计算机科学1997 Vol. 24 №. 3

程序测试与概率检查

卢先捷
(复旦大学计算机系 上海200433)

摘 要 本文在分析传统的程序结果检查方法的基础上,介绍了引进随机概率的自我检查/校正程序设计(self-testing/correcting)的原理及其特点。

关键词 程序测试,程序正确性,程序检查,容错计算,软件工程。

在当前的软件开发过程中,查错/改错占据了开发周期的相当一部分时间,成为影响开发进度的重要因素之一,如何从程序中快速而完全地查找存在的错误一直是人们关注的热点之一,在过去数十年中提出了各种方法及对策,但一直未从根本上解决问题。

首先,如何确定程序工作正常?即输出是否正确?在传统的测试中,除待测试程序外,假定有一个外部信息源(Oracle)可以提供"正确"的答案,根据该答案即可判断被测试程序的运算结果是否正确。这个外部信息源可以是一些计算速度较慢但相对可靠的程序,或者干脆是人自己。由此可见这样无法保证该外部信息源的绝对正确性及存在性。

其次,如何能够测试"所有"的输入/输出都是正确无误的呢?如果我们知道了所有答案,那也就没有必要开发程序了。因此,在绝大多数情况下,测试输入的所有组合是不可实现的,程序中总会存在着被遗漏的错误。

此外,即使软件是正确的,也不排除动态执行出错的可能性,各种硬件错误及外在不可预料因素必然导致软件执行错误,程序执行的绝对正确性是不可实现的。

自六十年代以来,不少人从事程序正确性证明方面的研究,希望能够找到自动证明及生成的有效方法,即通过严格的数学证明过程来验证程序具有所声称的性能。这方面取得了不少成果,能够自动生成小程序或证明一些小规模程序的正确性,但离实际中的大规模软件程序开发的要求尚有一段距离。

因此,在目前的软件开发中,仍将测试作为最重要的确保软件质量的有力工具。在后续章节中,我们在分析传统的测试过程的基础上,结合随机性介绍了自我检查/校正程序的原理。

一、传统测试过程

给定运行时间为T(n)的一段程序P及测试所需要的输入/输出对(x,y),我们有一个检查程序C,它能够在较少的时间(相对P)内检查y的正确性,即如果输入x后,若P的输出P(x)不为y,则认为出错,否则认为输出结果P(x)正确。

我们要求C的时间复杂性较低是由实际决定的,反之若C的时间复杂度超过P,此时为什么我们相信C的正确性而不是P呢?因此检查答案比计算出结果还要难是不切实际的,例如:给定一个整数,找到它的因子分解(如在RSA公钥体制中起决定作用的二素数因子分解)是困难的,目前尚无多项式时间内的实用算法,但如果给定两个因子,验证二者之积是否与该整数相等则是容易的。更为广泛的NP问题类正是根据验证的相对容易性来定义的,即NP完全问题的验证算法是多项式时间的,但不知道是否存在着多项式时间的解题算法。

同理,我们要求验证程序C的绝对正确性同样是不恰当的,因此,我们仅要求C的出错可能性远比P小得多,可靠性比P更高。

以下形式化地给出测试的规格说明:

●系统 实现函数P(x)的程序P及检验答案正确性的检查程序C

●输入 输入/输出对(x,y)

●输出 如果y=P(x),则接受,否则拒绝。

●可靠性 对所有输入/输出对(x,y),检查程序C给出正确回答的概率大于等于c,c是一个趋近于1的常数。

●时间复杂度 程序P的时间复杂度为T(n),检查程序C的时间复杂度为O(T(n))*,即检查程序C的时间复杂度较低。 □

---

* 若 $\lim_{n\to\infty}\frac{f(n)}{g(n)}=0$,则记其为 $f(n)=O(g(n))$

† 存在一个常数 $c\neq 0$ 和 n_0,使得当 $n\geq n_0$ 时有 $f(n)\leq O(g(n))$,记其为 $f(n)=O(g(n))$。

二、自我检查/校正程序

在实际中，既然绝对正确性是不可能的，那么一个自然的想法是获得可信度极高的计算结果，换句话说，即出错概率应极小且不依赖于具体输入，我们希望结合测试及计算过程以获得高可信度的输出。另一方面，我们不满足仅仅找到错误的存在，而且进一步要求在有错情况下获得正确输出。以下将会看到引入随机性后，我们的确能够设计出一些满足要求的算法。

我们定义自我检查/校正程序如下(以下简称校正程序)：

●系统　P为计算P(x)的程序，P对除小于p(p为一小于½的正数)的部分外的所有输入都能够获得正确答案(出错概率小于p)。同时由一个检查程序C能够检查y的正确性。

●输入　输入x

●输出　输出y

●可靠性　对于所有的输入x，输出y正确的概率大于等于c，c是一个趋近于1的常数。

●时间复杂度　P、C的时间复杂度分别为T(n)、O(T(n))，校正程序的时间复杂性，包括调用P及C的步骤，必须为$O(T(n))^+$；如果将调用P或C作为一步，则时间复杂度为O(T(n))。　□

上述时间复杂度要求表明了自我校正程序的时间复杂度不会比原来的计算程序P有较大的增长，否则就没有实际意义了。

以下我们以一个简单的示例阐述校正程序的工作过程。

假定我们有一个出错概率小于1/100的乘法计算程序P及出错概率小于1/1000的检查程序C，则对输入(x_1,x_2)，按照以下步骤计算：

1. 随机产生两个随机数r_1、r_2；

2. 计算$P(x_1-r_1,x_2-r_2)$、$P(x_1-r_1,r_2)$、$P(x_2-r_2,r_1)$和$P(r_1,r_2)$及以上四数之和S；

3. 检查以上的四组值是否正确，若都正确则输出S，否则转第1步继续计算；

若以上计算都正确无误，可得

$$\begin{aligned}S&=P(x_1-r_1,x_2-r_2)+P(x_1-r_1,r_2)+P(x_2-r_2,r_1)+P(r_1,r_2)\\&=(x_1-r_1)(x_2-r_2)+(x_1-r_1)r_2+(x_2-r_2)r_1+r_1r_2\\&=x_1x_2\\&=P(x_1,x_2)\end{aligned}$$

在以上算法中，四个P计算中都正确的概率为$(99/100)^4$，则至少有一个错误的概率为$1-(99/100)^4<4/100$，同理检查程序出错的概率为$1-(999/1000)^4<4/1000$，因此程序给出正确答案的概率大于$1-4/100-4/1000=956/1000$，出错概率小于44/1000(以上分析忽略了计算与检查程序相互作用的情况，若考虑则获得正确答案的概率将会更大)。

虽然粗看一下，似乎出错概率比原来的P计算程序反而增大了，但这个校正程序有一些极好的特点：第一，它的出错情况与具体输入无关，由随机性和计算过程可知不存在某些一定出错(出错概率为1)的输入计算；第二，可以多次选取随机数并多次计算，将各次计算结果按大小分类，取各类中数目最多的那个值作为计算结果输出，通过概率分析，我们知道这样可以获得出错概率任意小的计算结果。

讨论　什么样的问题具有自我检查/校正程序呢?不少人对此进行了研究，获得了一些成果，例如有限域上的运算、多项式环、整数运算[3,5,7]等等都存在自我检查/校正算法。如果一个函数具有所谓的随机自我归约性质，则可以很容易地实现其自我校正算法。所谓随机自我归约性，可以非形式化地认为，若一个函数f(x)可以表述成为由k个随机变量x_1，x_2，…，x_k的函数值$f(x_1)$，$f(x_2)$，…，$f(x_k)$的简单运算结果，则称之为其具有k-随机自我归约性。九十年代初，由于找到了有限域上的低阶多项式的自我校正算法，从而在NP问题研究方面获得了一系列成果，使人们从随机检查方面重新定义了NP问题类[2]。

寻找更多可用于自我校正算法的问题类是目前的一个研究热点，如节约随机串数量、加快计算过程[6]、部分检查、概率检查、近似检查[1,4]等等。

参考文献

[1]S. Ar, M. Blum et al., Checking approximate computations over the reals, Proc. 25th ACM STOC

[2]S. Arora and S. Safra, Probabilistic checking of proofs; a new characterization of NP, Proc 33rd IEEE FOCS, 1992

[3]M. Blum, M. Luby and R. Rubinfeld, Self-testing/correcting with applications to numerical problems, J. Computer & System Science, Vol. 47, 1995

[4]M. Blum and H. Wasserman, Program result-checking; a theory of testing meets a test of theory, Proc. 35th IEEE FOCS, 1994

[5]P. Genmmell et al., Selftesting/correcting for polynomials and for approximate functions, Proc. 23rd, ACM STOC, 1991

[6]R. Rubinfeld, Batch checking with applications to linear functions, Information Processing Letters, Vol. 42, 1992

[7]R. Rubinfeld, On the rubustness of functional equations, same to [4]