛盾进行突破”的基本思路使得基于程序切片的软件测试技术有很好的应用前景。

2 传统的回归测试中存在的问题

回归测试是软件测试的一种，是在软件增加了新功能或者经过了修改后进行的。其宗旨是验证软件中被修改的部分，以确保没有在测试过的代码中引入新的错误，并且没有对软件的其他部分产生错误影响[4]。也即确保所做的修改没有改变原有代码的行为特征，并且确实达到了想要实现的目标。

回归测试是软件测试中的重要组成部分，在软件开发过程中占有很大的比重，大约 30% 的错误都是通过回归测试发现的[5]。回归测试在渐进式或者迭代式的软件开发过程中占有很重要的份量，以上特点也决定了回归测试的实施是一项工作量大且烦琐的工作，因为在渐进式和迭代式的开发过程中，需要对软件不断地增加新的功能添加新的代码，这使得每一迭代阶段都要进行相应的回归测试以及时地确保软件开发的正确性。

对一个软件进行大规模测试后，如果对软件进行了某种小小的修改，这种修改可能仅仅影响程序中其它的很小一部分。回归测试通常要解决的问题有两点：一是识别出现有的测试用例中需要重新运行的部分；二是识别出需要重新测试的程序部分。传统软件维护的过程中回归测试所存在的问题是：必须对整个程序重新进行大规模测试。也就是说即使正确的部分可能还会进行重复的测试，因为在测试前可能并不知道这些部分和修改不相关。显然这会使维护工作具有很大的冗余性。所以重要的一点是如何避免进行无谓的重测；第二点：被修改的部分可能很容易被发现并重新测试，但是修改所产生的间接影响的部分可能就不容易被发现，从而导致测试的不完全，也就是覆盖率的问题。所以如何准确地定位由于修改而产生的间接影响的部分也是一个关键问题。

3 在回归测试中应用程序切片方法

软件维护过程中的修改一般都会保留软件修改前的大部分功能，某些修改可能只会影响个别路径，因此回归测试中并非对原程序的所有路径都要检测，只需对这些个别的路径进行检测就可以了，而运用程序切片就可以轻松地找到这些“个别”的路径，使得修改后的重新测试不再采用“全部重测”或者“从零开始”的模式[6]。程序切片可以大大地提高回归测试的效率，减小回归测试的时间代价。根据切片的性质，运用切片技术找到那些受修改影响的部分(包含直接影响和间接影响)，只对这些部分进行重新测试完全可以保证回归测试的正确性和覆盖性。下面给出证明：

定义(切片补集的概念)：如果从程序 P 可以根据切片变量计算切片 Slice(n，v_1)，Slice(n，v_2)，…，Slice(n，v_k)，令 C 表示切片补集，则有

$$C = P - \bigcup_{k}^{i=1}(\mathrm{Slice}(n, v_i))$$

性质(覆盖定理):任何一个程序都可以分解为一组程序切片和切片补集的并运算。也就是说任何一个程序都可以用一组切片及其补集覆盖。

$$P = (\bigcup_{k}^{i=1}\mathrm{Slice}(n, v_i)) \cup C$$

由上面的性质(覆盖定理)可知:任何一个程序都可以等价于一组程序切片的并集,而这些切片都是根据某个切片变量和切片准则计算出来的。根据切片的定义:所有能够影响到的语句、谓词等都被包含到该切片变量的切片中了。所以对某个切片变量的修改一定不会影响到其他切片变量的切片。

那么基于切片方法的回归测试思想可以描述如下:针对修改后的程序,首先找出被修改的变量信息,然后运用切片方法找到由于这些变量的变化所引起的直接定义 - 使用关系和间接定义 - 使用关系(通常是一些语句或者控制流和数据流信息),将这些信息提取出来,组成一个程序片段,设计测试用例对这些程序片段进行测试,最后把这些测试用例加入到原程序测试用例中,构成新的回归测试用例集。

目前使用的一种基于控制流的方法[7]就是利用程序切片进行回归测试以检测出受程序修改影响的变量的定义 - 使用关系。这种方法同传统的基于数据流方法比较,在检测新的定义 - 使用关系时既不需要数据流信息的历史数据,也不需要对整个程序重新计算数据流,并且这种方法的定义 - 使用关系的测试覆盖率和对整个程序进行重新测试所达到的覆盖率一样。

需要说明的一点是基于程序切片的测试并不适用于整个软件程序的测试,因为对于稍有规模的软件程序,构造其程序切片比较困难,即使能够构造出程序切片,相应的切片也会比较大,从而失去了简化的目的。基于切片技术的软件测试往往用来对软件中一些重点功能模块进行测试。在回归测试中的应用也正是符合了这一点。

4　基于切片方法的回归测试的一般步骤

P 表示:先前经过测试的程序。

P_1 表示:修改后的程序。

T' 表示:用于测试 P_1 新增功能的测试用例。这里的 T' 就可以通过切片方法快速构建。

测试用例集 T 测试 P,测试用例集 T_1 测试 P_1。一般情况下有:$T_1 = T + T'$,也即 $T_1 \supset T$。

通常,回归测试的步骤是这样的:

1) 识别出 P_1 中受影响的程序组件的集合。

2) 在 T 中识别出用于测试受影响的程序组件的子集。

3) 识别出 P_1 中没有被测试且需要生成新的测试用例的序组件。

4) 针对 3)的结果生成新的测试用例。

5) 令 T_1 为步骤 2) 和 3) 结果的并集,在 T_1 上测试 P_1。

回归测试原本是一项烦琐的工作,引入切片方法后,从理论上来说可以快速构建 T',从而减少得到最终的测试用例集 T_1 的时间,并且降低了这一过程的复杂度。

文中所提到的基于切片的方法主要是用于回归测试步骤中的第二步,即如何在原测试用例集中寻找出受影响的程序组件的子集。此外,对于最终的测试用例集 T_1,实际上应该有 $T_1 \leqslant T + T'$,可以这样理解:从修改后的角度来看,原程序 P 可以被划分成两部分:受修改影响的部分和不受修改影响的部分,那么原来的测试用例集 T 也就可以相应的被分为两部分:T_a 测试 P 中未受修改影响的部分;T_b 测试 P 中受修改影响的部分。对于修改后所得的程序 P_1,某些新增组件需要生成新的测试用例,这部分新的测试用例就是 T,那么最后新的测试用例集就是:$T_1 = T' + T_b \leqslant T + T'$。

5　结束语

基于切片技术的回归测试方法目前已经有实际运用,在软件维护过程中还可以采用一种叫分解切片的技术(decomposition slicing)。这种技术使得维护人员可以直观地判断出一个模块中哪些语句和变量可以被安全修改,也即这种修改不会扩散到其他模块中,哪些语句和变量不能随意被修改。K B Gallagher 等人还给出了一些用于禁止那些会影响其他模块的修改的原则。遵照这些原则所做的修改就可以在线性时间内回馈到原程序中[5]。带来更大的好处是:维护者可以在一个模块中测试对语句或变量的修改,而不必担心这种修改是否会影响到其他模块。由此在某种程序上说就可以省略传统软件维护过程模型中的回归测试了。但是在维护过程中如果必须对一些部分进行修改的话,那么回归测试还是避免不了的。所以基于程序切片方法的回归测试研究还是势在必行的。切片方法的运用使得回归测试更加快速、准确。对于如何更准确的实现其自动化则是下一步要做的工作。

参考文献:

[1]　李必信,方祥圣,郑国栋.基于切片的软件测试技术研究

(下转第 176 页)

分析、整理和过滤,生成报表、日志或控制信号,并通过数据库进行统一管理。实现上述应用模型的关键是OPC客户和各子系统的OPC服务器。OPC服务器主要有三个功能:封装、通讯和控制功能。它封装该子系统的状态和报警信息,使用定义好的OPC标准信息格式和OPC标准接口与集成平台(OPC客户)进行通讯。另外,它接收来自集成平台的控制信息,通过对该子系统的文件、数据库或应用程序接口(API)的系统调用来完成其控制功能。这些对象和集成平台的接口对象都是OPC对象,它们以客户/服务器的模式进行交互。当子系统因为升级等原因发生变化时,只需对于系统对象进行修改,而无需涉及集成平台,因而系统的开放性和可维护性好,升级空间大。在上述应用模型中,通过OPC接口模块,可以在各个子系统之间建立开放的、具有可互操作性的连接,用户不必再担心集成不同子系统的接口问题,可以自由地选择合适的软件和设备。

还可以利用工业以太网与外界 Internet 互连,将采集的数据进行组态并以 Web 的形式发布,使管理者和使用者可以及时了解建筑物各个系统的使用和运行情况,并根据特殊要求来处理事件,实现远程控制访问,真正解决地域的限制。同时处在不同地方的子系统通过OPC将实时采集的数据上传到后台数据库服务器中,使用数据挖掘等技术,为管理层提供更加准确可靠的信息,消除一些人为因素,更有助于管理层的分析与决策。

4 结束语

智能建筑管理系统集成是智能建筑一体化集成的重要前提,工业以太网是最具开放性的工业控制网络体系结构。这种新型的网络体系,与现场总线在以太网方面的发展相呼应。它和OPC技术的出现,大大改进了智能建筑控制系统的开放性和互操作性。此外,在异构计算机环境,使智能建筑管理系统集成变得更为简单,并为实现智能建筑整个弱电系统在实时控制域与信息管理域的全面集成创造了良好的软件环境,同时为管理者提供了更加及时、准确、可靠的决策信息,随着工业以太网及OPC技术的不断发展,必将为智能建筑系统集成带来新的天地。

参考文献:

[1] 张瑞武.智能建筑[M].北京:清华大学出版社,1996.

[2] OPC Common Definitions and Interface Version 1.0[S].OPC Foundation,1998.

[3] OPC Data eXchange Interface Specification Version 1.0[S].OPC Foundation,2003.

[4] OPC Data Access Custom Interface Specification, Version3.0 [S].OPC Foundation,2003.

[5] 罗公亮.希望的曙光——工业以太网数据交换标准 OPC DX[J]. 冶金自动化,2002(2):1-5.

(上接第 109 页)

IEEE Trans, Pattern Anal Machine Intell, 1986,8(6):679-698.

[2] Lee J S J, Haralick R M, Shpiro L G. Morphologic edge detection[J]. IEEE Trans on Roboties Automat, 1987,3:140-156.

[3] 崔 屹.图像处理与分析——数学形态学方法与应用[M].北京:科学出版社,2000.

[4] Chanda B, Kundu M K, Padamaja Y V. A Multi-scale Morphologic Edge Detection[J]. Pattern Recognition, 1998, 31(10):1469-1478.

[5] 刘 循,游志胜.多尺度形态学图像边缘检测方法[J].光电工程,2003,30(3):56-58.

(上接第 115 页)

[J].计算机科学,2001,28(12):97-112.

[2] Weiser M. Program Slicing: Formal, Psychological and Practical Investigations of an Automatic Program Abstraction Method[D]. Ann Arbor, Michigan: University of Michigan, 1979.

[3] Korel B, Rilling J. Program Slicing in Understanding of Large Programs[C]//6th International Workshop. [s.l.]:[s.n.], 1998:145-152.

[4] 郑人杰.计算机软件测试技术[M].北京:清华大学出版社,1997.

[5] Kaner C. Improving the Maintainability of Automated Test Suites[EB/OL]. 1997. http://www.kaner com/lawstl.htm.

[6] 郑亚玲,胡和平.回归测试策略的新领域[J].计算机应用研究,2000(6):23-25.

[7] Gupta R, Harrold M J, Soffa M L. An Approach to Regression Testing Using Slicing[C]. IEEE. International Conference on Software Maintenance(ICSM). [s.l.]:[s.n.],1992:299-308.

[8] Gallagher K B, Lyle J R. Using program slicing in software maintenance[J]. IEEE Transactions on Software Engineering, 1991,17(8):751-761.