一种并发程序可测试性分析框架

陈振强 徐宝文 许 蕾 张 斌

(东南大学计算机科学与工程系 南京 210096)
(武汉大学软件工程国家重点实验室 武汉 430072)
(江苏省软件质量研究所 南京 210096)

摘 要 软件可测试性是对测试软件难易程度的预测,在测试、度量等许多领域中得到了广泛应用.由于并发程序执行的不确定性,其可测试性分析尚有很多难点有待解决.该文提出了一种并发程序可测试性分析框架.在充分分析程序内部数据流、控制流以及并发和同步对数据流和控制流影响的基础上,从单个并发单元、并发因素、共享变量因素及通信关系 4 个方面对并发程序的可测试性进行了分析,为综合度量并发程序的可测试性提供了依据.

关键词 软件可测试性;并发;同步;复杂度
中图法分类号 TP311

A Framework to Analyze Testability of Concurrent Programs

CHEN Zhen-Qiang XU Bao-Wen XU Lei ZHANG Bin

(*Department of Computer Science and Engineering, Southeast University, Nanjing* 210096)
(*State Key Laboratory of Software Engineering, Wuhan University, Wuhan* 430072)
(*Jiangsu Institute of Software Quality, Nanjing* 210096)

Abstract Software testability is the degree to which a system facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met. It has been widely used in testing, measurement, etc. Since the execution of concurrent programs is non-deterministic, there are many issues to be solved in analyzing testability of concurrent programs. The paper proposes a framework to analyze testability for concurrent programs. In this framework, we analyze the data flow and control flow of the programs and the influence of concurrency and synchronization. Based on such information, we propose the testability measure from four facets: independent concurrent units, concurrency, shared variables and communication factors. This gives a guideline to measure the testability of concurrent programs.

Keywords software testability; concurrency; synchronization; complexity

1 引 言

软件测试是为了发现软件故障(faults)而执行程序的过程.随着软件规模的不断扩大、程序复杂度的不断提高,软件测试变得越来越困难.由于资源限制,不可能对一个软件进行完全测试.因此,很难保证一个复杂软件准确无误地执行.为了发现尽可能

收稿日期:2002-06-03;修改稿收到日期:2003-04-06.本课题得到国家自然科学基金(60073012)、国家"九七三"重点基础研究发展规划项目(G1999032701)、江苏省自然科学基金(BK2001004)、高等学校重点实验室访问学者基金、武汉大学软件工程国家重点实验室开放基金、江苏省计算机信息处理技术重点实验室开放基金(苏州大学)资助.**陈振强**,男,1976 年生,博士,主要从事软件分析、理解与测试方面的研究.**徐宝文**,男,1961 年生,博士,教授,主要研究领域为程序设计语言、软件工程、并行与网络软件等. E-mail:bwxu@seu.edu.cn.**许 蕾**,女,1978 年生,博士研究生,主要从事 Web 软件分析、测试与应用方面的研究.**张 斌**,男,1978 年生,硕士,主要从事并发程序分析与测试方面的研究.

多的故障,需要合理分配有限的资源.

软件可测试性分析是一种度量技术,是对测试软件难易程度的预测,为我们分配测试资源提供重要依据.软件的可测试性越低,测试越困难,隐藏故障的可能性越大.软件的可测试性由代码的结构和语义以及假定的输入分布决定[1].当软件测试开始时,利用可测试性分析对各个模块所需要的测试用例进行评估要比盲目测试好得多.对一些关键系统,可测试性已经成为其开发过程中需要考虑的一个重要因素.

估计软件的可测试性也是一项非常困难的工作.目前,国内外已提出一些分析可测试性的方法[2~8],其中比较典型的有故障/错误(Fault/Failure)模型[9]、PIE(Propagation,Infection 和 Execution)分析[10]、DRR(Domain/Range Ratio)分析[11]和复杂性度量[12].这些方法大多只适用于度量顺序程序的可测试性,尚未发现针对并发程序的可测试性分析方法.由于并发程序执行的不确定性,并发程序的可测试性分析变得更加困难.

在已有的工作中,我们已经对软件分析与度量技术进行了比较深入的研究,并取得一些研究成果[13~20].本文将在已有工作的基础上对并发程序中各种因素进行深入剖析,研究它们对可测试性的影响,提出一种分析并发程序可测试性的框架.为了叙述方便,本文将并发单元称之为"任务".

2 并发程序可测试性分析

一般地,软件可测试性可根据不同类型的测试(如随机测试、数据流测试等)进行不同的定义.许多研究人员根据自己的理解对软件可测试性进行了定义,但都大同小异.其中,比较典型的、引用比较广泛的是 IEEE 和 Voas 与 Miller 的定义.

2.1 定 义

IEEE 软件工程标准术语库中对可测试性定义有两层含义[21]:(1)在测试过程中,是否容易建立测试标准以及判断系统或组件满足测试标准的难易程度;(2)在测试过程中,是否允许建立测试标准以及判断需求满足测试标准的难易程度.

Voas 和 Miller 曾对软件可测试性进行两种不同但又相近的定义:

(1)任意给定输入,测试过程中揭示(reveal)程序中存在的故障的可能性[10].

(2)如果一个软件存在故障,在下一次测试时,软件失败的可能性,即软件可测试性是揭示程序中潜在故障的容易程度[8].

软件可测试性越高,越容易发现故障.与顺序程序相比,影响并发程序可测试性的关键因素是其运行的不确定性.总体上,影响并发程序可测试性的因素可分为三类:单任务因素、并发因素和通信因素.并发单元间的通信方式主要有两种:共享变量和同步.一般地,不确定因素越多,测试越困难,可测试性越低.本节将从以上这几个方面讨论并发程序的可测试性度量方法.

2.2 单个任务的可测试性分析

一般地,任务越复杂,测试越困难,可测试性越低.因此,可直接用任务的圈复杂度(cyclomatic complexity)的倒数作为任务内的可测试性度量.圈复杂度,通常称为程序复杂度或 McCabe 复杂度,是一种广泛应用的静态软件度量,由 McCabe 于 1976 年提出[12],用来度量程序中线性无关的路径数.它提供了一个单一的有序数,可与其它程序的复杂度进行比较.软件模块的圈复杂度可通过分析模块的连通图(表示模块的控制流)获得

$$McCabe(Task) = E - N + p,$$

其中,E 为连通图的边数;N 为图中节点数;p 为连通组件数.

一般地,程序中包含两种关系:控制流和数据流.圈复杂度度量了程序中的控制流间的关系,为了较全面地分析单个任务的可测试性,还需要分析程序中数据流关系,这可通过输入/输出比例(DRR)来描述.

DRR 方法由 Voas 和 Miller 首先提出[11],用来度量程序中的信息隐藏情况.对一个函数 f,设 α 为输入集的势;β 为输出集的势,则 $DRR(f)=\alpha:\beta$.例如函数 $f(x)=x+1$ 的每个输入 x,都对应一个可能的输出 $f(x)$,$DRR=1:1$;函数 $f(x)=x$ MOD 20 的 $DRR=\infty I:20$(∞I 表示整数集的势).一般地,$DRR\geq 1$.若 $DRR=1$,则输入与输出间是一一映射,任何输入阶段的信息都可保留到输出.若 $DRR>1$,则输入与输出间是多对一映射,这会丢失一些输入信息.通常,DRR 越大,可测试性越低.

若集合的势为无穷大,则首先采用区间划分的方法进行等价划分,将其转换为有限集;然后计算 DRR.若一函数没有明显的输入/输出,可简单地令 $DRR=1$.

综合这两种因素,单个任务的可测试性可定义为

$$T(Task)=C\times\frac{1}{McCabe(Task)}+D\times\frac{1}{DRR(Task)},$$

其中 $C,D>0$ 为权值,且 $C+D=1$.

2.3 并发因素

直觉上,任务越多,任务间的组合越复杂,测试越困难,可测试性越低.设程序 P 中共包含 k 个任务,第 i 个任务的可测试性为 $T(Task_i)$,可直接使用平均值作为 P 的可测试性

$$T(P)=1/k\sum_{i=1}^{k}T(Task_i),$$

或者根据任务的重要程度,给每个任务设定一个权值 W,则

$$T(P)=\sum_{i=1}^{k}W_i\times T(Task_i),$$

其中 $W_i>0$,并且 $\sum_{i=1}^{k}W_i=1$.

与顺序程序相比,一个个单独的任务对程序的可测试性影响不大,对并发程序的可测试性影响较大的是这些任务间的组合,任务间的并发执行导致程序执行路径指数级增加.当然,并非程序中所有任务都并发执行,有些是有一定"顺序"的.为了准确刻画程序的并发情况,用并发度来表示程序的并发程度.并发度指程序中可并发执行的任务的最大数目(记为 CN).并发度可通过分析系统的行为,根据程序语义来获得.在已有的工作中,我们已经对并发程序行为分析和表示技术进行了深入研究[13,15,17],建立了并发程序流图模型(CCFG),通过分析 CCFG 可很容易获得并发程序的并发度.限于篇幅,本文不再详述.

显然,并发度越高,组合情况越多,可测试性越低.我们定义可测试性与 2^{CN-1} 成反比.

$$T_{con}=2^{1-CN}\times\Big(\sum_{i=1}^{k}W_i\times T(Task_i)\Big).$$

2.4 共享变量因素

对共享变量的访问导致信息在任务间交互.由于并发任务执行的不确定性,共享变量的组合也是不确定的,从而导致程序内部的数据状态迅速增加.共享变量越多,组合越多,测试越困难,可测试性越低.

如果一个全局变量只被一个任务访问,或者不存在多个任务并发访问全局变量,那么这个变量可以作为局部变量处理.判断一个变量是否只被一个任务访问比较容易;但是判断一个变量是否被多个并发执行的任务并发访问非常困难[13,22].

为了分析共享变量对可测试性的影响,引入两个概念:访问度和依赖集.共享变量的访问度指其被"同时"并发访问的最大任务数.访问度越高,可测试性越低.共享变量的依赖集指其直接或间接依赖的其它共享变量的集合.这样,可通过两种方式定义共享变量对可测试性的影响:

(1)取最小值

$$T_{S\_V}=\min\left\{\frac{1}{AN(V_i)\times\prod\limits_{V_j\in DEP(V_i)}AN(V_j)}\right\},$$

其中,$AN(V_i)$为共享变量 V_i的访问度,$DEP(V_i)$为 V_i的依赖集.

(2)取平均值

$$T_{S\_V}=\frac{1}{m}\sum_{i=1}^{m}\left(\frac{1}{AN(V_i)\times\prod\limits_{V_j\in DEP(V_i)}AN(V_j)}\right),$$

其中,m 为共享变量数.

2.5 同步因素

同步是一种重要的任务间通信的方式,同步会影响程序的控制流,使得程序按照既定的方式运行.称程序中每个可能的同步位置为一个同步点.

在测试过程中,对程序中的同步点,一般需要进行追踪和重演,这会大大增加测试工作量.对于一个任务,包含的同步点越多,其控制流越复杂,可测试性越低.因此,同步点对单个任务可测试性的影响定义为 $SNT_i=1/(SN_i+1)$,其中 SN_i为任务 T_i中的同步点数目.

任务间的通信关系是影响并发程序可测试性的最关键因素之一,通信关系越复杂,可测试性越低.因此,需要提供方法度量任务间的通信复杂度,为此,我们首先引入任务间通信关系图的概念.

任务间通信关系图 $CG=\langle V,E\rangle$,V 为顶点集,表示程序中的同步点;E 为边集,表示程序中可能的通信关系(任务内的同步点通过控制流边连接).

对图 1 所示的 Ada 并发程序,其通信关系图如图 2 所示,在这个示例中,共有 3 个任务(用虚线框表示),8 个同步点.

```
--T1                --T2                --T3
...                 ...                 ...
T2.e1;  ---1        accept e1;          T2.e1;  ---6
A := B+1;                    ---3       if A>1 then
T2.e2;  ---2        B :=1;                T72.e2;
...                 accept e2;                   ---7
                             ---4       else
                    T3.e3    ---5         accept e3;
                    ...                             ---8
                                        end if;
                                        ...
```

图 1　Ada 程序示例

显然,CG 中边越多,任务间的同步越多,可测试性越低,本文将其定义为反比关系.

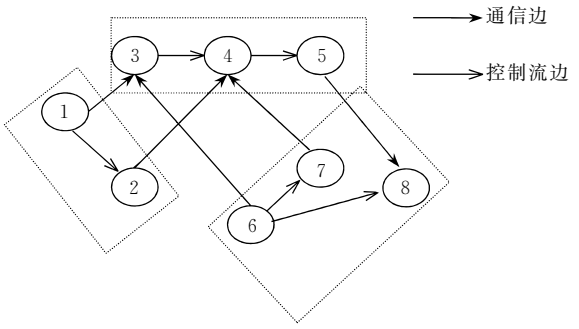


图 2 通信关系图示例

为了计算 CG 的复杂度,对一个连通图 CG,我们定义隔集 M_c 如下:

(1)若 CG 中去除通信边集 M_c,则 CG 将变成非连通图;

(2)对任何 M_c',$|M_c'|<|M_c|$,都不能使得(1)成立.

这样,通信关系图 CG 的复杂度定义为

$$C(CG)=F_c(CG)\times F_s(CG),$$

其中,$F_c(CG)=|M_c|$ 是连通因子,$F_s(CG)=\frac{1}{k}\cdot\sum_{i=1}^{k}C(CG^i)$ 是结构因子,CG^i 是通过 M_c 分解的第 i 个连通子图.明显地,该公式是一个递归计算的过程.

由 M_c 的定义可知,$|M_c|$ 表示了 CG 的连通强度,并且可能存在其它的 M_c',$|M_c'|=|M_c|$,使得(1)成立,即可能存在多个 M_c,因此,我们取它们中的最小值作为最后的度量结果.

对非连通图 CG,如果应用上面的计算方法,则 $|M_c|=0$,$C(CG)\equiv 0$.这显然与实际情况不符.因此,我们将非连通的 CG 作为经过一次分解后得到的连通子图集,并且这次分解的计算公式为 $C(CG)=F_s(CG)-\varepsilon$,其中 $\varepsilon\in(0,1)$.

对连通图,有 $|M_c|\geq 1$,根据定义,$|F_s(CG)|\geq 1$,因此 $C(CG)\geq 1$;对非连通图,若计算得到的 $C(CG)<1$,则设 $C(CG)=1$.

设 k 为任务数,则可测试性与任务间的通信复杂度的关系可定义为

$$T_{syn}=\left(\frac{1}{k}\sum_{i=1}^{k}SNT_i\right)\times\frac{1}{E(CG)+1}\times\frac{1}{C(CG)}.$$

2.6 综合分析

通过上面分析,我们获得了并发程序可测试性的几个不同侧面,为了综合度量其可测试性,需要将这些度量集中起来.这样并发程序 P 的可测试性 $T(P)$ 为

$$T(P)=k_1\times T_{con}+k_2\times T_{S\_V}+k_3\times T_{syn},$$

其中,$k_1,k_2,k_3>0$ 为权值,且 $k_1+k_2+k_3=1$.这些权值用来调整不同侧面对程序可测试性的影响.一般地,同步关系是影响并发程序执行的关键因素,因此,k_3 应取较大值.

3 实例分析

本节将通过对图 1 所示的并发 Ada 程序示例来分析如何利用本文的方法计算可测试性.在这个例子中,A 和 B 为共享变量.

各个任务的复杂度为

$$McCabe(T1)=McCabe(T2)=1,$$

$$McCabe(T3)=2.$$

由于这三个任务没有明显的输入/输出,令 $DRR=1$.设 $C=D=1/2$,则

$$T(T1)=T(T2)=(1/2)\times(1+1)=1,$$

$$T(T3)=(1/2)\times(1/2+1)=3/4.$$

假设这三个任务可并发执行(即并发度为 3),且权值相同,则

$$T_{con}=2^{1-3}\times(1/3)\times(1+1+3/4)=11/48.$$

例子中含有两个共享变量 A 与 B,其访问度都为 2.在 $T1$ 中对 A 的赋值引用了 B,因此 A 依赖于 B,我们采用平均值法可得

$$T_{S\_V}=\frac{1}{2}\left(\frac{1}{2\times 2}+\frac{1}{2}\right)=\frac{3}{8},$$

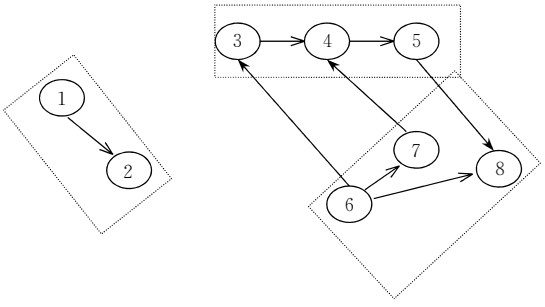
$T1$ 中有两个同步点,$T2$ 和 $T3$ 中各有三个同步点.

$$SNT_1=1/2,SNT_2=SNT_3=1/3,$$

在 CG 中,$E(CG)=5$;边集$\{\langle 1,3\rangle,\langle 2,4\rangle\}$为一个隔集,经过分解可得到两个子图(如图 3 所示),子图 2 可通过去掉边集$\{\langle 6,3\rangle,\langle 7,4\rangle,\langle 5,8\rangle\}$分解得到两个子图.这样

$$C(CG)=2\times\left(\frac{1}{2}\times\left(1+\left(3\times\left(\frac{1}{2}(1+1)\right)\right)\right)\right)=4,$$

$$T_{syn}=\frac{1}{3}\left(\frac{1}{2}+\frac{1}{3}+\frac{1}{3}\right)\times\frac{1}{5}\times\frac{1}{4}=\frac{7}{360}.$$



(a) 子图1 (b) 子图2

图 3 分解示例

令 $k_1=k_2=1/4;k_3=1/2$,则程序的可测试性为

$$T(P)=\frac{1}{4}\times\frac{11}{48}+\frac{1}{4}\times\frac{3}{8}+\frac{1}{2}\times\frac{7}{360}=\frac{463}{2880}.$$

4　结束语

本文提出一种并发程序可测试性分析框架,它从单个并发单元、并发因素及通信因素三个方面对并发程序的可测试性进行了较全面地分析.这三个方面既可以单独度量并发程序的可测试性,也可以综合使用.

可测试性度量是一项非常困难的工作,由于并发程序执行的不确定性,并发程序的可测试性分析更加困难.本文提供的框架只是从直觉上进行分析,还未经过严格证明或进行试验验证,并且还有其它因素没有充分考虑.在将来的工作中,我们将对本文的分析方法进行验证,并分析应该满足的性质.

参　考　文　献

1　Lee J *et al*. Testability analysis based on structural and behavioral information. In: Proceedings of IEEE VLSI Test Symposium, Atlantic City, NJ, 1993. 139～145

2　Bache R, Mullerburg M. Measures of testability as a basis for quality assurance. Software Engineering Journal, 1990, 5(2): 86～92

3　Freedman R S. Testability of software components. IEEE Transactions on Software Engineering, 1991, 17(6): 553～564

4　Howden W, Yudong H. Software testability analysis. ACM Transactions on Software Engineering and Methodology, 1995, 4(1): 36～64

5　Lo B, Shi H. A preliminary testability model for object-oriented software. In: Proceedings of International Conference on Software Engineering: Education & Practice, Dunedin, New Zealand, 1998. 330～337

6　Traon L, Ouabdesselam F, Robach C. Analyzing testability on data flow designs. In: Proceedings of Symposium on Software Reliability Engineering (ISSRE 2000), San Jose, CA, 2000. 162～173

7　Voas J, Miller K. Dynamic testability analysis for assessing fault tolerance. High Integrity Systems Journal, 1994, 1(2): 171～178

8　Voas J, Miller K. Software testability: The new verification. IEEE Software, 1995, 12(3): 17～28

9　Voas J, Miller K, Payne J. Designing programs that are less likely to hide faults. Journal of Systems and Software, Elsevier Science, 1993, 20(1): 93～100

10　Voas J. PIE: A dynamic failure-based technique. IEEE Transactions on Software Engineering, 1992, 18(8): 717～727

11　Voas J, Miller K. Factors that affect software testability. In: Proceedings of the 9th Pacific Northwest Software Quality Conference, Portland, Oregon, 1991. 235～247

12　McCabe T J. A complexity measure. IEEE Transactions on Software Engineering, 1976, 2(4): 308～320

13　Chen Zhen-Qiang, Xu Bao-Wen, Yu Hui-Ming. Detecting concurrently executed pairs of statements using adapted MHP algorithm. ACM Ada Letters, 2001, 21(4): 107～113

14　Chen Zhen-Qiang, Xu Bao-Wen. Slicing concurrent Java programs. ACM Sigplan Notices, 2001, 36(4): 41～47

15　Chen Zhen-Qiang *et al*. A novel approach to measuring class cohesion based on dependence analysis. In: Proceedings of IEEE ICSM, Canada, 2002. 377～383

16　Chen Zhen-Qiang, Xu Bao-Wen, Zhao Jian-Jun. An overview of methods for dependence analysis of concurrent programs. ACM Sigplan Notices, 2002, 37(8): 45～52

17　Chen Zhen-Qiang, Xu Bao-Wen, Yang Hong-Ji, Zhao Jian-Jun. Concurrent Ada dead statements detection. Information and Software Technology, 2002, 44(13): 733～741

18　Xu Bao-Wen, Chen Zhen-Qiang. Dependence analysis for recursive Java programs. ACM Sigplan Notices, 2001, 36(12): 70～76

19　Xu B, Zhou Y. Comments on "A cohesion measure for object-oriented classes". Software——Practice and Experience, 2001, 31(14): 1381～1388

20　Xu Bao-Wen, Zhou Yu-Ming. Extracting objects from Ada83 programs: A case study. Journal of Computer and Science, 2001, 16(6): 574～581

21　IEEE Standard Glossary of Software Engineering Terminology, ANSI/IEEEE Standard 610.12-1990. New York: IEEE Press, 1990

22　Taylor R N. Complexity of analyzing the synchronization structure of concurrent programs. Acta Informatica, 1983, 19(1): 57～84

CHEN Zhen-Qiang, born in 1976, Ph. D.. His research interests include software analysis, understanding and testing.

XU Bao-Wen, born in 1961, Ph. D., professor and Ph. D. supervisor. His research interests include programming language, software engineering, concurrent and internet software.

XU Lei, born in 1978, Ph. D. candidate. Her research interests include Web software analysis, testing and applications.

ZHANG Bin, born in 1978, master. His research interests include concurrent software analysis and testing.