中国海洋大学学报
PERIODICAL OF OCEAN UNIVERSITY OF CHINA
51(11):086～093
Nov., 2021

# 一种基于执行路径状态比对的科学计算程序等效性判定方法*

曲海鹏，张敏媛，韩庆迪，林喜军
（中国海洋大学信息科学与工程学院，山东 青岛 266100）

**摘　要：** 科学计算程序的计算等效性判定是科学计算领域的重要问题。科学计算程序在算法改进、程序优化和版本迭代过程中经常对函数进行等价转换，转换前后的同一函数需要确保其计算过程的等效性。针对该问题，提出基于程序路径的状态匹配验证方法——SCEP，通过对程序不同版本在相同的输入空间的路径状态进行约束求解和精确比对，判定其计算的等效性。将SCEP方法与已有的判定方法在多个库函数组成的spcLib测试程序集上进行了对比实验，结果表明SCEP能判定更多的函数转换类型，不仅可用于“Fortran-to-Fortran”源码等效性检查，还能对“Fortran-to-C/C++”源码转换进行等价性判定。该方法有助于为科学计算程序的优化改进过程和向C/C++迁移过程提供完备的计算等效性保证。

**关键词：** 程序等效性验证；科学计算程序；状态匹配；回归验证；语义不变的转换
**中图法分类号：** TP311.5　　**文献标志码：** A　　**文章编号：** 1672-5174(2021)11-086-08
**DOI：** 10.16441/j.cnki.hdxb.20200379
**引用格式：** 曲海鹏，张敏媛，韩庆迪，等. 一种基于执行路径状态比对的科学计算程序等效性判定方法[J]. 中国海洋大学学报(自然科学版)，2021，51(11)：86-93.
Qu Haipeng，Zhang Minyuan，Han Qingdi，et al. A method for equivalence checking of scientific programs based on state comparison of execution paths[J]. Periodical of Ocean University of China，2021，51(11)：86-93.

科学计算程序通常使用Fortran和C/C++编写。其中，Fortran是科学计算的特定领域语言，广泛用于气候科学，天气预报，化学循环反应器，等离子物理等领域[1]，在科学计算程序中占有很高的比例。在20世纪中后期，几乎所有科学计算代码都用Fortran实现。与其他语言相比，Fortran的语言特点使其更适用于科学计算，Fortran在数值计算中的性能优于C++，这也是Fortran被广泛用于科学计算的原因。迄今，科学计算领域已经积累了许多高度优化的数值计算代码，这些程序随着计算方法的改进和计算架构的更新，需要不断进行算法改进、程序优化、架构迁移、版本迭代等维护和更新活动。但是，作为一种面向过程的编程语言，Fortran的后续维护比较困难，维护周期甚至会超过开发新代码的周期，这导致了Fortran代码维护和更新是一项艰巨的任务。而对修改前后的代码能否进行可靠完备的自动化等效性判定，则是阻碍Fortran代码维护和更新的主要困难之一。

对于程序的等效性判定问题，研究者主要在编译器正确性检查、软件伪造鉴定以及回归验证等多个领域进行了研究，并开发了一些工具。但这些工具多是针对特定用途的代码等效性检测，对Fortran程序的研究成果更为少见，因此难以直接用于科学计算程序的等效性判定。目前常用的基于用例测试的判定方法，因其无法覆盖所有程序分支和全部输入条件，判定结果不具有完备性。而基于人工审查的代码等效性检查缺乏严格性，在大规模程序检查中由于复杂度增大更缺乏可行性。

近年来，随着C/C++使用范围的扩大及其在操作系统功能调用和底层硬件操作等方面的优势，经常需要在C/C++和Fortran程序之间进行函数调用操作。在大规模代码的开发过程中，混合调用会降低程序的性能，混合调用还因其编译、链接的环境依赖性显著降低代码的可移植性。因此，研究者将一些Fortran科学计算程序向C/C++进行转换，转换通常由程序员人工完成，并借助于目前已存在的一些自动转换工具，如F2C[2]，F2CPP，FABLE[3]等，但这些转换工具存在较大局限性，实际应用并不广泛。无论是人工转换还是自动转换，转换前后的程序都需要进行等效性判定，而对于Fortran到C/C++的等效性判定问题，目前更是缺乏实用的技术和工具。

* 基金项目：国家自然科学基金项目(41976184)资助
Supported by the National Natural Science Foundation of China(41976184)
收稿日期：2020-12-31；修订日期：2021-04-26
作者简介：曲海鹏(1972—)，男，副教授，研究方向为网络与信息安全。E-mail：quhaipeng@ouc.edu.cn

前人已提出的等效性判定方法主要应用于编译器正确性检查等场景，通常建立在待比较的程序控制流高度相似的前提下，但在科学计算程序的版本更新和语言转换中，待比较程序通常具有控制流不具备高度相似性的特点。本文面向这一问题，研究控制流相似性较低前提下的等效性判别问题，主要解决以下三个方面的挑战：被比较的两个分支条件的约束可能在形式上有所不同，并且在大多数情况下，无法通过 SMT (Satisfiability Modulo Theories)约束求解器获取所有可满足的解，这为匹配相同的路径带来了难度。在本文中，利用两条路径的约束表达式构造出新的约束表达式，然后根据求解器对新约束的求解结果确定路径是否匹配；程序中的迭代为路径匹配带来了难度，当迭代的次数与程序输入相关时，程序中的路径条数也无法确定。本文采用添加“循环上限”的方法来处理这个问题；程序调用也是等效性判定过程中的困难问题。当被调函数的源码已知时，使用本文的 SCEP 方法能够进行等效性验证，而当程序包含源码未知的库函数调用时，此种情况下的等效性判别问题尚待解决。

本文的创新点包括：提出基于执行路径状态比对的等价性验证方法，对科学计算程序进行自动化的语义等价性验证；采用 Matching-Constraint 求解的方式进行执行路径条件约束的等价性判定；基于 Horn 范式约束求解进行执行路径的状态等价性判别。

## 1　相关研究

程序等效性检查在代码转换验证[4-7]、软件剽窃检测[8-15]以及回归验证[16-19]等领域具有广泛的应用。代码转换验证是证明源码和转换后的代码具备语义等价性的过程，主要应用于编译器正确性检测领域。编译器正确性检测的重要思想是将编译器优化的代码与原始代码进行比较，并确定这两个代码在语义上是否相等，主要通过构造关联两个程序的中间状态的不变式[6]，以比较中间状态的等效性，这种检测方法要求被比较的程序控制流高度相似，而编译器优化前后的程序基本都满足此条件，因此该方法在编译器正确性检测中有着较为广泛的应用；软件剽窃是指通过非法复制他人软件的代码，并利用代码混淆技术使代码在文本上发生变化，违反软件原始许可条款的行为，软件剽窃检测中同样采用了程序等效性检查的方法；回归验证是由 Godlin 和 Strichman[18]最先提出的，它用于相似程序的等效性检查，其目标是建立两个不同版本程序等效的形式证明。

Invariant-sketching[6]是一种通过不变式初稿和查询降解进行编译前后程序等效性检测的算法，它将待比较的程序的位置点抽象成节点，将两程序中由两个节点确定的路径抽象成边，并设计算法构造描述两个函数关系的 JTFG(关联传递函数图，joint transfer function graph)。在构造 JTFG 的过程中，使用“不变式初稿”来表示两条边中变量的线性关系，从而进行“边”的关联；如果存在无法关联的边，则判定两程序不等价。SamaTulyataII[5]提出针对循环的代码转换验证。该方法将转换前后的代码抽象成基于 Petri Net 的模型，在带有循环的程序中，循环执行的次数未知，路径构造器将 CPN(Colour Petri Net)模型表示成有限路径的形式，再使用基于路径的等价性检查器进行等价性验证。

Xu 等人[11]总结了软件剽窃检测五个方面的需求，并根据这五个方面的需求评估了现存的软件剽窃检测方法。目前软件剽窃检测中采用最广泛的是是动态软件胎记技术。早期一些基于源码的技术中，也通过程序等价性检查的方式进行剽窃检测。GPLAG[12]是基于静态源码比对的检测方法，通过构造程序依赖图(PDG，program dependence graph)并使用图同构检测算法，能够检测经过 FA、IR、SR、CR、CI[12]这几种混淆方式混淆的代码的等价性，但对于不透明谓词和循环展开这两种方式的代码混淆则无法进行软件剽窃鉴别。Myles 和 Collberg[13]提出基于动态控制流的方法 WPPB(Whole Program Path Birthmark)进行剽窃鉴别。这种方法利用 WPP(Whole Program Path)[20]能够标记程序动态行为中的固有规律的特点，识别出保留语义的代码转换。LoPD[10]是一种动态和静态相结合的软件剽窃检测技术，利用动态符号执行来获得执行路径的语义并寻找路径偏移，再检测两条偏移的路径是否在语义上等价。这种技术能够针对多种代码混淆方式进行整个程序的剽窃检测以及局部的剽窃检测。

LLRêVE-DYNAMIC[16]是针对回归验证的等效性判定技术，采用插入同步点的方式打破程序中的循环控制流，在同步点处构造霍恩约束来表示两个被比较的程序中间状态间的关系，再使用约束求解器进行求解，根据求解结果判断程序的中间状态是否等效。这种方法对程序控制流的相似度有较高的要求，当程序中存在循环结构并且迭代次数不相同时，需要对程序在源码级别上进行“循环展开(Loop Unrolling)”和“循环剥离(Loop Unpeeling)”操作，将程序转换为控制流相似的程序才能进行等效性判定。

这些针对特定领域的等效性判定技术都难以直接应用于科学计算程序的等效性判定，SCEP 方法的提出有助于解决该问题。

## 2　方法

### 2.1 SCEP 介绍

本文将程序等效性判别技术应用于科学计算程序的等效关系证明，提出基于 C/C++源码和 Fortran 源码的程序等价性判别方法 SCEP。与之前专门针对 C/C ++语言的代码验证工具不同，SCEP 方法在工作过程中不需要用户理解程序并对程序进行人工干预，此外，它也可以在控制流相似度较小的程序上进行判定工作。

在方法设计中，SCEP 是基于程序路径的等效性验证。通过比较两个程序中的分支条件，构造 Matching-Constraint，根据 SMT 求解器的求解结果，可以构造两个程序中相匹配的路径。此外，已有的方法是通过构造两个程序的中间状态的等价关系来验证整个程序的等效性，这种方法能够处理的情况是有限的，SCEP 不考虑程序中间状态是否等价，只关注程序的出口状态，这使得 SCEP 在已有方法的基础上能够应对更多类型的程序转换方式。

本文收集 C/C++和 Fortran 编程语言实现的函数，针对采用不同方式进行的语义不变的函数转换，构造“spcLib”数据集，分别使用前人的方法和 SCEP 对数据集中的函数转换进行等效性验证实验，我们发现 SCEP 能够针对更多的函数转换方式得到验证结果，并且能够在 Fortran 代码上工作。

### 2.2 框架概述

SCEP 的主要架构包括预处理、路径采集与匹配、状态比较三个模块，如图 1 所示。如果输入的两个待比较的源代码在功能上等效，则将输出“true”作为处理结果；对于其他任何无法判定为等效的程序对，输出结果“undecidable”。

在预处理阶段，LLVM 框架将源代码编译成 LLVM 中间表示的形式。待分析的代码使用的编程语言是 C/C ++或 Fortran，在将 Clang 和 Flang[21]。安装为 LLVM 的前端之后，可以使用 LLVM 框架将 C/C++和 Fortran 编译为中间语言。除了安装上的差异，使用 Clang 和 Flang 进行源码到中间语言的转换在操作非常相似。

路径匹配阶段会将待分析的程序中具有等效约束的两条路径匹配为一组路径对。根据两程序的分支条件构造 Matching-Constraint，再使用 SMT 求解器（例如 Z3[22]）对它进行求解。如果转换前后的程序中所有路径的约束都相同，则可以匹配这两个程序中的所有路径；否则，将认为这两个程序的等效性无法判定。

最后一步是比较所有路径对的路径状态。通过两程序中的变量构造 Horn 范式约束，再次使用求解器进行约束求解，根据求解情况，判定两程序在语义上是否等效。

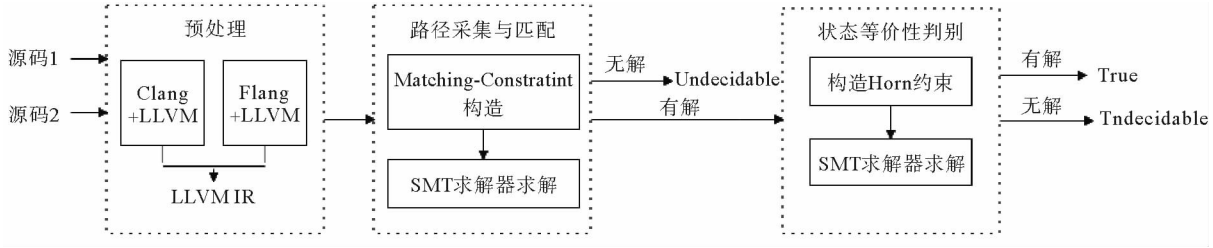


图 1 SCEP 整体架构

Fig.1 Overall architecture of SCEP

### 2.3 预处理

在预处理阶段，程序源码将被转换为 LLVM 中间表示（Intermediate Representation，IR），接着会在 IR 级别上进行等效性判别。在 LLVM IR 上进行等效性判别的优势主要体现在以下三个方面：

（1）屏蔽不同编程语言语法上的差异性。

本文的主要工作是判定 C/C ++代码与 Fortran 代码或 Fortran 代码与 Fortran 代码之间的等效性，然而，Fortran 和 C/C ++语法完全不同，这将导致在具体的判别工作之前需要对源代码进行复杂的分析工作，LLVM + Clang/Flang 可以将 C/C ++源代码和 Fortran 源代码分别编译为中间语言。LLVM IR 将指令、函数和程序抽象为相应的类，用户只需要使用 LLVM 框架提供的接口来对程序进行分析和处理。

（2）减少程序分析过程中的工作量。

采用 LLVM 框架便于在中间语言上生成控制流图。在 SCEP 方法中，程序的每条路径都由约束表达式表示，为了获得所有路径的约束，必须对程序的控制流进行分析，通过遍历 LLVM 生成的控制流图上的基本块来获得路径约束，为整个处理过程减少了很多冗余的工作。

（3）便于实现程序向逻辑公式的转换。

为了将程序的等效性判定问题转化成逻辑公式的求解问题，首先需要将程序转化为 SSA 形式（Static single assignment form），以便能够利用自动化的方式将 SSA 形式的代码表示转化成逻辑公式。LLVM IR

是 SSA 形式的中间表示，能够满足我们的需求。

### 2.4　路径采集与匹配

程序的执行路径指的是在一个回合执行期间执行的所有指令的序列。执行路径的语义可以通过符号执行来获取。更确切地说，执行路径的语义可以表示为输出变量对应于输入变量的符号表达式以及路径约束表示。

在检查程序状态的等效性之前，SCEP 方法会将两个目标程序中的对应路径组合起来。如果两个程序中某两条路径的约束等效，则它们将构成一对路径组合，在后续过程中将基于这两条路径进行状态等效性验证。要找到两条路径所对应的约束的所有解，然后判断它们在无限解的情况下是否都相同显然是不切实际的。本文提出一种新的方法进行路径等效性判别：分别根据两个程序所有分支处的谓词构造每条路径的约束，再利用不同程序的两条路径的约束构造 Matching-Constraint。如果 SMT 求解器对 Matching-Constraint 求解的结果为“unsatisfiable”，则两条路径不等价；如果求解结果为“satisfiable”则路径等价。假设使用“ $cond_A$”和“ $cond_B$”表示两条路径的约束，“ $cond_{new}$”表示构造出的 Matching-Constraint，则，

$$cond_{new}=(cond_A \wedge \neg\ cond_B) \vee (\neg\ cond_A \wedge cond_B)$$

如果满足条件“$cond_{new}$”的集合为空，则得出结论 $cond_A$ 和 $cond_B$ 是等效的，否则得出相反的结论。上述结论很容易得到证明：

$$cond_A \wedge \neg\ cond_B = false \Leftrightarrow set_A \bigcap \overline{set_B} = \Phi \Leftrightarrow set_A \subseteq set_B$$

$$\neg\ cond_A \wedge cond_B = false \Leftrightarrow \overline{set_A} \bigcap set_B = \Phi \Leftrightarrow set_B \subseteq set_A$$

$$cond_{new} = false \Rightarrow cond_A \wedge \neg\ cond_B = false,$$

$$\neg\ cond_A \wedge cond_B = false \Rightarrow set_A = set_B \Rightarrow cond_A = cond_B$$

（“ $set_A$”和“ $set_B$”分别表示满足 $cond_A$ 和 $cond_B$ 的集合。）

### 2.5　状态比较

状态转换谓词用来抽象程序从入口处到出口处状态的转换过程，具体来说，状态转换谓词指的是程序入口处变量集合和出口处变量集合的关系表达式。耦合不变式指的是两条路径约束相匹配的执行路径的不变式，利用耦合不变式和状态转换谓词构成 Horn 范式约束可以进行状态的等价性判别。

在路径匹配完成后，SCEP 不考虑两程序中间值的大小关系，而仅对函数每条路径的出口处的变量的值进行比较。假设两个待比较的程序分别为 P 和 Q，P 与 Q 入口处的变量分别表示为 $Xp$ 和 $Xq$，出口处的变量分别表示为 $Xp'$ 和 $Xq'$。针对这些变量，在函数的入口和出口处构造耦合不变式 $\mathbb{C}b(Xp,Xq)$ 和 $\mathbb{C}e(Xp',Xq')$，再根据程序 P 和 Q 的指令构造 $Xp$ 和 $Xq$ 的状态转换谓词，分别用 $\varphi(Xp,Xp')$ 和 $\pi(Xq,Xq')$ 来表示。据此可以生成 Horn 范式约束，其形式为：

$$\mathbb{C}b(Xp,Xq) \wedge \varphi(Xp,Xp') \wedge \pi(Xq,Xq') \rightarrow \mathbb{C}e(Xp',Xq')。$$

最后，将生成的约束提供给可用于 Horn 子句求解的求解器 Z3，Z3 将尝试寻找一个满足上述约束公式的实例。如果求解器能找到解，那么这条路径的程序状态等效；如果求解器显示解不存在或者求解超时，则两程序的等效性判定结果为“undecidable”。

### 2.6　程序示例

图 2 中函数的功能是计算非负十进制数 $n(n \geqslant 0)$ 的位数，很容易看出函数(a)和函数(b)在功能上是等价的。由于循环次数由 $n$ 的大小来决定，路径匹配的过程中函数路径的条数将是一个非常大的数字，这会导致路径爆炸问题，因此在分析过程中，需要为两个程序的路径约束添加“循环上限”，这种方法提高了验证错误率，但能有效解决路径爆炸问题。假设添加的循

```
1 int dig10(int n){
2   int result = 1;
3   n = n/10;
4   while(n>0){
5       result++;
6       n = n/10;
7   }
8   return result;
9 }
```

(a)

```
1 int dig10(int n){
2   int result = 1;
3       While(n>0){
4           if(n<10) return result;
5           if(n<100) return result+1;
6           if(n<1000) return result+2;
7           if(n<10000) return result+3;
8           n=n/10000;
9           result+=4;
10      }
11      return result;
12 }
```

(b)

图 2　计算非负十进制数的位数

Fig.2　Computing the digits of a non-negative decimal number

环上限是 $n<10^8$，那么对于函数(a)，有：

path1：$(n_{a1}>0)\wedge(n_{a1}/10\leqslant 0)$；

path2：$(n_{a2}/10>0)\wedge(n_{a2}/100\leqslant 0)$；

path3：$(n_{a3}/10>0)\wedge(n_{a3}/1\ 000\leqslant 0)$；

……

path8：$(n_{a8}/10>0)\wedge(n_{a8}/100\ 000\ 000\leqslant 0)$；

对于函数(b)，有：

path1：$(n_{b1}>0)\wedge(n_{b1}<10)$；

path2：$(n_{b2}>0)\wedge(n_{b2}\geqslant 10)\wedge(n_{b2}<100)$；

path3：$(n_{b3}>0)\wedge(n_{b3}\geqslant 10)\wedge(n_{b3}\geqslant 100)\wedge(n_{b3}<1\ 000)$；

path4：$(n_{b4}>0)\wedge(n_{b4}\geqslant 10)\wedge(n_{b4}\geqslant 100)\wedge(n_{b4}\geqslant 1\ 000)\wedge(n_{b4}<10\ 000)$；

……

path8：$(n_{b8}>0)\wedge(n_{b8}\geqslant 10)\wedge(n_{b8}\geqslant 100)\wedge\cdots\cdots\wedge(n_{b8}<100\ 000\ 000)$。使用 2.4 节中提到的路径匹配方法能构造这两个函数路径一一对应的关系，例如，对于函数(a)的 path2 和函数(b)的 path2，可以构造出函数入口处的耦合不变式：

$$n_{a2}=n_{b2};$$

函数(a)的状态转换谓词为：

$$(result_1=1)\wedge\left(n_{a2}'=\frac{n_{a2}}{10}\right)\wedge(result_1'=result_1+1);$$

函数(b)的状态转换谓词为：

$$(result_2=1)\wedge(result_2'=result_2+1);$$

如果函数(a)和函数(b)等效，函数出口处的变量关系满足：

$$result_1'=result_2'。$$

如果在 Horn 范式约束的前件有效的条件下，能够推测出其后件有效，则该约束是可满足的。据此，可以判定两函数在$(n_{a2}/10>0)\wedge(n_{a2}/100\leqslant 0)$和$(n_{b2'}>0)\wedge(n_{b2}'\geqslant 10)\wedge(n_{b2'}<100)$这两条等价的路径下状态等效。按照这种方法，同样可以对其他路径下的状态进行比较，如果每条路径的状态都是相同的，那么判定这两个函数等效。

## 3 实验与评估

### 3.1 实验测试程序集

本文的实验测试程序集 spcLib 通过两种途径进行获取：一是基于常见的代码混淆方法以及程序优化方法构造的不同功能的函数集合(以下称该类函数集合为一类函数集合，包括相同数量的 C++和 Fortran 两个版本)；二是从三种不同的 libc 实现中提取的程序样例(以下称该类函数集合为二类函数集合)。三种不同的 libc 实现分别为：dietlibc[22]、glibc[23] 和 OpenBSD[24]。

其中，一类函数集合通过本文总结的六种常见的代码混淆或程序优化方式进行构造，这六种语义等价转换的方式包括：

(1)格式变更。

插入或删除程序中的空格或注释。这种混淆方式仅对基于文本的检测技术产生效果。

(2)标识符重命名。

可以在不影响程序正确性的情况下不断地更改程序中的标识符名称。本文的方法将源码转换成中间语言，当程序仅有标识符被改变时，改变前后的代码会被转换成相同的 LLVM IR。

(3)插入噪声指令。

插入的代码不干扰原始程序逻辑。在进行路径探索的过程中，没有任何一个基本块会跳转到噪声指令所在块进行执行，因此本文的方法可针对添加噪声指令前后的等价程序进行程序的等价性验证。

(4)语句重新排序。

某些语句可以重新排序而不会导致程序错误，例如变量的声明语句，这种语句重新排序并不会改变 LLVM 中间语言中基本块的跳转顺序。

(5)控制语句替换。

例如将 for 循环由 while 循环进行等价替换，或者将有害的 goto 语句等价转换成其他控制语句。这种代码混淆方法同上一条一样，并不会改变程序的控制流，程序的执行路径不会发生改变。

(6)循环语句展开。这是一种循环转换技术，这种转换方法基于“以空间换时间”的思想，用于优化程序的执行速度。转换可以由程序员手动执行，也可以由优化编译器手动执行。

### 3.2 实验结果统计

使用 SCEP 验证 spcLib 测试集中所有函数的等价性，通过验证结果来证明 SCEP 方法的可靠性。目前发现的与本工作最为接近的方法是 Kiefer 等人提出的 LLRêVE-DYNAMIC[16]，因此实验中选择该工具对 spcLib 测试集合中 C++语言实现的一类函数以及所有二类函数进行语义等价性验证，统计出两种方法下的验证结果数据，并将它们进行比较，据此评估 SCEP 的优势。

实验使用 SCEP 和 LLRêVE-DYNAMIC 分别对 spcLib 中的函数进行程序等价性验证，统计的实验结果数据如表 1 所示。一类函数集合中包括等数量的 Fortran 函数和 C++函数，SCEP 对所有一类函数的验证结果为等价；二类函数集合中有 150 对函数，其中，判定结果为等价的有 132 对。由于 LLRêVE-DYNAMIC 不支持 Fortran 程序的等价性验证，因此使用该工具对一类函数集合中的 C++函数和二类函数集

合中的函数进行等价性验证，统计结果表明，对于一类函数集合中的函数对，判定结果为等价的有 126 对，对于二类函数集合中的函数对，判定结果为等价的有 123 对。

**表 1　两种方法的验证结果**

Table 1　Verification results of two methods

| 类别 Class | 编程语言① | | 是否需要理解源码并标记源码(Y/N)② | | 检测的函数数量(对)③ | | 检测结果为等价的数量(对)④ | |
|---|---|---|---|---|---|---|---|---|
| | LLRêVE-DYNAMIC | SCEP | LLRêVE-DYNAMIC | SCEP | LLRêVE-DYNAMIC | SCEP | LLRêVE-DYNAMIC | SCEP |
| 一类函数集合⑤ | C++ | C++/Fortran | Y | N | 150 | 300 | 126 | 300 |
| 二类函数集合⑥ | C | C | Y | N | 150 | 150 | 123 | 132 |

Note:①Programming languages;②Is it nuessary to understand the source code and mark(Y/N);③Number of detected functions(pair);④Number of detected results as equivalent(pair);⑤First class function collection;⑥Second class function collection.

### 3.3　评估

LLRêVE-DYNAMIC[16]在进行程序的等价性验证时，要求待验证的程序控制流高度相似，与 LLRêVE-DYNAMIC 不同，SCEP 不要求被验证的两程序的控制流相似；另外，LLRêVE-DYNAMIC 在工作工程中，如果系统无法自动生成同步点(Synchronization Point)，需要用户对源码进行理解，并在源码的适当位置手工插入同步点，从而进行同步点之间的等价性验证，而 SCEP 不需要预先对源码进行理解基础上的处理。从这两个方面来看，SCEP 方法比 LLRêVE-DYNAMIC 在使用过程中的限制更少，并且更加自动化。

SCEP 和 LLRêVE-DYNAMIC 对 spcLib 中二类函数的验证情况对比如图 3 所示，该图显示了两种方法分别对两类函数集合进行验证，得到等价的结果所占的比例。在一类函数的验证中，LLRêVE-DYNAMIC 能对 84%左右的函数进行等价性判别，而 SCEP 方法能够达到 100%的等价验证率；在验证的 150 对二类函数中，LLRêVE-DYNAMIC 能对 82%左右的函数进行等价性判别，而 SCEP 方法能够验证 84%左右的函数对的语义等价性。

从条形图的高度可以看出，在两类函数集的实验中，本文的 SCEP 方法验证成功率略高于 LLRêVE-DYNAMIC 方法。SCEP 在对第一类函数集合的验证上可以达到 100% 的成功率，远高于 LLRêVE-DYNAMIC 方法，但在第二类函数集合上，两种方法的成功率十分接近。

从检测程序的源码上进行分析，推测产生上述实验结果的原因可能是第一类函数集合是通过本文的六种变换方式进行构造的，而我们的验证方法在设计过程中专门考虑到了这六种类型的语义等价转换，因此针对这类函数的等价性验证成功的概率更大。而二类函数包含更多种类别的语义等价的转换方法，某些转换方式在本文方法下无法成功验证函数语义的等价性。函数等价变换的方式十分复杂，要对该方法进行更准确的评估，还需要在更大的测试程序集上设计实验进行验证，以便完善本文的 SCEP，使其能够对更多类别的等价转换方式下的语义等价函数进行等价性验证。但总的来说，本文的 SCEP 成功进行程序语义等价性验证的几率比 LLRêVE-DYNAMIC 更高。

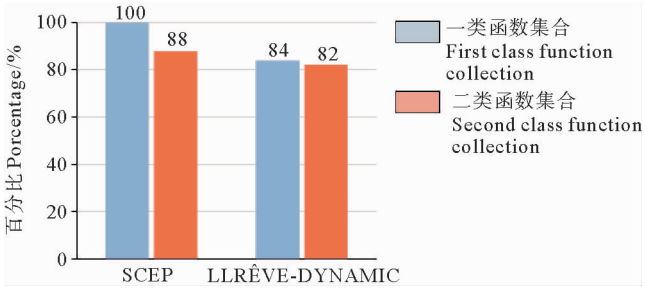


图 3　两种方法在两类函数集合上的验证情况

Fig.3　The verification effect of the two methods on the two types of function sets

## 4　结语

SCEP 是一种基于路径的状态等价性检测方法。针对手工的程序转化比自动化程序转化具有更大的差异性，SCEP 对转化前后控制流差异较大的代码采用构造状态不变式的方式进行验证，基于对函数的结果状态验证，来构造对程序整体的等效性判定，可作为科学计算程序版本更新和平台迁移的正确性验证工具。在下一步工作中，我们将完善 spcLib 数据集，并对方法进行改进，提高方法的计算效率和适用性，使其能更为高效地处理大规模科学计算程序转换的等效性判定。

**参考文献：**

[1] Johnson S R, Prokopenko A, Evans K J. Automated fortran-C++ bindings for large-scale scientific applications[J]. Computing

in Science and Engineering, 2019, 22(5): 84-94.

[2] Feldman S I. A fortran to C converter[J]. ACM SIGPLAN Fortran Forum, 1990, 9(2): 21-22.

[3] Grosse-Kunstleve R W, Terwilliger T C, Sauter N K, et al. Automatic fortran to C++ conversion with FABLE[J]. Source Code for Biology & Medicine, DOI: 10.1186/1751-0473-7-5.

[4] Abate C, Blanco R, Ciobc T, et al. Trace-Relating Compiler Correctness and Secure Compilation[M]. [s.l.]: Programming Languages and Systems, 2020.

[5] Mittal R, Banerjee R, Sarkar S, et al. Translation Validation of Code-Optimizing Transformations, Involving Loops, Using Petri Net-Based Program Models[C]. [s.l.]: Petri Nets and Software Engineering Workshop, 2020.

[6] Gupta S, Saxena A, Mahajan A, et al. Effective use of smt solvers for program equivalence checking through invariant-sketching and query-decomposition[J]. SAT 2018: Theory and Applications of Satisfiability Testing, 2018, 10929: 365-382.

[7] Kiefer M, Klebanov V, Ulbrich M. Relational program reasoning using compiler IR[J]. Journal of Automated Reasoning, 2016, 60(3): 337-363.

[8] Zhenzhou, Tian, Qinghua, et al. Software plagiarism detection with birthmarks based on dynamic key instruction sequences[C]. [s.l.]: IEEE Transactions on Software Engineering, 2015.

[9] Tian Z, Liu T, Zheng Q, et al. Reviving sequential program birthmarking for multithreaded software plagiarism detection[J]. IEEE Transactions on Software Engineering, DOI: 10.1109/TSE.2017.2688283.

[10] Ming J, Zhang F, Wu D, et al. Deviation-based obfuscation-resilient program equivalence checking with application to software plagiarism detection[J]. IEEE Transactions on Reliability, 2016, 65(4): 1-18.

[11] Xu X, Fan M, Jia A, et al. Revisiting the Challenges and Opportunities in Software Plagiarism Detection[C]. IEEE: 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER), 2020.

[12] Liu C, Chen C, Han J, et al. GPLAG: detection of software plagiarism by program dependence graph analysis[C]. [s.l.]: Acm Sigkdd International Conference on Knowledge Discovery & Data Mining, 2006: 872-881.

[13] Myles G, Collberg C. Detecting Software Theft Via Whole Program Path Birthmarks[J]. International Conference on Information Security, 2004, 3225: 404-415.

[14] Jhi Y C, Wang X, Jia X, et al. Value-based program characterization and its application to software plagiarism detection[C]. [s.l.]: 2011 33rd International Conference on Software Engineering (ICSE), 2011: 756-765.

[15] Jhi Y C, Jia X, Wang X, et al. Program characterization using runtime values and its application to software plagiarism detection[J]. IEEE Transactions on Software Engineering, 2015, 41(9): 925-943.

[16] Kiefer M, Klebanov V, Ulbrich M. Relational program reasoning using compiler ir: Combining static verification and dynamic analysis[J]. Journal of Automated Reasoning, 2017, 60(3): 337-363.

[17] Angelis E D, Fioravanti F, Pettorossi A, et al. Relational verification through horn clause transformation[C]. Springer Berlin Heidelberg: International Static Analysis Symposium, 2016: 147-169.

[18] Godlin B, Strichman O. Regression verification[C]. [s.l.]: IEEE Design Automation Conference, 2009: 466-471.

[19] Klebanov V, Philipp Rümmer, Ulbrich M. Automating regression verification[J]. Formal Methods in System Design, 2018, 52(3): 229-259.

[20] Larus J R. Whole program paths[J]. Acm Sigplan Notices, 1999, 34(5): 259-269.

[21] Klimowicz G. Flang: The LLVM Fortran Front-End[EB/OL]. https://tcevents. chem. uzh. ch/event/12/contributions/42/, 2020.

[22] von Leitner F. Diet libc[EB/OL]. https://www.fefe.de/dietlibc/. 2016.

[23] GNU C. Library [EB/OL]. https://www. gnu. org/software/libc/. 2016.

[24] OpenBSD libc[EB/OL]. http://cvsweb.openbsd.org/cgi-bin/cvsweb/src/lib/libc/. 2016.

# A Method for Equivalence Checking of Scientific Programs Based on State Comparison of Execution Paths

Qu Haipeng, Zhang Minyuan, Han Qingdi, Lin Xijun
(College of Information Science and Engineering, Ocean University of China, Qingdao 266100, China)

**Abstract:** The computing equivalence checking of scientific programs is an important issue in the field of scientific computing. Semantics-preserving transformation on scientific programs is quite common in the process of algorithm improvement, program optimization, and version evolution. The same function before and after transformation must ensure the equivalence of the calculation process. For this problem, A state comparison of execution paths method—SCEP is proposed, which determines the equivalence of calculations through constraint solving and precise comparison of the execution path states of different program versions in the same input space. SCEP method was compared with the existing methods on the spcLib test data set composed of multiple library functions. The result shows that SCEP can identify more equivalent function transformation types, not only for "Fortran-to-Fortran" transformation, but also for the "Fortran-to-C/C++". SCEP approach helps to provide complete calculation equivalence guarantee for the optimization and improvement process of scientific codes and the migration process from Fortran to C/C++.

**Key words:** program equivalence checking; scientific codes; state matching; regression verification; semantics-preserving transformation

责任编辑　徐　环

(上接第 65 页)

# Evaluation of the Effect of Anaerobic Fermentation of Sludge Pretreated with Ozone in Different Dosages for Methane Production

Wang Taiheng, Chen Caixia, Wang Xudong, Li Xianguo, Zhang Dahai
(College of Chemistry and Chemical Engineering, Ocean University of China, Qingdao 266100, China)

**Abstract:** The sludge in the secondary sedimentation tank was concentrated and pretreated with different doses of ozone to study the methane production capacity of the sludge in the anaerobic fermentation process. Experiments have proved that ozone pretreatment can promote the release of sludge organic matter and the utilization of microorganisms to organic matter is improved, thereby enhancing the ability of anaerobic fermentation to produce methane. The SCOD content of concentrated sludge and the removal rate of SCOD in anaerobic fermentation increased with the increase of ozone dosage. The methane production is positively correlated with the SCOD removal rate. When the ozone dosage is 0.24 g/g TSS, the methane production from the sludge is the largest, which is 1.88 times that of the blank group. Analyzing the composition of microbial diversity, it was found that the relative abundance of bacteria associated with the process of methanogenesis increased. The main methanogen in the anaerobic fermentation process is *Methanosaeta*. The increase in ozone dose is conducive to the growth of methanogens, thereby enhancing the methanogenic effect.

**Key words:** ozone pretreatment; anaerobic fermentation; methane; microbial community

责任编辑　徐　环