**等价和非等价的平等：在面向对象软件的类级测试中对等价和非等价标准的回顾**

摘要

代数规范已经用于面向对象程序的测试，并从1990年代以来受到了很多关注。 通常认为基于代数规范的类级测试涉及两个独立的方面：等价和非等价基项的测试。 研究人员引用了直观的例子来说明这样一种哲学，即即使一个实现满足由基项的等价性规定的所有要求，它仍然不能满足由基项的非等价性所规定的一些要求。 因此，等价基项的测试和非等价基项的测试被认为是重要的，并且不能彼此替换。

在本文中，我们提出一个创新的发现：给定一个带有适当输入的类的任何规范，一个完整的实现满足所有观察上等价的基项当且仅当它满足所有观察上非等价的基项。 因此，软件测试的这两个方面互相覆盖，因此可以相互替代。 这些发现提供了对基于代数规范的软件测试的更深入的理解，使得该理论更加优雅和完整。 我们还强调了我们的理论结果的几个重要的实践意义。

索引术语 - 软件测试，等价准则，非等价准则，代数规范，面向对象软件

介绍

作为定义面向对象软件的功能需求的主要正式的方法，代数规范在它们的实现的测试中非常有用，当然它们的实现具有许多益处，包括改进测试用例生成的自动化和有效性。

特别地，基于代数规范的面向对象软件的类级测试已经做了广泛的研究。 以前的工作在两个方面对待实现的正确性的测试：是否根据规范证明是等价的两个操作序列（正式地称为基项）将在实现中得到等同的对象，以及是否根据规范证明是不等价的两个操作序列将在实现中得到不等价的对象。

先前的研究认为，一般来说，所有等价基项的测试不足以揭示由于实现中的各种故障可能导致的错误。 同样，所有不等价基项的测试也不行。 因此，必须进行两种测试。 有关通常采用的直观图示，请参阅第4节中的示例2。

在本文中，我们提出了我们的创新发现，给定一个具有适当导入和完整实现的类的规范， 在观察上等价的基项的测试和在观察上不等价的基项的测试彼此覆盖.1换句话说， 如果由于实现中的某个确定故障而导致的错误可以通过测试观察性非等价性来揭示，则由于相同故障引起的另一个错误也可以通过测试观察等价性来揭示，反之亦然。 因此，与以前工作的一般理念相反，我们不需要进行这两种测试。 这个发现加深了对基于代数规范的软件测试的理解，使得该理论更加优雅和完整。 此外，该理论在现实世界软件测试中具有重要的实践意义。 其余部分详细描述我们的新的理论和实践结果。

本文组织如下：第2节描述了基于代数规范的面向对象软件测试的前期工作。 第3节概述了论文中使用的基本概念。 第4节研究等价标准和非等值标准之间的创新关系。 第5节将第4节的结果应用于面向对象软件的类级测试，并举例说明。 第6节讨论了与以前基于代数规范的类级测试相关的理论含义。 第7节突出了两个重要的实际含义。 第8节总结本文。

相关以前的工作

代数规范的想法源于Zilles的工作[33]，Goguen等人。 [21]，和Guttag和Horning [22]。 有很多建议。 例如，Goguen等人引入OBJ3 [20]并将其扩展到面向对象范例的FOOPS [6]，而Bidoit和Mosses [5]设计了通用代数规范语言（CASL），它将各种代数规范特征整合到一种通用语言中。

Bernot等人提出了基于代数规范的软件测试的一般理论。 [3]，[4]。 它包括规则性假设，均匀性假设和其他oracle假设，以形式化oracle问题。 框架的主要优点是明确表达测试的能力和限制以及测试和程序验证之间的差距。 基于该框架，根据均匀性和规则性假设用基项代替规范的公理中的所有变量[7]，[14]，他们已经开发了一个工具来生成测试用例。

Gannon等人的DAISTS方法 [17]建议选择一个参数值的元组作为公理的左侧和右侧的输入，然后调用用户提供的等式函数来检查输出。 如果两侧的输出不相等，则显示故障。 Antoy和Hamlet [2]提出的测试方法使用了证明技术。 与DAISTS不同，它们在抽象域中而不是在具体域中测试抽象日期类型的值的相等性。

Machado [27]提出了以一阶逻辑表达的扁平代数规范所产生的oracles。 Machado [28]将Bernot等人[3]，[4]的框架扩展到具有对量词的限制的一阶逻辑。 Machado和Sannella [29]进一步扩展了CASL架构规范的框架。

Jalote [24]认为公理是重写规则，建议从操作的所有合法组合（即所有基项）中选择测试用例，并通过重写规则导出相应的等价基项。 它们检查对应于每个选择的等价项对的执行结果是否相等。 如果不是，则显示失败。

为了方便在类层面测试面向对象的软件，Doong和Frankl [15]提出了LOBAS，一种代数规范语言，其语法与面向对象编程语言类似，其语义与OBJ3和FOOPS类似 。Doong和Frankl [15]，[16]提出的代数规范的LOBAS语言更适合于面向对象软件的类级测试。 基于LOBAS，他们扩展了Jalote的工作[24]，并开发了一个称为ASTOOT的工具，通过重写从等价项对生成测试用例，以及通过“交换路径条件”从非对等项对生成测试用例。

在代数规范中，当且仅当一个基项不能在给定规范中使用任何公理进一步重写到另一个基项时，它被称为正常形式。 在我们早期的工作中[9]，[10]，两个基项被认为是等价的，当且仅当它们都可以根据给定的规范写成相同的正常形式。 如果通过用给定规范的正常形式替换公理两侧的所有变量来形成一对基项，我们将它们称为等价项的基本对或简称为基本对（表示为“u1〜fun u2“）。 我们在文献[9]中证明，对于给定的任何具有适当导入的类的规范，一个完整的实现满足所有的基本对，当且仅当它满足所有在观测上的基本对。

下面对TACCLE方法和在第3节和第6节中给出的结果[9]，[10]做进一步描述。

根据以前的工作，如[10]，[16]，基于代数规范的类级测试涉及两个方面，即观察上等价基项的测试和观测上非等价基项的测试。 直观的例子2被用来说明这样的哲学：即使观察上等价基项的测试是详尽的，它不能揭示两个不同的状态被混淆为单个状态的失败，也就是说，两个观察上不等价的基项错误地生成两个等价对象[10]，[16]。 因此，已经指出，观察上非等价基项的测试是必要的，即使在对观测上等价基项进行详尽测试后也不能忽略。

Hierons[23]和朱[32]等人调查了这些方面。 Kong[25]和Yu[31]等人扩展代数测试方法来覆盖软件组件，并提出了一种称为CASCAT的Java组件自动测试工具。

基于代数规范面向对象软件的类级测试的基本概念

本文考虑了基于代数规范的面向对象软件的类级测试。为了使本文可以是独立的，我们首先总结代数规范的基本概念。读者可以参考现有的工作[9]，[10]，[16]了解面向对象软件的代数规范的更多细节。为了与以前的工作进行比较的公平性，我们将使用与Aiguier等人相同的代数规范约束水平。 [1]，Doong和Frankl [15]，[16]和Chen et al。 [9]，[10]。读者还可以参考[10]的第4.6节，解释为什么约束在面向对象软件测试中是合理的。我们承认，像基于代数规范的面向对象软件测试的相关工作，我们不能涵盖使用规范的非确定性系统。虽然这是一个有趣的话题，值得进一步研究，但超出了本文的范围。

类的代数规范由两部分组成。 第一部分通过根据输入和输出类声明操作来定义类的语法。 第二部分通过以条件方程的形式列出公理来定义类的语义，其描述操作的功能需求。 以下是代数规范的示例。 为了便于阅读，我们将在本文中为不同的目的重用相同的示例。

Example 1 (adapted from Chen et al. [9]). Algebraic specification of a class *IntStack* of stacks of integers.

**module** *INTSTACK* **class** *IntStack* **import classes***Int*

*Bool*

**operations**

*new*: → *IntStack*\_.*isEmpty*: *IntStack* → *Bool* \_.*push*(\_): *IntStack Int* → *IntStack* \_.*pop*: *IntStack* → *IntStack*\_.*top*: *IntStack* → *Int* ∪{*nil*}

**variables**

*S*: *IntStack*; *N*: *Int* **axioms**

*a*1: *new*.*isEmpty* *true a*2: *S*.*push*(*N*).*isEmpty* *false a*3: *new*.*pop* *new a*4: *S*.*push*(*N*).*pop* *S a*5: *S*.*top* *nil if S.isEmpty a*6: *S*.*push*(*N*).*top* *N*

语句“\_.push（\_）：IntStack Int→IntStack”意味着操作push有一个输入类IntStack，一个参数类Int和一个输出类IntStack。

给定代数规范，语法有效的操作序列称为项。 没有变量的项被称为基项。 在示例1中，例如，S.push（N）.pop.top是一个项，new.push（2）.pop.top是一个基本项。

假设基项u包含作为公理ai的左侧输入vi的实例的子词v：vi = vi'。 如果我们用右侧输入vi'的相应实例v'替换子词v，并且如果结果是基项u'，则u被称为重写到u'，使用公理ai作为左到右的重写规则。 例如，可以使用公理a4作为重写规则将示例1中的基本项new.push（2）.pop.top重写为new.top。

令u1，u2，...，un为基项。 如果u1可以重写为u2，并且u2可以重写为u3，...，并且un-1可以重写为un，则我们说存在从u1到un的重写关系（表示为“u1〜 rew un“）。我们说，当且仅当它不能被说明书中的任何公理进一步重写时，一个基项处于正常形式。 当且仅当每个基项可以通过有限数量的重写重写为唯一的正态形式时，代数规范被称为规范。

我们将规范中指定的函数称为操作，并将程序中实现的函数称为方法。 在C类中创建对象的操作（或方法）称为C的创建者。对象的属性是该对象的可见属性。 对于任何给定的对象，所有属性及其相应值的集合称为对象的状态。 返回C中对象的属性的值而没有任何状态改变的操作（或方法）被称为C的观察者。改变C中对象的状态的操作（或方法）被称为C的构造函数或变换器 特别地，对象的构造函数可以在通过公理进行变换之后保持正常形式，而变换器将最终通过一些重写规则被消除，并且不能以正常形式出现。

对于任何操作或方法\_.f（\_，\_，...，\_）：C C1 C2 ... Cn→D，我们将C称为f的输入类，D作为f的输出类， C1，C2，...，Cn作为f的参数类别。 令f和g是运算（或方法）。 如果f的输出类与g的输入类相同，我们说g适用于f，而f.g在语法上是有效的。 令f是基项u（或方法序列u中的最后一个方法）中的最后一个运算，g是基项v（或方法序列v中的第一个方法）中的第一个运算。 如果g适用于f，我们说v适用于u，u.v在语法上是有效的。

类C上的可观察上下文或者是观察者，或者是构造函数的一个句法上的有效序列，或者是C中以观察者结束的变换器。

考虑给定类的规范。 当且仅当它们都具有相同的正态形式时，两个基本项u1和u2被认为是正常等价的（由“u1〜nor u2”表示）。 当且仅当1）u1.oc和u2.oc在任何适用的可观察到的上下文oc的观察等价时，两个基本项u1和u2被认为是观察等价的（由“u1〜obs u2”表示），以及2）u1和 u2在没有适用的可观察上下文oc时通常是等价的。 （注意这个定义是递归的）当且仅当1）u1.ob和u2.ob在任何适用的情况下观察等价时，两个基本项u1和u2被认为是属性等价的（由“u1〜att u2”表示） 观察者ob，以及2）当没有适用的观察者ob时，u1和u2通常是等价的。 从定义来看，正态等价是观察等价的特殊情况，这是属性等价的一种特殊情况。

考虑规范和给定类的实现。 当且仅当1）O1.oc和O2.oc是任何适用的可观察上下文oc的观察等价对象时，两个对象O1和O2被认为是观察等价的（由“O1≈obsO2”表示），以及2）当没有适用的可观测上下文oc时，O1和O 2是相同的对象（由O 1 = O 2表示）。 当且仅当1）O1.ob和O2.ob是任何适用的观察者ob的观察等价对象时，两个对象O1和O2被认为是属性相等的（由“O1≈attO2”表示），以及2）O1和O2 当没有适用的观察者ob时是相同的对象。

如果基项u中的最后一个操作的输出类是原始类型（即，它对应于实现语言中的内置类型，例如C ++中的int），则u的常规形式是常数k 的原始类型。 在这种情况下，我们在说明书中将“u〜nor k”，“u〜obs k”和“u〜att k”写为“u = k”，其中“=”表示基本类型中的相等。 类似地，在实现中将“O1≈obsk”和“O1≈attk”写为“O1 = k”。

给定类C的代数规范，假设oci（i = 1,2，...，n）是C上的可观测上下文，并且每个ocj（j = 2,3，...，n）适用于ocj -1。 然后，序列oc1.oc2 ... ... cn在C上被称为可观察的上下文序列（或简称为oc序列），并且其长度被称为n。 如果ocn的输出类是一个基本类型（即在实现语言中的内置类型），则oc1.oc2。... .ocn被称为C上的基本oc序列。

代数规范中的模块可以导入已知的类以支持指定类的功能需求。 我们说，当且仅当每个oc序列具有有限长度并且可以在有限数量的步骤中扩展到原始oc序列时，类的规范具有适当的导入。 我们说，当且仅当Sp中的每个操作f通过P中的唯一方法Θ（f）来实现时，类的实现P相对于代数规范Sp完成，使得输出类或类型的Θ（f ）与f的输出类或类型一致，P中的常数与Sp中的常数一致。

对于完整的实现，给定Sp中的任何基项u，我们使用Θ（u）来表示由执行对应于u的实现的方法序列导致的唯一对象。 特别地，对于完全实现，Sp中的任何常数k对应于P中的常数k。我们写Θ（k）= k。

给定具有适当导入和完全实现P的类的规范Sp，我们定义P相对于Sp if是正确的，当且仅当满足以下两个标准时：

1、等价准则。对于任何观察等价项u1和u2对，从执行相应实现的方法序列得到的对象Θ（u1）和Θ（u2）在观察上是等价的。也就是说，（∀u1）（∀u2）（（u1〜obs u2）→（Θ（u1）≈obsΘ（u2）））。在这种情况下，我们还说P满足在Sp中指定的所有观测上等价基项对。

2.非等价标准。对于任何观察上不等价的基项对v1和v2，从执行相应实现的方法序列得到的对象Θ（v1）和Θ（v2）在观察上是不等价的。也就是说，（∀v1）（∀v2）（¬（v1〜obs v2）→Θ（v1）≈obsΘ（v2）））。在这种情况下，我们还说P满足在Sp中指定的所有观测非等价基项对。

我们说如果任一上述标准被否定，即如果满足以下条件之一，则显示P相对于Sp的失败：

1'。等价失败标准。 （∃u1）（∃u2）（（u1〜obs u2）∧（Θ（u1）≈obsΘ（u2）））。

2'。非等价标准。 （υv1）（∃v2）（¬（v1〜obs v2）∧（Θ（v1）≈obsΘ（v2）））。

等价和不等式标准之间的新关系

基于代数规范的类级测试涉及两个方面，即观测上等价基项的测试和观测上非等价基项的测试。 两个重要的问题马上出现：这两个方面之间的关系是什么？ 一个对另一个覆盖程度如何？

以前的工作，如Doong和Frankl [16]，Gaudel [18]，Chen  [10]和Zhu [32]认为，成功的彻底测试一方面不需要成功测试另一方面。 正如高德尔[18]所指出的，“ExhaustSp [所有公理的所有实例的集合]的定义来自于满足的概念[21]。 然而，它不完全对应于代数规范的初始语义，因为不测试不等式：它反而对应于宽松的语义。 ...在[16]中，提到了一个更详尽的测试集，其中包括对于每个基项与其他正则形式的不等式，定义初始语义。“[32]进一步认识到，”[Doong and Frankl's ]最重要的贡献…是测试用例的扩展，包括阴性测试用例，它包括两个基项，应该产生非等价的结果。

以下是一个代表这一理念的直观例子。

例2（改编自Doong和Frankl [16]和Chen等人[10]）。 假设没有一个操作改变故障实现中的对象的状态。 给定任何两个观察等价的基项，实现返回的相应对象当然是等价的。 直观地，仅通过测试等价项不能揭示故障。 因此，可以得出结论，观察性非等价基项的测试可能是必要的，不能被忽略。

虽然示例2看起来直观有效，但实际上并不总是这样。 我们可以构造一个简单的反例，如下：

示例3.再次考虑示例1中的IntStack类的代数规范。假设本规范的实现如下，其中array [1]是堆栈的顶部，array [100]是底部。

# include <iostream> # define SIZE 100 # define NIL 0 class intStack {

int array[SIZE];

public: ... };

... void intStack :: newStack() {

for (int j = 1; j <= 100; j ++) array[j] = NIL;

} void intStack :: push(int N) {

for (int j = 100; j > 1; j −−) array[j] = array[j −1];

array[1] = NIL; /\* There is a fault in the above statement. \*/ /\* It should be "array[1] = N;". \*/

} void intStack :: pop () {

for (int j = 1; j < 100; j++) array[j] = array[j + 1];

array[100] = NIL; }

int intStack :: top () { return array[1];

}

作为所实现的方法push（int N）中的fault的结果，实现类中所有对象的状态总是（NIL，NIL，...，NIL），满足示例2的“没有操作改变对象的状态”的前提条件。但是，实例2的“仅通过测试等价项不能揭示故障”的结论是不正确的，因为可以这样构造一对故障揭示等价项：

用在公理a5：S.push（N）.top = N中的任何值（例如6）的“new”和N替换S，我们获得以下等价项的基本对：new.push（6）.top〜6。

根据上述实施，这两个术语返回不同的可观察结果，即，“nil”和“6”。因此，这个基本对显示由于所实现的方法push（int N）中的故障而导致的故障。因此，反驳实施例2。

按照软件测试中的标准做法，我们使用术语“fault”表示程序中的不正确指令，术语“failure”表示不正确的执行结果。 在示例3中，例如，array [1] = NIL是fault，Θ（new.push（6）.top）≈obsΘ（new.push（8）.top）是由上面fault引起的failure 。通常，fault可能导致多个failure。 例如，Θ（new.top）≈obsΘ（new.push（4）.top）是另一个由于同一fault而导致的failure。 我们还注意到，在示例3以及后面的示例4和5中，我们假设在规范中的类型Int的实现语言中存在内置类型int。 这个假设是合理的。

一方面，例3用作例2的反例。另一方面，例3仅显示一个特定情况，其中给定观察上非等价对，其揭示failure，存在观察上等价对，其也揭示由于同一fault导致的failure。 为了采取更广泛的观点，让我们先考虑一个稍微更一般的例子4，然后跟进广义的引理，定理和证明。

示例4.再次采用示例1中的IntStack类的代数规范和示例3中的实现。考虑一对基项new.push（1）.push（3）.push（5）和new.push（2）.push（3）.push（5）。它们具有以下特性：

（a）在规范中有一个可观察的上下文，即pop.pop.top，它操作两个基项以给出不同的结果：

new.push（1）.push（3）.push（5）.pop.pop.top = 1（4.1）

new.push（2）.push（3）.push（5）.pop.pop.top = 2（4.2）

因此，两个原始基项在观测上是不等价的。

（b）基于本说明书中的两个基项，将执行实施例3中的实现中的两个方法序列，

Θ（push（1））Θ（push（3））。Θ（push（5））和

Θ（new）.Θ（push（2））。Θ（push（3））。Θ（push（5））。

由于执行对应于两个基项的这些方法序列而产生的两个对象在观察上是等价的，因为它们都在观测上等同于Θ（new）。因此，两个基项和

Θ（new）.Θ（push（1））.Θ（push（3））.Θ（push（5）).

Θ（pop）.Θ（pop）.Θ（top）= 0（4.3）

Θ（new）.Θ（push（2））.Θ（push（3））.Θ（push（5）). Θ（pop）.Θ（pop）.Θ（top）= 0（4.4）

（c）规范中（4.1）和（4.2）中至少一个右边的值与实施的（4.3）和（4.4）中的右边的值不一致。在该特定示例中，Θ（1）= 1和Θ（2）= 2，它们都不同于0。在不失一般性的情况下，考虑（4.1）。它可以被认为是一对观察上等价的基项

new.push（1）.push（3）.push（5）.pop.pop.top和1。

但是，它们对应的对象是不等价的：

Θ（new）.Θ（push（1））.Θ（push（3））.Θ（push（5）). Θ（pop）.Θ（pop）.Θ（top）= 0。但是 Θ（1）≠ 0。

因此，它们还揭示了由于相同fault的failure。

总之，基于给定的一对观察上非等价的基项new.push（1）.push（3）.push（5）和new.push（2）.push（3）.push ，我们已经确定了一对观察等价的基项new.push（1）.push（3）.push（5）.pop.pop.top和1，也揭示了由于同一fault引起的failure。

让我们也考虑相反的情况。

示例5.再次采取示例1中的IntStack类的代数规范。考虑另一个错误实现，其中pop作为一个null操作实现，不执行任何操作，而其他操作正确实现。一对观察上等价基项new.push（1）和new.push（1）.push（3）.pop显示失败，因为它们在观察上等价，但在给定的实现中产生不同的对象。它们还表现出以下性质：

（a）在规范中有一个可观察的上下文，即top，它操作两个基项以给出相同的结果

new.push（1）.top = 1（4.5）

new.push（1）.push（3）.pop.top = 1（4.6）

（b）执行new.push（1）和new.push（1）.push（3）.pop中的操作（即Θ（new）.Θ（push（1） ））和Θ（new）.Θ（push（1））.Θ（push（3））.Θ（pop））观察上是不等价的，因为在实现中相应的可观察上下文操作两个对象给出不同结果

Θ（new）.Θ（push（1））.Θ（top）= 1（4.7）Θ（new）最佳）

= 3（4.8）

（c）实施的（4.7）和（4.8）中的至少一个右边的值与（4.5）和（4.6）中的右边的值不一致。在此特定示例中，（4.8）与规范不一致。让我们看一对观察上非等价基项new.push（1）.push（3）.pop.top（即，左手（4.6））和3.它们的相应对象Θ（new）.Θ（push（1））.Θ（push（3））.观察上等价于（4.8）。因此，他们也显示由于相同fault的failure。

简而言之，基于给定的一对观察上等价基项new.push（1）和new.push（1）.push（3）.pop显示失败，我们可以识别一对观察上非等价基项。 push（1）.push（3）.pop.top和3也显示由于相同的fault的failure。

是否有一个特定类别的规范和实现，如果一个观察到的非等值基项对显示由于某个fault的failure，那么存在一个观察上等价的基项，也将显示由于同一fault的faiure，反之亦然吗？ 也就是说，在什么条件下，观测上等价基项和观测上非等价基项的检验是否相互覆盖？ 以下引理和定理有助于回答问题。

引理1.给定一个具有适当导入和完全实现P的类的规范Sp，如果P满足Sp中指定的所有观测上等价基项对，它也将满足Sp中规定的所有观测上非等价基项对。 正式地，给定这样的规范和实现，（∀u1）（∀u2）（（u1〜obs u2）→（Θ（u1）≈obsΘ（u2）））

导出（∀u1）（∀u2）（¬（u1〜obs u2）→Θ（u1）≈obsΘ（u2）））。(4.9)

以下是引理1的正式证明。上面的例子4说明了它的思路

证明。我们通过反证法证明了引理。假设（4.9）的左边是真，但是右边是假，即，（∃u1）（∃u2）（¬（u1〜obs u2）∧（Θ（u1）≈obsΘ （u2）））。在这种情况下，Θ（u1）≈obsΘ（u2）由一个实现fault引起，我们将用f表示。

由于¬（u1 ~obs u2）和给定类有适当的导入，根据观察等价约束的定义和适当导入的定义，在规范上存在oc序列ocs，使得观察者的输出类ocs的结束是原始类型，并且u1.ocs≠u2.ocs.3因此，存在原始类型的两个值k1和k2，使得k1≠k2，u1.ocs = k1和u2.ocs = k2。当实现完成时，存在实现ocs的唯一方法序列Θ（ocs）。由于类具有适当的导入并且实现完成，Θ（ocs）也必须以原语类型结束，使得对于原语类型的某个值k，Θ（u1）.Θ（ocs）= k。因为Θ（u1）≈obsΘ（u2），根据观察等价≈obs的定义，我们有Θ（u2）.Θ（ocs）=Θ（u1）。

Ask1≠k2，wemusthavek≠k1 ork≠k2。不考虑一般情况，假设k≠k1。在这种情况下，我们有u1.ocs = k1，但Θ（u1）.Θ（ocs）= k≠k1。

令Θ（u1.ocs）表示对应于指定的操作序列u1.ocs的实现的方法序列的执行结果。由于实现完成，我们有Θ（u1.ocs）=Θ（u1）.Θ（ocs）= k≠k1。令Θ（k1）表示k1的执行结果。因为k1是基本类型的值，所以我们有Θ（k1）= k1≠k。基于这些关系，我们获得u1.ocs = k1，但是Θ（u1.ocs）≠Θ（k1）。

根据观察等价的定义，由于u1.ocs结束时的操作的输出类是原始类型，u1.ocs = k1表示u1.ocs〜obs k1和Θ（u1.ocs）≠Θ（k1 ）意味着¬（Θ（u1.ocs）≈obsΘ（k1））。 （参见第3节中原语类型的基本概念）。因此，我们有（u1.ocs〜obs k1）∧¬（Θ（u1.ocs）≈obsΘ（k1））。这与（4.9）的左边相矛盾。

我们从上述注意到，从由fault f导致的Θ（u1）≈obsΘ（u2）导出¬（Θ（u1.ocs）≈obsΘ（k1））。换句话说，Θ（u1.ocs）≈obsΘ（k1）和Θ（u1）≈obsΘ（u2）是由于相同的fault f。

引理1的逆命题也是真的，如下所述：

引理2.给定一个具有适当导入和完全实现P的类的规范Sp，如果P满足Sp中规定的所有观测非等价基项，它也将满足Sp中规定的所有观测等价基项。 正式地，给定这样的规范和实现，（∀u1）（∀u2）（¬（u1〜obs u2）→Θ（u1）≈obsΘ（u2））） u1）≈obsΘ（u2）））。 （4.10）

以下是引理2的正式证明。上面的例子5说明了它的思路。

证明。我们还通过反证法证明了引理。假设（4.10）的左边是成立的，但是右边是假的，也就是说，（∃u1）（∃u2）（（u1-obs u2）∧¬（Θ（u1）≈obsΘ u2）））。

由于¬（Θ（u1）≈obsΘ（u2））并且给定类具有适当的导入，根据观察等价≈obs的定义和适当导入的定义，在实现中存在oc序列Θ（ocs）以使得在Θ（ocs）的结尾处的观察者的输出类是原始类型，并且Θ（u1）·θ（ocs）≠Θ（u2）·θ（ocs）.4换句话说，Θ ）θ（ocs）= k1和Θ（u2）.Θ（ocs）= k2，使得k1≠k2。

当实现完成时，存在由Θ（ocs）实现的唯一操作序列ocs。由于类具有适当的导入并且实现完成，ocs还必须以原语类型结束，使得对于原语类型的某个值k，u1.ocs = k。因为u1〜obs u2，根据观察等价约束的定义，我们有u2.ocs = u1.ocs = k。由于k1≠k2，wemusthavek≠k1 ork≠k2。不考虑一般情况，假设k≠k1。在这种情况下，我们有u1.ocs = k≠k1，但是Θ（u1）.Θ（ocs）= k1。令Θ（u1.ocs）表示实现指定的操作序列u1.ocs的方法序列的执行结果。由于实现完成，我们有Θ（u1.ocs）=Θ（u1）.Θ（ocs）。令Θ（k1）表示k1的执行结果。因为k1是基本类型的值，所以我们有Θ（k1）= k1。基于这些关系，我们获得u1.ocs≠k1，但是Θ（u1.ocs）=Θ（k1）。换句话说，¬（u1.ocs\_obs k1）∧（Θ（u1.ocs）≈obsΘ（k1））。这与（4.10）的左手相矛盾。

类似于引理1的证明的结尾处的注释，Θ（u1.ocs）≈obsΘ（k1）和¬（Θ（u1）≈obsΘ（u2））由于相同的fault。

将Lemmas 1和2放在一起，我们立即得到下面的定理。

定理1.给定一个具有适当导入和完整实现的类的规范，有以下两个标准：

1、等价标准：

（∀u1）（∀u2）（（u1〜obs u2）→（Θ（u1）≈obsΘ（u2）））

2、非等价标准：

（υv1）（νv2）（¬（v1 ~obs v2）→Θ（v1）≈obsΘ（v2）））

暗示彼此。 因此，当且仅当我们能够表明满足a或b时，实现满足规范。我们可以表达定理1（∀u1）（∀u2）（（u1〜obs u2）→（Θ（u1）≈obsΘ（u2）））⇔（∀v1）（∀v2）（¬（v1 ~obs v2）→Θ（v1）≈obsΘ（v2）

令OE表示所有对的观测等价基底项的集合，OE'表示所有观察非等价基底项对的集合。 从（4.11）可以得出（∀{u1，u2}∈OE）（Θ（u1）≈obsΘ（u2））⇔（∀{v1，v2}∈OE'）（Θ（v1）≈obsΘ（v2）））。 （4.12）

然而，这对于平均软件测试者来说可能太正式。 为了简洁起见，我们将在本文中定义“P满X”的概念如下：

定义1（P满足X）。

a. 给定属性等价基项对的任何集合AE，“P满足AE”意味着对于AE中的任何一对项u1和u2，它们的对应实现Θ（u1）和Θ（u2）属于相等的。也就是说，（∀{u1，u2}∈AE）（Θ（u1）≈attΘ（u2））。

b. 给定除了属性等价之外的指定类型的等价基项对的任何集合X，“P满足X”意味着对于X中的任何基项对u1和u2，其对应的实现Θ（u1）和Θ（u2）当量。也就是说，（∀{u1，u2}∈X）（Θ（u1）≈obsΘ（u2））。

C. 给定属性非等价项对的任何集合AE'，“P满足AE'”意味着对于AE'中的任何一对基项v1和v2，它们对应的实现Θ（v1）和Θ（v2）属于非等价的。那是，

（∀{v1，v2}∈AE'）（Θ（v1）≈attΘ（v2）））。

d. 给定任何指定类型的除属性非对等性之外的非对等项对的任何集合X’，“P满足X'”意味着对于X中的任何对术语v1和v2，其对应的实现Θ（v1）和Θ（v2）是观察性不等价的。也就是说，（∀v1，v2∈X'）（Θ（v1）≈obsΘ（v2）））。

属性等价在定义1a中被单独处理，因为它太强而不需要u1和u2在规范中属于等同的，但Θ（u1）和Θ（u2）在实现中是观察等价的。 读者可以参考[10，定理2和3]和[10，注（f），p。 78]。 为了统一风格，我们在定义1c中分别处理属性非等价性。基于定义1中的符号，我们可以简单地把（4.12）写为（P满足OE）⇔（P满足OE'）。 （4.13）这是一个重要的结果，因为OE和OE'直观上不是彼此的子集。 事实上，OE∩OE'=Ø。在第6节中，我们将进一步扩展（4.13）以前的相关工作提出的其他形式的“P满足X”，例如“P满足FP”，“P满足CI”，“P满足GI” ，“P满足NE”，“P满足AE”，“P满足AE”。

等价和不等价标准的互换性

在本节中，我们将讨论在面向对象软件的类级测试的最后一节中的新结果的含义。

取定理1中的标准a和b的否定，我们得到：

推论1.给定一个具有适当导入和完整实现的类的规范，有以下两个标准

1. 等价failure标准：（∃u1）（∃u2）（（u1〜obs u2）∧¬（Θ（u1）≈obsΘ（u2））

b. 非等价failure标准： (∃v1)(∃v2)(¬(v1 ∼obs v2) ∧ (Θ(v1) ≈obs Θ(v2)))

暗示彼此。

推论2.给定具有适当导入的类的规范，假设其实现完成。 如果一对非等价基项¬（v1〜obsv2）揭示由于fault f引起的failure，则存在一对等价基项（u1〜obs u2），其也将揭示由于相同fault f而导致的failure，反之亦然。 换句话说，观测上等价基项的测试和观测上非等价基项的测试彼此覆盖。

证明。 如果一对非等价项¬（v1〜obs v2）显示failure，根据非等价failure标准，我们有Θ（v1）≈obsΘ（v2）。 类似于引理1的证明，我们具有（v1.ocs〜obs k1）和¬（Θ（v1.ocs）≈obsΘ（k1）），其中，Θ（v1.ocs）≈obsΘ ））和Θ（u1）≈obsΘ（u2）是由于相同的fault f引起的。 因此，一对等价基项（v1.ocs〜obs k1）也揭示了由于相同fault f引起的failure。 相反的证明是类似的。

推论3.给定具有适当导入的类的规范，假设其实现完成。 令OE为所有观测上等价基项对的集合，OE'为所有观测上非等价基项对的集合。 给定任何有限测试套OEs’ ⊂ OE’，存在有限测试套件OEs ⊂ OE ，使得对于由OE中的测试用例所显示的任何failure（由fault导致的），failure（由相同的fault导致的）也可以 在OEs中的测试案例中显示。 相反，对于任何有限测试套件OEt⊂OE，存在一个有限测试套件OEt’ ⊂ OE’ ，使得对于OEt中的测试用例所显示的任何failure（由某个确定的fault导致的），failure（由相同的fault导致的 ）也可以通过OEt'中的测试用例来揭示。 因此，我们可以通过观测等价基项对的有限测试套替换观测非等价基项对的有限测试套，同时揭示由于相同fault导致的failure，反之亦然。

证明。 对于任何failure（由fault引起的，例如f）由OE'中的非等价对oe'所揭示，通过推论2，存在一个等价对oe，其也将显示由于相同fault f引起的failure。 让OEs成为所有这样的oe的集合。 由于OEs'是有限的，并且OE中的每个测试用例可以揭示最多一个fault，因此OE也是有限的并且满足后条件。 相反的证明是类似的。除了正式的证明之外，读者还可能对OEs如何实际构建感兴趣。 这可以通过以下过程实现：

过程1.给定具有适当导入的类的规范，假设其实现完成。进一步假设存在可以通过观察性非等价基项对的有限测试套件 OEs’ (⊂ OE’)揭示的failure集合F.以下过程显示存在另一个观测上等价基项对的有限测试套件 OEs (⊂ OE) ，使得对于F中的每个failure（由于某个确定的fault引起），在OEs中存在也可以显示由同一fault引起的failure。

Procedure {

Read OEs’;

OEs = Ø;

For each oe’ ∈ OEs’ do {

/\* Each oe’ should be of the form ¬(u1 ∼obs u2). Each oe’ reveals a failure in F. \*/

/\* Under such oe’, Θ(u1) ≈obs Θ(u2). \*/ There exists an oc sequence ocs on the

specification such that the output class of the observer at the end of the ocs is a primitive type and u1.ocs ≠ u2.ocs;

Suppose u1.ocs = k1 and u2.ocs = k2 such that k1 ≠ k2;

There exists a unique method sequence Θ(ocs) that implements ocs;

If Θ(u1).Θ(ocs) = k, as Θ(u1) ≈obs Θ(u2), we should have

Θ(u2).Θ(ocs) = Θ(u1).Θ(ocs) = k;

Ask1 ≠k2,wemusthavek≠k1 ork≠k2; Without loss of generality, suppose k ≠ k1; Thus, we have u1.ocs = k1

but Θ(u1).Θ(ocs) = k ≠ k1;

Let Θ(u1.ocs) denote the execution result of

implemented method sequence

corresponding to u1.ocs;

We have Θ(u1.ocs) = Θ(u1).Θ(ocs) = k ≠ k1; Thus, we obtain u1.ocs = k1

but Θ(u1.ocs) ≠ Θ(k1) = k1,

that is, (u1.ocs ∼obs k1) ∧ ¬(Θ(u1.ocs) ≈obs Θ(k1)); Denote the pair of equivalent terms

(u1.ocs ∼obs k1) by oe; Set OEs = OEs ∪ {oe};

}

Write OEs; }

总之，给定一个具有适当导入和完整实现的类的规范作为用于测试用例选择的候选集合，所有观察上非等价基项对的无限集合可以被所有观察上等价基项对的无限集合代替，同时揭示由于相同的fault导致的failure，反之亦然。 此外，作为测试套件，给定的有限一组观察上非等价基项对可以被一组有限观测等价基项对代替，同时揭示由于相同fault引起的failure，反之亦然。 当然，我们不应该期望所有观察上非等价（或等价）基项对的无限集合可以由观察上等价（或不等价）基项对的有限集合代替，同时揭示由于相同fault引起的failure。

关于基于代数规范的类级测试的前期工作的理论影响

在本节中，我们展示了我们目前工作的理论影响和贡献与以前基于代数规范的面向对象软件测试的研究。我们将在下一节中介绍实际影响。

（a）在以前的工作，如Aiguier et al。 [1]，Bernot et al。 [3]，[4]，Le Gall和Arnould [26]，Machado [27]，和Machado和Sannella [29]，作者定义了程序P相对于规范Sp是正确的当且仅当P满足Sp中每个公理的所有基本实例的集合时。为了简洁起见，我们将使用GI来表示这组所有这些基本实例。虽然上述作者将GI称为“详尽的测试集”，Gaudel [18]指出“它不完全对应于代数规范的初始语义，因为不等式没有被测试：它相当于宽松的语义。

（b）Aiguier 等人[1]证明，给定规范Sp，如果P满足只包含创建者或构造函数（但不包括变换器）的Sp中每个公理的所有基本实例的集合，则P满足GI。我们将使用CI来表示每一个只包含创造者或构造函数的公理的所有ground实例的集合。换句话说，Aiguier et al。证明了（P满足CI）（P满足GI）。另一方面，我们注意到任何公理的基础实例可能或可能不只包含创作者或构造函数。因此，

CI⊂GI，其中“⊂”表示“是的正确子集”。换句话说，CI≠GI。它从（6.1）和定义1b得出

（P满足GI）（P满足CI）。 （6.2）

（c）在1994年，Doong和Frankl [16]定义了当P满足Sp中所有“等价”基项对的集合和Sp中所有“非等价”基项对的集合时，P相对于Sp是正确的，其中两个基项被称为是“等价的”当且仅当第一个可被重写为第二个时。直观地，新定义要求P不仅应该单独满足每个公理，而且还满足从公理的多个使用作为从左到右重写规则的结果。前者并不一定意味着后者，正如Weyuker的反分解和反分解公理[30]所指出的那样。

我们将使用RP表示所有这样的“等价”基项对的前一组，并且使用RP'表示所有“非等价”基项对的后一组。根据Gaudel [18]，这个“更大的穷尽测试集合[RP∪RP'] ...包括对于每个基项，与初始语义的定义之后的其他正态形式的不等式。”Gaudel和Le Gall [19]这是“一个例子，其中[更大的]详尽的测试集不是从公理的实例化建立的，而是更广泛地从规范的语义后果的足够的集合。”Zhu [32]进一步认识到， “[Doong和Frankl的]最重要的贡献之一是将测试案例扩展为包括负测试案例，其由两项组成，这两项应该产生非等价结果。

当且仅当u1可以使用一个或多个公理作为从左到右重写规则被重写为u2时，在Doong和Frankl [16]中，一对基本项u1和u2被称为是“等价的”。特别地，如果仅使用一个公理，u1和u2将是公理的ground实例。因此，我们有

GI⊂RP，（6.3）其中符号“⊂”再次表示“是的一个适当的子集”。从（6.3）和定义1b可以得出

（P满足RP）（P满足GI）。 （6.4）

我们将在合并（6.16）证明中的所有逻辑关系之后讨论相反。

（d）1998年，Chen等人[9]发现对应于“P满足RP”的Doong和Frankl [16]中的“等价”标准是有问题的。为了解决这个问题，他们定义了正态等价和基本对的概念。当且仅当它们都可以被重写为相同的正态形式时，任何一对基本项u1和u2通常是等价的。每个基本对通过用公式替换公理两侧的所有变量形成。我们将使用NE来表示正常等价对的集合，并且FP表示所有基本对的集合。我们注意到以下属性：

（i）Chen 等人在[9，定理2]中证明，给定一个具有适当导入和完全实现P的类的规范，

（P满足NE）⇔（P满足FP）（6.5）

即使FP仅是NE的正确子集。

 （ii）CI中的每对项通过用仅包含创建者或构造的基本术语替换公理两侧的所有变量来形成。由于每个正常表单只包含创建者或构造函数，因此我们有FP⊆CI。相应地，在[8，命题4]中证明每个包含创造者或构造函数的基本项不一定是正态形式，因此我们有FP≠CI，因此

FP⊂CI。(6.6)

它来自（6.6）和定义1b

(P satisfies CI) 推出符号(P satisfies FP). （6.7） 我们还将在合并（6.16）证明中的所有逻辑关系之后讨论相反。

（e）2001年，Chen 等人[10]表明，对应于“P满足RP和P满足RP'”的Doong和Frankl [16]中的“等价和不等价”标准是有问题的。给定任何一对基本项u1和u2，如果u1可以重写为u2，则它们在RP的意义上被称为“等价”。然而，由于u2在许多情况下不能被重写为u1，所以在RP的意义上它们不是“等同的”，因此给出矛盾的判断。更严重地，当实现正确时，在示例1的规范下的术语new.push（1）.push（3）.pop和new.push（5）.pop.push（1）产生观察等价的对象。然而，在[16]中，它们不是“等价”的意思。因此，他们错误地报告failure。

如[10，定理1]所证明的，在RP意义上的“等价”意味着正常的等价，但反之亦然。因此，我们有

RP⊂NE。 （6.8）

它来自（6.8）和定义1b

（P满足NE）（P满足RP）。（6.9）

在合并（6.16）证明中的所有逻辑关系之后，我们将再次讨论相反。

（f）Chen等人 [10]进一步定义，当且仅当P满足Sp中所有观察上等价基项对的集合（OE）和Sp中所有观察上非等价基项对的集合（OE'）时，P相对于Sp是正确的。令AE表示所有属性等价基项对的集合，AE'表示所有属性非等价基项对的集合。 Chen 等人证明在[10，定理1，3和4]

NE⊂OE⊂AE和AE’⊂OE'，（6.10）

（P满足AE）⇔（P满足OE）⇔（P满足NE），（P满足OE'）⇔（P满足AE'）。 （6.11）

（g）在本文的定理1中，当且仅当“P满足OE”时，通过数学证明“P满足OE”，我们进一步连接（6.11）和（6.12）是用于验证“P满足OE∪OE”的一般程序正确性标准的必要且充分的条件。换句话说，尽管

OE⊂OE∪OE'，我们有

（P满足OE∪OE'）⇔（P满足OE）⇔（P满足OE'）。

将子集关系（6.1），（6.3），（6.6），（6.8），（6.10）和（6.13）结合在一起，对于具有适当导入和完全实现的类的规范说明，

FP⊂CI⊂GI⊂RP⊂NE⊂OE⊂AE

和OE⊂OE∪OE'。

取逻辑关系式（6.2），（6.4），（6.5），（6.7），

（6.9），（6.11），（6.12）和（6.14），对于具有适当导入和完全实现的类的规范规范，我们有

（P满足OE∪OE'）

（P满足OE）⇔（P满足OE'）⇔（P满足AE）⇔（P满足AE'）⇔（P满足NE） FP）⇔（P满足NE）。

注意，上述语句的行6中的从“4（P满足NE）”到“（P满足NE）”的关系形成一个周期。我们得出结论

（P满足OE∪OE'）

（P满足OE）⇔（P满足OE）⇔（P满足AE）⇔（P满足AE）⇔（P满足NE）⇔（P满足RP） ⇔（P满足CI）

总之，早期工作通过GI定义程序正确性，其中仅包括用于验证等价基项的测试用例子集，并且不包括用于验证非等价基项的任何测试用例。 Doong和Frankl通过将GI扩展到RP∪RP'来增强语义，其中包括用于验证等价和非等价基本项的测试用例子集。 Chen等人通过用OE∪OE’替换RP∪RP'来进一步改进语义。它们还证明，给定具有适当导入和完全实现的类的规范，等价标准“P满足OE”，非等价标准“P满足OE”可以在具有等价性和非等价性的项中表示，这在现实世界的实践中可以更容易地验证。在本文中，我们进一步证明，等价标准可以由非等价标准代替，反之亦然，同时揭示由于相同的fault导致的failure。换句话说，“P满足OE”或“P满足OE”将是必要的并且足以确认“P满足OE∪OE”。

有关视觉概要，请同时参考表1。从表中，我们看到研究人员已经走了很长的路，因为根据“P满足GI”的程序正确性的第一个提议。

通常，当且仅当程序P满足其规范的要求时，程序P才是正确的。在代数规范Sp的情况下，要求包括两个方面：

程序P必须满足Sp中的每个公理，

程序P必须满足从Sp中公理得出的所有后果。

OE∪OE'是所有这些后果的集合。因此，当且仅当P满足由早期作者定义的GI时，程序P的正确性的定义仅考虑方面1。然而，当且仅当P满足OE∪OE'（在我们前面的工作中定义）时，程序P的定义不仅考虑方面1而且还考虑方面2.由于GI仅仅是OE∪OE'的一个适当子集， ，我们有（P满足OE∪OE'）（P满足GI），但是相反不一定是真的。在什么条件下，反之亦然？其他作者没有调查这个问题，而（6.16）及其在我们当前的论文中的证明表明，给定一个具有适当导入和完整实现的类的规范规范，反之亦然。在该特定条件下，根据“P满足GI”的先前关于程序正确性的提议在理论上可以被保留。类似的论证也适用于FP，CI，RP，NE，AE和AE'。

给定（6.16），是否意味着任何这些标准需要在现实世界实践中软件测试人员的相同数量的测试工作？答案不是那么简单。我们将在下一节讨论（6.16）的实际影响。

基于代数规范的类级测试的实践意义

**7.1测试案例选择中的实践意义**

本节讨论由于观测等价基项对的测试和观测非等价基项对的测试之间的新关系所产生的实际影响。 为简洁起见，我们将前者称为OE的测试，后者称为OE'的测试。 一般来说，OE的测试和OE’的测试之间的关系可以落入以下任一可能性中：

可能性1：OE的测试和OE的测试'不暗示着彼此。这是以前的工作的假设，如Doong和Frankl [16]，Gaudel [18]，Gaudel和Le Gall [19]，Chen et al。 [10]和朱[32]。在这种观点下，通过OE的测试导致failure检测的fault可能或可能不会导致由OE'的测试可检测到的failure，反之亦然。因此，给定任何实现fault，它可以

a.导致通过OE的测试可检测到的failure，但不会导致通过OE'的测试可检测到的failure，或

b。导致通过OE'的测试可检测到的failure，但不会导致通过OE的测试可检测到的failure，或

C。导致由OE的测试可检测到的故障，以及导致OE'的测试可检测的另一故障。

因此，必须选择测试用例来验证a，b和c。这类似于传统程序测试中的分区测试方法。可以应用诸如比例采样策略[12]的技术，其中用于测试a，b和c的选定数量的测试用例与它们的相对输入域大小成比例。如果可能性1确实是真的，则我们需要面对的困难是a，b和c的相对输入域大小不容易估计以便应用比例采样技术。幸运的是，本文的主要定理证明，给定一个具有适当导入和完全实现的类的规范，可能性1不是真的。

可能性2：OE的测试和OE'的测试意味着彼此。 在这种情况下，导致由OE的测试可检测的failure的fault必须导致通过OE'的测试可检测到的failure，反之亦然。 作为本文的主要贡献，定理1证明，给定一个具有适当导入和完全实现的类的规范，只有可能性2是真实的情况。 在这种情况下，如果我们选择测试用例来验证OE和OE'，它可能会导致冗余，降低测试的有效性和效率。 因此，我们应该为OE或OE’选择测试用例，而不是两者都选择。

**7.2验证对象的观察等价性和非等价性的实践意义**

如我们在第6节中所看到的，各种研究者已经提出了不同的标准来测试程序P相对于规格Sp的正确性，包括P是否满足OE∪OE'，P是否满足OE'，P是否满足OE， P满足NE，P是否满足RP，是否满足GI，是否满足CI，是否满足FP，是否满足AE以及是否满足AE'。 我们在7.1节中基于定理1的分析消除了验证P是否满足OE∪OE'的需要。 我们在第6节（6.16）的证明还表明，给定一个具有适当导入和完全实现的类的规范，所有上述标准在理论上是等价的。 然而，这是否意味着任何这些标准都需要在现实世界中软件测试人员的相同数量的测试工作？ 本节进一步分析这个问题，并提出一个切实可行的选择。

基于第4节中的定义1，上述标准可以分为三个不同的类别：

1、对于“是否P满足X”的形式的标准，其中X是OE，NE，RP，GI，CI和FP，我们需要测试对于X中的任何一对项u1和u2， Θ（u1）和Θ（u2）在观察上是等价的。实际上不可能验证真实世界软件测试中的观察等价，因为每个测试用例涉及潜在可观察上下文的无限集合。这些标准在实践中不是理想的选择。虽然我们在[9]，[10]中提出我们测试P是否满足FP，这是最小的集合

在（6.15）中，我们需要一个启发式白盒技术ROCS [11]来选择可观察上下文集合的相关有限子集，以确定对象的观察等价性。

2、对于“P是否满足OE”的标准，“我们需要测试对于OE中的任何一对项u1和u2，它们相应的实现Θ（u1）和Θ（u2）在观察上是不等价的。在现实世界测试中发现观察到的非等价实际上是困难的，因为我们需要为每个测试用例经历可观的上下文的无限集合。因此，这个标准也不是一个实际的选择。

3、对于“P是否满足AE”和“P是否满足AE”的标准，我们需要测试对于AE中的任何一对项u1和u2，它们对应的实现Θ（u1）和Θ（u2）属性等价和不等价。在现实世界测试中验证对象的属性等价和非对等是很简单的，因为任何类中的属性集是有限的，通常很小。因此，这两个标准是潜在的实际选择。读者可参考[10节，第4.3节推论2后的段落]进行更多分析。

我们已经分析了需要测试“是否满足AE”或“是否满足AE'”或两者，类似于7.1和OE和OE'。我们得出了一个类似的结论，我们应该选择AE或AE的测试案例，但不是两者。特别地，我们建议测试标准“P是否满足AE”，因为已经开发了用于从AE中选择有限数量的测试案例的技术。我们在[10，第4.4节]中生成主要的非等价项（GAN）方法使用状态转换图（STD）中的技术来处理这个过程，并且通过来自用户的交互式输入将其转换成终止过程用于循环的最大迭代次数路径。 GAN方法的实现和实验也在[10，第4节]中讨论。 GAN方法的一个限制是它假设了规则性假设[3]，[4]，即如果已经针对一些常数k测试了正整数1,2，...，k的语句，假定语句将对所有正整数保持。因此，已经测试了用于由用户指定的最大数量的迭代的循环路径，假定实现在任何数量的迭代下是正确的。

总之，基于第7.1节和第7.2节中的实际考虑，给定一个具有适当导入的类的规范规范，并在真实世界软件测试中完全实现，我们建议从集合AE’中选择测试用例，执行相应实现的方法序列的结果归因于不等同，而不是从OE'，OE，NE，RP，GI，CI，FP或AE中选择测试用例。

作为未来的工作，我们还将研究ARTOO [13]的应用，通过定义非等价项的对象距离，从AE中选择测试用例，以便在AE中均匀地扩展测试用例。 这将减轻用户不得不假定正则性假设并且对循环路径的最大迭代次数做出决定。

结论

一般认为，基于代数规范的面向对象软件的类级测试涉及两个独立的方面：等价基项的测试和非等价基项的测试。以前的研究人员引用了直观的例子来说明后者不能被前者替代，因此具有同等重要性。

然而，我们在本文中正式证明，给定一个具有适当导入和完全实现的类的规范，测试观察上等价基项（由“P满足OE”表示）的等价准则和测试观测上非等价基项上的非等价准则（由“P满足OE”表示）彼此暗示。基于这个结果，我们已经表明，观测上等价的基项的测试和观测上不等价的基项的测试彼此覆盖。换句话说，如果由于实施中的某个故fault而导致的failure可以通过测试观察到的非等价基项来揭示，则由于同一故fault导致的另一failure也可以通过测试观测上等价基项来显示，反之亦然。

我们已经讨论了我们的新发现对基于代数规范的类级测试的相关工作的理论含义。我们已经证明，给定具有适当导入和完全实施的规范，由先前研究者提出的所有正确性标准在理论上等同于“P满足OE”和“P满足OE”。在该特定条件下，以前的研究人员提出的程序正确性的标准理论上可以保留。

另一方面，测试“P满足OE”或“P满足OE”的需要在软件测试中是不可能的任务，因为即使对于单个测试情况也需要验证无限数量的行为结果。我们已经讨论了现实世界的影响，并建议使用更实用的标准“P满足AE”进行测试，这是由我们的理论结果保证，以揭示由于相同的fault导致的failure。即使如此，我们需要假设一个规律性假设。作为未来的工作，我们建议研究ARTOO作为替代技术的应用，以减轻这一假设。