X10 程序的差别分析方法

陈雨亭[1),3)] 杨 威[1)] 赵建军[1),2),3)]

[1)](上海交通大学软件学院 上海 200240)

[2)](中国科学院软件研究所计算机科学重点实验室 北京 100190)

[3)](上海市计算机软件评测重点实验室 上海 201112)

摘 要 程序差别分析是程序调试的常见手段,其主要用于分析程序不同版本之间的差异信息.然而,将现有程序差别分析算法扩展到并行程序语言还面临众多挑战.其主要原因在于并行程序复杂性较高,且存在支持并行活动的特殊机制,如地址(或线程)、活动、同步等,从而为有效进行程序差别分析设置了障碍.文中研究基于 PGAS 模型的 X10 并行程序的程序差别分析方法,并设计了一种语句级的、针对 X10 程序的程序差别分析算法 X10Diff. X10Diff 包括下列步骤:(1)匹配原程序和修改后程序中的类、接口、方法及地址;(2)为待分析程序片段构建基于地址的程序流图,并建立相应简化图;(3)迭代扩展并比较简化图,并将差别信息定位到代码中.

关键词 程序差别分析;X10;程序流图;软件测试;程序调试

中图法分类号 TP311 **DOI 号** 10.3724/SP.J.1016.2015.01082

Program Differencing for X10

CHEN Yu-Ting[1),3)] YANG Wei[1)] ZHAO Jian-Jun[1),2),3)]

[1)](*School of Software, Shanghai Jiao Tong University, Shanghai* 200240)

[2)](*State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing* 100190)

[3)](*Shanghai Key Laboratory of Computer Software Testing & Evaluating, Shanghai* 201112)

Abstract Program differencing is a widely used technique for program debugging, while it is still not easily used for parallel programs. One main reason is that a parallel program can be complex, and some mechanisms (e.g., place, activity, clock, and barrier) also set barriers for program differencing. In this paper we focus on program differencing for X10 parallel programming language, and design an algorithm for differencing of X10 programs. The algorithm contains three steps: (1) match the places, classes, interfaces, methods, and places between programs of two versions; (2) construct the extended program diagrams for the programs and simplify them to simplified diagrams; (3) iteratively unfold and compare the simplified diagrams and identify the differences between the programs.

Keywords program differencing; X10; program diagram; software test; program debugging

1 引 言

程序差别分析(Program Differencing)是一种重要的程序调试技术[1],主要用于分析不同程序或程序不同版本之间的差异信息,如程序修改位置、程序相对于原版本所做出的修改类别(增加、删除、修改)等.

本文研究 X10 并行程序的差别分析方法. X10

收稿日期:2013-08-27;最终修改稿收到日期:2014-10-13.本课题得到国家"九七三"重点基础研究发展规划项目基金(2015CB352203)、国家自然科学基金(91118004,61100051,61272102)、中国科学院软件研究所计算机科学国家重点实验室开放基金(SYSKF1101)和上海市计算机软件评测重点实验室开放基金(SSTL2011_02)资助.**陈雨亭**,男,1978 年生,博士,讲师,主要研究方向为软件工程、程序分析. E-mail: chenyt@cs.sjtu.edu.cn. **杨 威**,男,1989 年生,博士研究生,主要研究方向为软件工程. **赵建军**,男,1964 年生,博士,教授,博士生导师,主要研究领域为软件工程、程序分析.

是 IBM 为并行编程开发的语言①，采用分布式全局地址空间(Partitioned Global Address Space，PGAS)模型. 目前，存在针对 C、Java 等语言的程序差别分析算法，但几乎没有针对并行程序语言(包括 X10)的程序差别分析算法. 一方面，程序差别分析算法是语言相关的，不易在不同程序范型(面向过程、面向对象、并行等)之间实现算法迁移. 例如 BMAT[2] 不能识别与面向对象特性(如异常、继承结构)相关的程序差别信息；Semantic diff[3] 及 Horwitz 的技术[4] 用于分析面向过程的程序，不能用于分析面向对象程序中的差别信息等. 另一方面，现有程序差别分析算法难以扩展至 X10 等并行程序语言中，主要原因在于并行程序复杂性较高，多线程编程方式及同步、异步并行机制对差别分析造成困难，具体包括：

(1) 对 X10 程序的差别分析不仅需要涵盖与类、方法、语句等程序语法或结构相关的差别，也需要考虑程序活动(Activity)与地址(Place)的差别. 其中，活动可被看成是轻量级线程，地址用于存储数据及提供活动范围，待分析程序之间活动或地址存在差别也意味着程序语义上存在差异. 例如，程序员通过重构技术将单个逻辑地址上的活动拆分为多个逻辑地址上的并行活动，对重构前后程序之间进行基于语法树的差别分析不能得出程序从顺序向并行、分布式执行方式的转变. 我们需要考虑 X10 程序中与并行相关的活动及地址信息，进行基于语义的程序差别分析.

(2) 基于语义的程序差别分析通常依赖于程序流图的匹配. 这里的难点包括为 X10 程序建立支持并行活动的程序流图、判断两个程序流图是否存在同构关系、及对同构流图实现匹配与比较等.

本文设计了一种针对 X10 并行程序的程序差别算法 X10Diff，用于实现两个 X10 程序(或程序版本)语句级差别分析. X10Diff 包含下列步骤：(1) 匹配两个程序的地址、类、接口及方法；(2) 为待分析程序构建扩展程序流图，并将其简化为简化图. 这里采用基于分布式地址的程序控制流图，以描述 X10 中的并行机制；(3) 迭代扩展、比较简化图，将差别定位至代码中.

本文设计的程序差别算法对 X10 程序的维护、测试等具有意义. 在程序维护过程中，程序变更可能导致程序多个方面受到影响，可以对程序进行变化影响分析以确保程序的正确性. 在该过程中，程序差别分析定位 X10 程序改变的具体位置，而影响分析进一步分析程序哪些部分会受到改变的影响. 关于回归测试用例选择的研究中，程序差别信息可以帮助测试人员评估 X10 程序修改轮廓信息[2]，决定哪些测试用例需要在修改后的程序版本上重新测试运行，以减少测试的代价.

本文主要贡献和创新点包括两方面. 首先，大多数程序差别分析技术是针对顺序程序或者面向对象程序而言的，而本文将差别分析技术扩展至并行程序分析中. 其次，现有程序差别分析技术主要针对于程序文本或语法结构，而 X10Diff 是一种基于语义的程序差别分析技术：对 X10 程序进行基于语义的差别分析有助于辨识程序的全局行为差异及各逻辑地址上的活动差异.

2　X10 语言

X10 是 IBM 为在分布式系统上通过共享内存而进行并行计算所设计的编程语言，其采用分布式全局地址空间 PGAS 模型. PGAS 是并行编程模型，其假设一个全局地址空间是逻辑分布的，每一个逻辑地址对应一个本地处理器.

X10 利用了活动和派生的概念来描述并行执行的软件行为. 在 X10 中，一个计算被拆分为若干活动(Activity)到多个地址(Place)中并行完成；一个活动可以在本地或者远程派生出多个子活动，子活动也可以派生出更多子活动. async(*place*) *S* 语句表明在地址 *place* 中创建一个异步活动，*S* 为活动中执行的语句. 一个活动在其生命周期中，仅能在一个地址中被执行，且仅能直接访问在同一地址中的数据，对远程数据(即不在同一地址中的数据)的访问只能通过在远程地址中派生新活动实现.

图 1 中 $QSort_2$ 显示了一个描述快速排序算法的 X10 程序片段：第 25 行的 *partition* 函数将待排序的数组 *data* 分割成独立的两部分，从索引 *left* 到 *index* 的数据比从 *index* 到 *right* 的所有数据都要小；第 27～28 行对从 *left* 到 *index* 的数据进行快速排序；第 30～31 行对从 *index* 到 *right* 的数据进行快速排序.

$QSort_2$ 是一个并行算法. 假设当前活动为 A_1，活动地址为 $place_1$. A_1 在第 28 行派生一个异步活动(假设活动名为 A_2，地址为 $place_2$). A_2 和 A_1 分别对索引 *index* 前后的数据快速排序. 第 26 行关键字 finish 指明活动 A_1 和 A_2 完成后排序任务结束.

① http://x10-lang.org/

```
24. static def qsort(data:Rail[int],left:int,right:int) {
25.   index:int=partition(data,left,right);
26.
27.   if (left<index-1)
28.     qsort(data,left,index-1);
29.   if (index<right)
30.     qsort(data,index,right);
31. }
```

程序 $QSort_1$

```
24. static def qsort(data:Rail[int],left:int,right:int) {
25.   index:int=partition(data,left,right);
26.   finish {
27.     if (left<index-1)
28.       async qsort(data,left,index-1);
29.
30.     if (index<right)
31.       qsort(data,index,right);
32.   }
33. }
```

程序 $QSort_2$

图 1 排序算法 $QSort_1$ 与其并行版本 $QSort_2$

3 程序差别例子

本节首先采用一个简单例子说明 X10 程序间的差别. 图 2 中,程序 $APPTest_2$ 的 foo 方法是程序 $APPTest_1$ 的 foo 方法经过重写(修改语句、插入语句或者删除语句)所得. 可以得到 $APPTest_1$ 和 $APPTest_2$ 中节点对应关系及差别信息,如表 1 所示. 这里,G_1、G_2 分别为目标程序(如 $APPTest_1$ 与 $APPTest_2$)的程序流图(见本文第 4 节);节点号对应程序语句行号;$APPTest_1$ 与 $APPTest_2$ 存在三处差别,含两处语句修改和一处语句插入.

```
public class APPTest {
    public def foo():void {
1.      var size:int=rand();
2.      var count:int=1;
3.      while(count<size){
4.        var num1:int=rand();
5.        var num2:int=rand();
6.        var product:int=num1*num2;
7.        var sum:int=sum+product;
8.        count++;
9.      }
10.     Console.OUT.println(count);
    }
  }
```

程序 $APPTest_1$

```
public class APPTest {
    public def foo():void {
1.      var sum:int=0;
2.      var size:int=rand();
3.      var count:int=0;
4.      while(count<size){
5.        var num1:int=rand();
6.        var num2:int=rand();
7.        sum=num1*num2+sum;
8.        count++;
9.      }
10.     Console.OUT.println(count);
    }
  }
```

程序 $APPTest_2$

图 2 具有差别的程序 $APPTest_1$,$APPTest_2$

表 1 图 2 中两个 foo 方法差别信息

差别类型	差别信息
无差别节点及对应关系	G_1 中节点(1)对应 G_2 中节点(2) G_1 中节点(3)对应 G_2 中节点(4) G_1 中节点(4)对应 G_2 中节点(5) G_1 中节点(5)对应 G_2 中节点(6) G_1 中节点(8)对应 G_2 中节点(8) G_1 中节点(9)对应 G_2 中节点(9) G_1 中节点(10)对应 G_2 中节点(10)
差别①(修改节点)	G_1 节点(2)修改至 G_2 节点(3)
差别②(修改节点)	G_1 节点(6,7)修改至 G_2 节点(7)
差别③(插入节点)	G_1 节点(无)修改至 G_2 节点(1)

图 2 介绍了一个简单的 X10 程序差别的例子,并未涉及地址及活动等更为复杂的情况. 这里回顾图 1 的例子以说明与语义相关的程序差别. 图 1 中提供了两个快速排序的代码片段:$QSort_1$ 与 $QSort_2$ 的差别不仅在于后者使用了 async 和 finish 关键字,更在于 $QSort_2$ 通过两个并行活动完成快速排序. 表 2 显示了 $QSort_1$ 与 $QSort_2$ 的差别信息,既包括两段程序语句之间的对应关系,也包括与活动相关的三处差别,包括一处活动缩减,一处活动新建及一处活动同步关系的建立.

表 2 图 1 中两个 qsort 方法差别信息

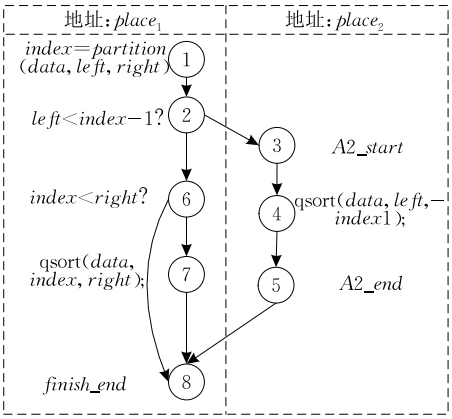
差别类型	差别信息
无差别节点及对应关系	G_1 中节点(25)对应 G_2 中节点(25) G_1 中节点(27)对应 G_2 中节点(27) G_1 中节点(28)对应 G_2 中节点(28) G_1 中节点(29)对应 G_2 中节点(30) G_1 中节点(30)对应 G_2 中节点(31)
差别①(缩减活动)	G_1 节点(27,28,29,30)修改至 G_2 节点(30,31)
差别②(新建活动)	G_1 节点(无)修改至 G_2 节点(27,28)
差别③(新建同步)	G_1 节点(无)修改至 G_2 节点(26)

4 X10 程序程序流图的同构关系

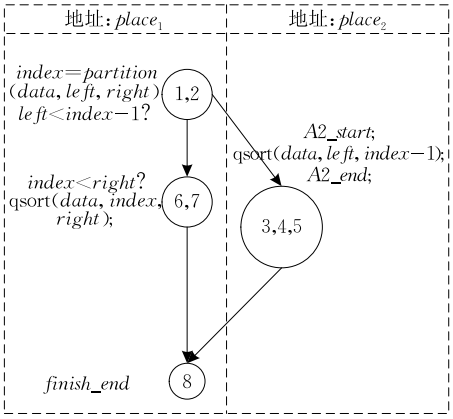
本文设计一种基于程序流图的、针对 X10 程序的程序差别方法 X10Diff. 传统上,一个程序流图 G 可定义为一个四元组$(N,E,START,END)$,其中:N 是节点集合;E 是有向边集合;$START$ 和 END 是程序流图唯一的入口点和唯一的出口点,且 $START\in N,END\in N$. 此外,程序流图中,如果一个节点有多个后继节点,其为决策节点(例如,条件谓词对应的节点),从该节点发出的边称之为分支;如果一个节点存在 0 或 1 个后继节点,其为处理节点.

上述定义并不能有效表示 X10 语言中分布存储的特性. 研究中采用多泳道的概念来扩展程序流

图，以对不同地址中的活动进行描述：每一个泳道对应程序一个逻辑地址，并包含一个或多个子程序（活动）流图. 例如，图 1 中 $QSort_2$ 程序对应的程序流图如图 3(a)所示，其包含两个并行活动，分别处于不同泳道中.



(a) $QSort_2$的X10程序流图



(b) $QSort_2$的X10程序流图的简化图

图 3　拥有两个地址的程序流图及简化图

当程序活动与地址相关时，程序流图的构建依赖于对活动驻存地址的计算. 我们此前提出一种基于约束的地址分析技术 Leopard(Locality Analyses of Partitioned Global Address Space Programs)[5]，以精确识别程序单个地址（泳道）中的活动和对象：Leopard 定义了对象、活动和地址之间的约束关系，并计算每一个地址表达式的可能值；Leopard 进一步分析程序中的指向关系，建立位置约束图来计算每一个实际地址中的可能活动与对象. 基于 Leopard，我们将活动对应流图指定至不同泳道中，建立包含地址信息的程序流图.

程序差别分析中的一个重要步骤是匹配程序结构. 本文基于集簇的概念建立简化程序流图，以确立两个程序流图的同构关系. 令一个多泳道的程序流图的节点集合 N 中有子集 C，$N\backslash C$ 为 N 中 C 的补集. 如果 C 满足如下条件，则称其为 G 中的集簇：(1) C 中所有节点属于同一泳道；(2) 在 C 中有入口点 $STARTc$，其为 C 中唯一由 $N\backslash C$ 中节点所指向的点；(3) 在 $N\backslash C$ 中有 $ENDc$，其为 $N\backslash C$ 中唯一被 C 中节点所指向的点. 四元组 $\langle C\cup\{ENDc\},(C\cup\{ENDc\})\times(C\cup\{ENDc\})\cap E,STARTc,ENDc\rangle$ 则构成一个程序子图.

用单个节点替代程序流图中的一个集簇，所获得的流图被称之为简化图. 例如，图 3(a)可以被简化为 4 个节点的简化图，如图 3(b)所示.

设 $G_1=\langle N_1,E_1,START_1,END_1\rangle$ 和 $G_2=\langle N_2,E_2,START_2,END_2\rangle$ 为两个程序流图（或简化图），B_1 和 B_2 为相应的分支集合. 定义 1 定义 G_1 和 G_2 的同构关系.

定义 1.　带标记的程序流图的同构关系. 当且仅当以下条件满足时，映射 $f:N_1\rightarrow N_2$ 是 G_1 和 G_2 的同构关系：

(1) 对于 N_1 中任意 n_1，存在唯一的 $n_2\in N_2$ 使得 $nl(f(n_1))=nl(n_2)$；

(2) 对于 N_1 中任意 m、n，$(f(m),f(n))\in E_2$ 当且仅当 $(m,n)\in E_1$；

(3) 对 B_1 中分支 (d_1,n_1)，存在唯一的分支 $(d_2,n_2)\in B_2$，使得 $al(f(d_1),f(n_1))=al(d_2,n_2)$.

定义中，nl、al 为相应的标记函数：如果 N_1 或者 N_2 中节点 n 对应一条程序语句，$nl(n)$ 为相应的程序语句；如果 n 对应一个集簇，那么 $nl(n)$ 对应标记 MOD；$al(b)$ 将决策节点中的每一个分支 b 赋予不同标记（例如，对于条件语句，默认将边标记设为 TRUE 或 FALSE）.

5　程序差别计算

X10Diff 的主要思想在于为修改前后的 X10 程序分别建立多泳道的程序流图，将其简化为多泳道的简化图，迭代扩展简化图并完成同构简化图的比较.

5.1　建立同构简化图

给定程序版本 P_1 和 P_2，建立多泳道的程序流图 G_1 和 G_2，并识别出相同数量的集簇，使得：

(1) 对于 G_1 中的集簇，G_2 中有且只有一个集簇对应；

(2) 当每个图中的集簇被单个节点所代替时，两图中的节点一一对应.

对于扩展的程序流图 G_1 和 G_2，首先匹配程序

流图中的泳道(即活动地址),匹配中遵循如下原则:

(1)相似名匹配.如果 G_1 和 G_2 中两个泳道 P_{1i} 和 P_{2j} 具有相似的标签,则匹配 P_{1i} 和 P_{2j}.例如,图 4(a)中,P_{11} 和 P_{21} 展现相似名匹配的情况;

(2)相似流图匹配.如果 G_1 和 G_2 中两个泳道 P_{1i} 和 P_{2j} 中包含的活动(或子流图)相似,则匹配泳道 P_{1i} 和 P_{2j}.例如,图 4(b)中,P_{11} 和 P_{22} 具有相似的集簇,可以对它们实现匹配;

(3)复杂匹配.如果 G_1 中泳道 P_1 包含的活动(或子流图)对应 G_2 中泳道 $P_{21}\sim P_{2n}$ 的活动(或子流图),将 P_1 分解为 n 条泳道 $P_{11}\sim P_{1n}$,并与 $P_{21}\sim P_{2n}$ 分别匹配,反之亦然.例如,图 4(c)中,由于部分活动地址变更,P_{11} 与 P_{21} 和 P_{12} 与 P_{22} 不能完全匹配.我们将各泳道中的活动进一步分割,得到匹配关系为 $\langle P_{111},P_{211}\rangle$、$\langle P_{112},P_{213}\rangle$、$\langle P_{121},P_{22}\rangle$ 和 $\langle P_{122},P_{212}\rangle$.

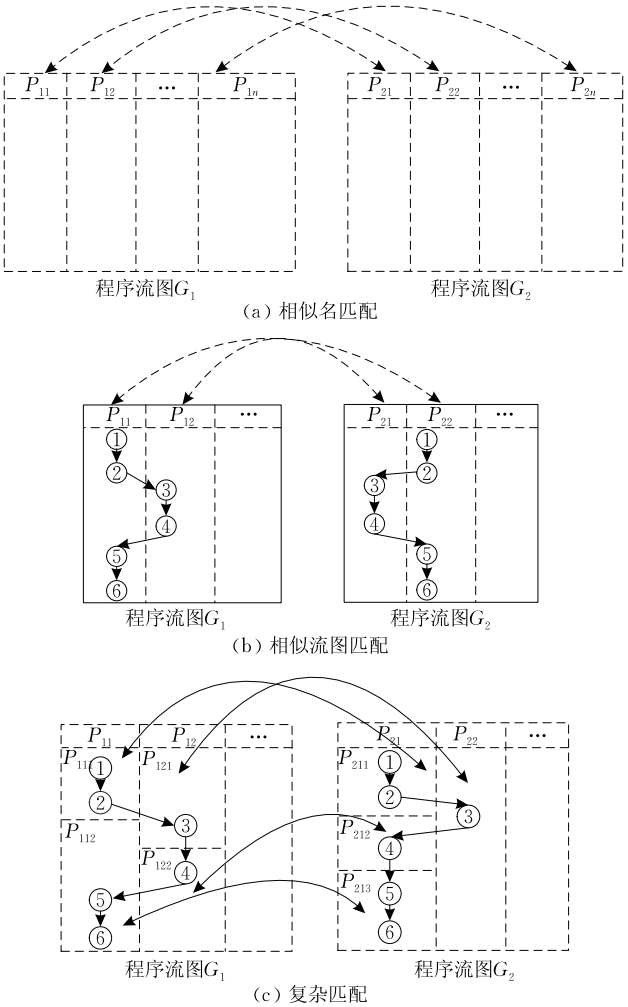


图 4 扩展程序流图中的泳道匹配

对应于匹配泳道内的程序子图 G_1 和 G_2,如果 G_1 中的节点无法与 G_2 中的节点对应,则对程序子图执行如下 3 个操作:

(1)重新标记.将某个节点标记改成 MOD;

(2)折叠节点.如果 C 是一个节点集簇,则用标记为 MOD 的节点替代 C;

(3)移除节点.当确定程序修改是在原程序中移除一组唯一入口点和唯一出口点的指令集以后,可在相应程序流图上移除节点.如果 C 是一个集簇,C 中节点及相关的边可从图中移除.

如算法 *Find_Isomorphism* 所示,对程序子图 G_1 和 G_2 中的一个或多个节点进行上述操作后,将其简化为同构简化图 SG_1 和 SG_2.同构简化图为程序差别计算的基础.

算法 1. *Find_Ismorphism.*

输入:匹配泳道中的程序子流图 G_1、G_2
输出:匹配的简化图 SG_1、SG_2
BEGIN
  FOR (G_1 或 G_2 上的任意决策节点 n) DO
    IF(n 为一个包含多个节点集簇的起始点) THEN
      计算出 n 的最小集簇 $Cluster(n)$;
    ENDIF
  ENDFOR
  调用 $Search\_and\_Reduce(G_1,G_2)$,获得匹配的简化图 SG_1,SG_2;
  FOR (SG_1 和 SG_2 中的每一个集簇) DO
    将相邻集簇合并为新的集簇并标记成 MOD;
  ENDFOR
END

算法 2. *Search_and_Reduce.*

输入:集簇 C_1,C_2
输出:集簇 RC_1,RC_2
BEGIN
  FOR(C_1 和 C_2 中的决策节点 n) DO
    IF(n 是起始节点或一个已被计算的集簇) THEN
      折叠 n 节点的集簇;
    ENDIF
  ENDFOR
  获得临时集簇 C_1' 和 C_2';
  IF (C_1' 和 C_2' 无法形成同构) THEN
    对 C_1' 和 C_2' 进行深度优先搜索,并通过节点移除、节点折叠和标记节点操作将其简化为 RC_1 和 RC_2;
    FOR(同构简化图 RC_1 和 RC_2 中对应嵌套集簇(R_1,R_2)) DO
      扩展 R_1 和 R_2;
      $Search\_and\_Reduce(R_1,R_2)$;
      更新 R_1 和 R_2 为下一对嵌套集簇;
    ENDFOR
  ENDIF
END

用图 2 中程序 $APPTest_1$ 和 $APPTest_2$ 以说明算法，图 5 是匹配过程中相应的简化图 G_{11}～G_{16} 以及 G_{21}～G_{26}. 具体匹配过程如下：

(1) 将两程序流图中的决策节点转换为集簇，得到简化图 G_{11} 和 G_{21}；

(2) 使用简化算法 *Search_and_Reduce* 对两个图进行深度优先访问，并采用折叠节点操作，将图分别简化为 G_{12} 和 G_{22}；

(3) 对 G_{12} 和 G_{22} 归导，使其在全局范围内同构；若无法同构，则对图折叠，在折叠集簇中建立对应关系. 例子中，G_{12} 和 G_{22} 中折叠集簇的起始点临时标记“while (*count*<*size*)”. 因为节点 1“*sum*：int＝0”在 G_{12} 中无对应匹配，保留 G_{12}，并在 G_{22} 中去掉节点 1，得到了 G_{23}. 因此，$G_{12}=G_{13}$；

(4) 重新标记 G_{13} 中节点 2“*count*：int＝1”和 G_{23} 中节点 3“*count*：int＝0”为 MOD，得到 G_{14} 和 G_{24}；

(5) 由于匹配的折叠集簇需保留尽可能多的标签，还原图至 G_{15} 和 G_{25}；

(6) 比较 G_{15} 中集簇 $C_1=\{3,4,5,6,7,8\}$ 和 G_{25} 中集簇 $C_2=\{4,5,6,7,8\}$，发现对应 while 循环代码不一致. 前者中的语句 6、7 被后者中的语句“*sum*：=*sum*＋*num*1＊*num*2”所替代，因此折叠 G_{15} 中的节点 6、7，重新标记 G_{25} 中节点 7，得到 G_{16} 和 G_{26}.

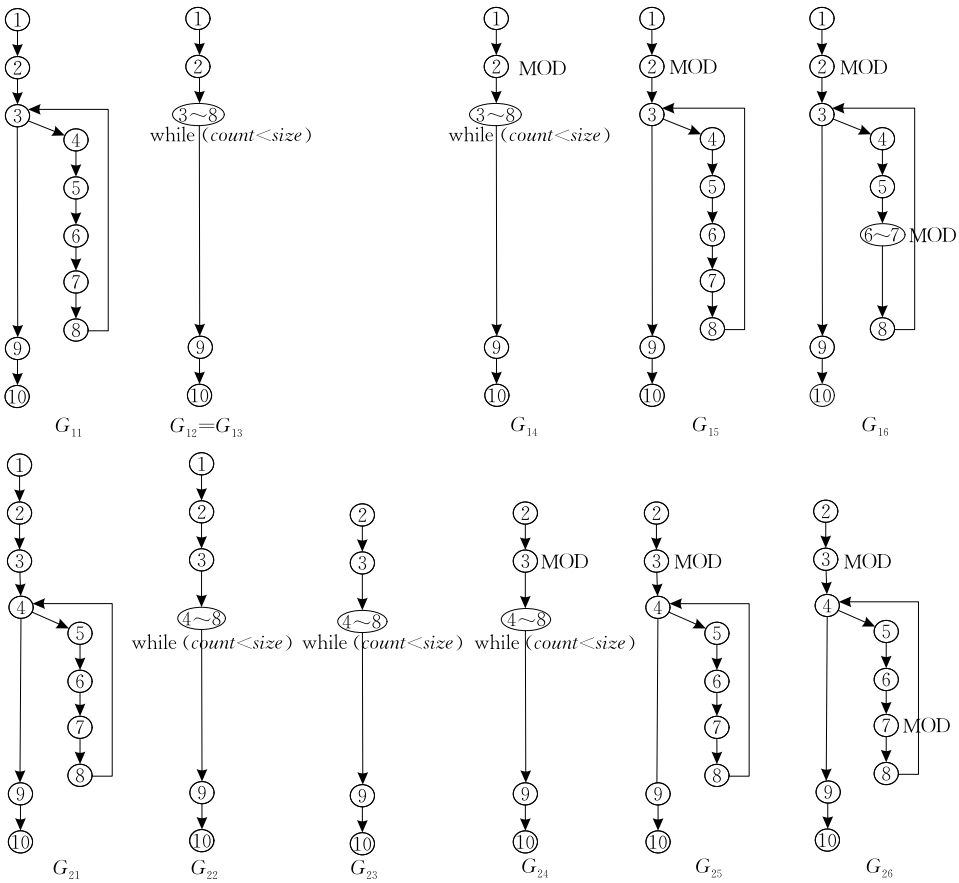


图 5　程序流图 G_{11}～G_{16} 以及 G_{21}～G_{26}

根据定义 1，G_{16} 和 G_{26} 同构.

算法正确性和复杂性分析 *Find_Ismorphism* 是图同构关系的构建算法，其算法正确性主要表现为两个方面：建立简化图步骤的正确性和迭代扩展简化图并建立同构关系步骤的正确性. 前者依赖于简化图的定义，对于一个多泳道的程序流图，其最小简化图唯一，从而保证算法正确性. 后者正确性依赖于程序节点、子图等的相似度：对分析程序中相似节点进行节点移除、节点折叠和标记节点、扩展节点等基本操作，从而完成同构图的匹配.

算法复杂性的最坏情况为 $O(n^2)$，这里，n 为程序流图的节点数目. 一个节点数为 n 的程序流图简化的最坏情况是程序流图得不到简化，因此要求两个程序流图所有节点进行一一匹配；假设判断两个节点是否匹配的代价为常量，匹配的复杂性为 $O(n^2)$.

算法复杂度的最好情况为 $O(n)$. 程序一开始简

化为单个节点,然后最多经过 n 次扩展,实现同构图匹配.最好情况下,每次扩展过程中,被扩展节点均实现精确匹配.

5.2 节点匹配

下面给出匹配算法 *Match*,以在同构简化图的基础上建立节点匹配.算法中,用户自定义相似度标准 S,以决定两个节点是否相似;$Comp(n_1,n_2,LH,S,R)$根据相似度标准 S 对节点 n_1 和 n_2 匹配,如果实现匹配,则将带标记的节点对加入集合 R 并返回 TRUE,否则返回 FALSE;$succ(Set)$计算 Set 中每个节点的后继节点并以节点集合的形式返回;$EdgeMatching(n_1,n_2)$返回配对的边.

算法 3. *Match*.

输入:源程序中的节点 n_1;
　　修改后程序版本中的节点 n_2;
　　往前读入最大节点数 LH;
　　相似度标准 S;
　　带标记节点对集合$\langle\langle node,node\rangle,label\rangle R$
输出:布尔值 *result*
BEGIN
　result=FALSE;
　将节点 n_1 和 n_2 定位至程序流图 G_1 和 G_2 中;
　将G_1和G_2出口点对(即$\langle\langle x_1,x_2\rangle$,"UNCHANGED"$\rangle$)加入到 R;
　将所有点对$\langle s_1,s_2\rangle$压入栈 ST;
　WHILE (ST 不为空) DO
　　从 ST 中弹出顶部元素$\langle c_1,c_2\rangle$;
　　IF $Comp(c_1,c_2,LH,S,R)$ THEN
　　　match=$\langle c_1,c_2\rangle$;
　　　result=TRUE;
　　ELSE
　　　match=NULL; $L_1=\{c_1\}$; $L_2=\{c_2\}$;
　　　FOR($d=0$;$d<LH$;d++) DO
　　　　$L_1=succ(L_1)$; $L_2=succ(L_2)$;
　　　　IF $Comp(c_1,p_2\in L_2,LH,S,R)\parallel$
　　　　　$Comp(p_1\in L_1,c_2,LH,S,R)$ THEN
　　　　　把第一对节点标记为配对;
　　　　　BREAK;
　　　　ENDIF
　　　ENDFOR
　　ENDIF
　　IF *match*!=NULL THEN
　　　将$\langle$*match*,"UNCHANGED"$\rangle$压栈至 R;
　　　将 c_1 和 c_2 设为匹配的两个节点;
　　ELSE
　　　将$\langle c_1,c_2$,"MODIFIED"$\rangle$压栈到 R 中;
　　ENDIF
　　将由 $EdgeMatching(c_1,c_2)$返回的每一对边的相关节点压栈到 ST 中;
　ENDWHILE
　RETURN *result*;
END

考察图 6 中的例子,图 7 显示了修改前后的程序流图.算法将出口节点$\langle 3,3'\rangle$添加到配对节点集中,将节点对$\langle 1,1'\rangle$压栈到 ST 中.在 while 第一次循环中,调用 $Comp(1,1',2,0.5,R)$进行比较,将$\langle 1,1'\rangle$设为匹配并放入 R 中.由于 1、1′节点均只有一条出边,且两条边匹配,将节点对$\langle 2,2'\rangle$压入工作表.

```
1.  public class A {
2.    public def m1(): void {
3.      …
4.    }
5.  }
6.
7.  public class B extends A {
8.    public def m2(): void {
9.      …
10.   }
11. }
12.
13. public class D {
14.   public def m3(var a:A): void {
15.     a.m1();
16.     …
17.   }
18. }
```

程序 P

```
1.  public class A {
2.    public def m1(): void {
3.      …
4.    }
5.  }
6.
7.  public class B extends A {
8.    public def m1(): void {
9.      …
10.   }
11.   public def m2(): void {
12.     …
13.   }
14. }
15.
16. public class D {
17.   public def m3(var a:A): void {
18.     a.m1();
19.     …
20.   }
21. }
```

程序 P'

图 6 待比较的示例程序 P,P'

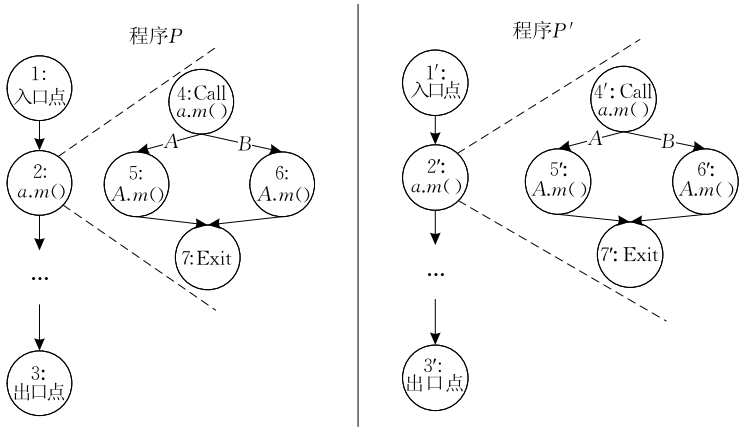


图 7　简化图节点间匹配

将节点 2、2′扩展为两个子图. 算法配对起始点对〈4,4′〉、出口点对〈7,7′〉，并将〈5,5′〉、〈6,6′〉压入至 ST_1 中.

算法接下来将节点对〈5,5′〉配对并压入 R_1 中，将后继节点对〈7,7′〉压入至 ST_1.

依次比较节点对〈6,6′〉、〈6,7′〉、〈7,6′〉，因为均不匹配，则添加节点对〈6,6′〉到 R_1 中，标记为“MODIFIED”，将后继节点对〈7,7′〉压入至 ST_1.

因为大多数节点在子图中都被标记为“UNCHANGED”，而设定相似度标准 S 为 0.5，故两个流图匹配.

因此，$Comp(2,2',2,0.5,R)$返回 TRUE，将节点对〈2,2′〉加入至 R，并标记为“UNCHANGED”. 算法对简化图剩余部分继续完成匹配.

上述例子包括了简化图节点匹配的多种情况，包括节点的匹配、向前读入节点、流图扩展等.

算法正确性和复杂性分析. *Match* 算法用于判断两个节点是否匹配. 算法正确性主要取决于：(1) 匹配节点与流图中其他节点之间控制依赖关系的建立，即如果节点依赖的节点集能与另一个程序中节点依赖的节点集形成相似匹配关系，则为这两个节点建立匹配关系；(2) 相似度取值，如果相似度要求高，则不易为两个程序的节点匹配，反之则容易匹配.

单个节点匹配算法复杂性最坏情况为 $O(n^2)$，n 为待匹配的流图节点数. 当该节点与其他所有节点都有控制依赖，最坏情况下需要检查每一条控制边是否连接到相应的节点上. 然而，由于设立了相似度，实际计算中当不满足相似度要求后计算会结束. 同理，针对流图所有节点匹配算法复杂性最坏情况为 $O(n^4)$，在此情况下，需要对两个流图中所有节点进行一一匹配.

5.3　差别分析

结合匹配算法，设计 X10 程序差别方法如下. 这里，将修改前后的程序版本 P_1 和 P_2 作为输入，并接收另两个参数：LH 是差别分析中匹配节点时向前读入的最大节点数；S 用于计算两个程序流图的相似值. 输出为类对集合 CP、接口对集合 IP、方法对集合 MP、地址对集合 PP 及以〈〈*node*, *node*〉, *status*〉形式的节点对集合 NP. X10Diff 首先完成在类与接口级别的对比，然后进行方法级别的对比及节点级别的对比.

(1) 对比 P_1 和 P_2 中的类，将匹配的类成对放入类对集合 CP 中. 在进行类的对比的时候，算法会寻找 P_1 中与 P_2 中名字相似的类，并输出匹配类对集合(CP). 那些未被加入 CP 的 P_1 中的类，是被删除的类，而 P_2 中未被加入 CP 中的类，是新增类.

(2) 对比 P_1 和 P_2 中的接口，并采用类似类匹配的方法判断删除或新增接口，将匹配的接口成对放入集合 IP 中.

(3) 在每一对类或者接口中继续匹配方法. 算法匹配 P_1 与 P_2 中具有相同(或相似)签名的方法，并输出匹配方法对集合(MP). P_1 中未出现的在 MP 中的方法为在后一版本中被删除的方法，反之则为后一版本中新增的方法.

(4) 节点匹配过程中，算法为匹配好的方法对〈m_1, m_2〉创建控制流图 G_1 和 G_2 及建立扩展的程序流图，并实现程序流图中泳道的匹配，将匹配成功的活动地址对〈$place_1$, $place_2$〉加入到 PP 中. 将每一对匹配的泳道中的程序子流图分别折叠到一个节点. 接下来，利用 *Match* 算法匹配简化图，创建节点对集 NP.

算法结束时，会输出成对节点的集合，也会返回匹配成功的类对 CP、接口对 IP 和方法对 MP、地

址对 PP.

5.4 X10Diff 支撑工具架构

我们基于上述算法开发了 X10Diff 工具,以识别程序之间的差别. X10Diff 工具以两个 X10 程序为输入,输出报告为这两个程序的差别信息. 图 8 展示了 X10Diff 工具的开发框架.

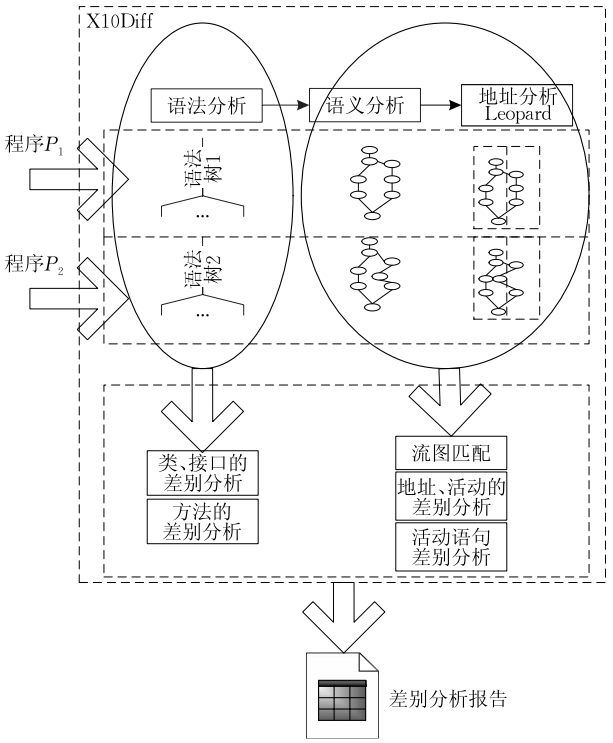


图 8 X10Diff 工具架构

针对目标程序,X10Diff 基于 WALA(Watson 分析库①)对其进行语法分析,生成抽象语法树,并在抽象语法树的基础上完成类、接口和方法的匹配与差别分析. X10Diff 接下来进行程序语义分析,生成控制流图. 我们也利用 WALA 框架为程序源码生成控制流图,并利用 Leopard 地址分析算法分析程序中地址、活动与对象信息. 接下来,X10Diff 在多泳道程序控制流图的基础上,利用匹配算法完成两个同构图匹配与活动内语句的差别分析,生成程序差别信息.

5.5 实验效果

研究中选择 X10 程序样例程序②进行实验,以检验 X10Diff 的效果. 实验中安排两组独立的参与人员:程序差别植入者负责在 X10 程序中植入若干程序差别,程序差别分析者采用 X10Diff 对程序差别予以识别. 实验主要步骤包括:

(1)植入程序差别. 程序差别植入者针对被分析程序,手工植入若干差别,即对程序进行修改,并在植入过程中对修改点予以记录. 手工植入的程序差别类型包括对类、方法、地址、语句等各级别的修改,并保证每个程序的修改程度小于 10%,以使得程序不同版本之间得到匹配. 这里将植入的程序差别集合记为 $Diff_1$. 这一步骤后,可以获得修改前后的程序版本供程序差别分析. 注意的是,实验中未能够严格保证修改后的程序是能够被执行的.

(2)程序差别分析. 程序差别分析者独立采用 X10Diff 进行识别差别,从而识别出程序中的类、方法、地址、语句等的匹配关系及程序差别. 我们将识别出的程序差别集合记录为 $Diff_2$.

实验结束后,实验参与者对比 $Diff_1$ 和 $Diff_2$,以验证 X10Diff 的效果. 实验结果如表 3 所示. 这里,我们将($Diff_1$—$Diff_2$)记录为差别漏报,将($Diff_2$—$Diff_1$)记录为差别误报. 识别率 $X10DiffRate = \#(Diff_1 \cap Diff_2)/\#Diff_1$,误报率 $FalsePositiveRate = \#(Diff_2 - Diff_1)/\#Diff_2$. 实验结果表明,X10Diff 可以识别程序中 83%的差别,但也存在 17%的漏报率和 17%的误报率.

表 3 实验中的程序差别分析结果

程序	手工植入差别数	植入差别类型	识别的差别数	识别率/%	误报率/%
ArrayCopy	1	地址	1	100	0
MontePi1	1	语句	2	100	50
MontePi2	2	类、语句	3	100	33
MontePiAsync	2	地址、表达式	1	50	0
MontePiAsync2	2	表达式	2	100	0
MontePiCluster	2	方法、地址	2	100	0
MontePiCluster2	2	表达式、地址	1	50	0
合计	12	/	10(识别)+2(误报)	83	17

进一步的分析表明,被疏忽的差别主要来源于地址的计算,即静态计算不能获取准确的地址,对地址的匹配会存在失误. 而误报主要原因在于语句进行了次序调整,例如,在植入错误的时候被记为 1 处差别,但在分析过程中被记为多处差别.

6 相关工作

典型的程序差别分析方法可以分为两类:基于程序语法的和基于程序语义的. 前者强调以程序文本、程序结构、语法树为基础,实现程序差别分析. GNU Diffutils③是一个用来定位文件差别的工具

① http://wala.sourceforge.net
② http://x10.sourceforge.net/documentation/guide/2.1.0/src/
③ http://www.gnu.org/software/diffutils/

集，通过使用 diff 命令，用户可以定位两个文件（包括程序）之间区别，并采用按行输出的方式输出这些区别. Duley 等人[6]提出针对 Verilog HDL 语言的程序差别算法，该算法可以将 Verilog 特定的变化类型以语法形式报告出来. Tsantalis 等人[7]开发了一个基于网络的差别分析工具 WebDiff，其核心算法为树差别算法 VTracker，该算法可以将 XML 文档表示为偏序树，并进行差别分析. UMLDiff 是一个用于 UML 类图的差别分析工具，可以识别出 UML 类图中对类的增、删、改等操作[8]. 然而，程序是具有语义的，上述基于语法或程序结构的程序差别分析方法并不能识别程序语义上的差异.

基于语义的程序差别分析方法需要根据目标语言来设计. 对于传统顺序程序语言（如 C）和面向对象程序语言（如 C++、Java 等），目前均已存在程序差别算法并已经被广泛应用到程序分析中. 例如 BMAT[2]、Semantic diff[3]及 Horwitz 的技术[4]等主要用于处理面向过程的程序. Term 提出的差别分析算法[9]主要针对 Java 程序. Görg 等人[10]提出了针对 AspectJ 程序的面向方面程序差别分析方法等. Loh 和 Kim 开发了一个逻辑结构差别分析工具 LSdiff[11-12]，可以自动将程序结构差别识别为一组与变化相关的逻辑规则. CloneDifferentiator 是一个用于对程序克隆实现差别分析的工具[13]，其从程序依赖图中获取克隆中的语义信息，并使用图形匹配技术计算精确的克隆差别.

对于基于 PGAS 模型的 X10 语言，其程序分析方法并不成熟. 本文针对 X10 这一高性能计算语言，研究相关程序差别分析方法. X10Diff 是一个基于语义的程序差别分析方法. 就目前所检索到的文献资料而言，几乎没有基于语义的并行程序差别分析方法，而 X10Diff 则对此方面进行了探索. 此外，X10Diff 在差别分析过程中结合了逻辑地址计算和活动匹配方法，有效地提高了分析的精度.

7　结束语

对并行程序中程序差别的识别对于并行程序分析、测试、理解和维护有很重要的作用. 本文提出了比对两个 X10 程序并寻找语句级差别的算法 X10Diff：给出两个 X10 程序，识别匹配的类与方法，建立每一个匹配方法对和地址对，并比较活动、语句等方面的差别. 本文所提出的方法对其他并行语言的程序分析同样有启示作用，比如说 Cray Inc. 开发的 Chapel 或是 Sun 开发的 Fortress 语言.

进一步的工作主要包括：现在对 Place 的处理方法仍不能识别比较复杂的程序机制，如时钟、原子块等，需要进一步解决这些机制基础上的并行程序的差别分析；将 X10 程序差别分析的工具集成到 X10 集成开发工作中；进行综合实验，以检验算法的实际效果等.

参　考　文　献

[1] Myers E W. An $O(ND)$ Difference algorithm and its variations. Algorithmica, 1986, 1(2): 251-266

[2] Wang Z, Pierce K, McFarling S. BMAT—A binary matching tool for stale profile propagation. Journal of Instruction-Level Parallelism, 2000, 2(1): 1-20

[3] Jackson D, Ladd D. A. Semantic diff: A tool for summarizing the effects of modifications//Proceedings of the Internation Conference on Software Maintenance. Victoria, Canda, 1994: 243-252

[4] Horwitz S. Identifying the semantic and textual differences between two versions of a program//Proceedings of the ACM SIGPLAN'90 Conference on Programming Language Design and Implementation. White Plains, New York, USA, 1990: 234-246

[5] Sun Q, Chen Y, Zhao J. Constraint-based locality analysis for X10 programs//Proceedings of the ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation. Rome, Italy, 2013: 137-146

[6] Duley A, Spandikow C, Kim M. A program differencing algorithm for verilog HDL//Proceedings of the IEEE/ACM International Conference on Automated Software Engineering. Antwerp, Belgium, 2010: 477-486

[7] Tsantalis N, Negara N, Stroulia E. WebDiff: A generic differencing service for software artifacts//Proceedings of the 27th IEEE International Conference on Software Maintenance (ICSM'2011). Williamsburg, USA, 2011: 586-589

[8] Xing Z, Stroulia E. UMLDiff: An algorithm for object-oriented design differencing//Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE 2005). Long Beach, USA, 2005: 54-65

[9] Apiwattanapong T, Orso A, Harrold M J. A differencing algorithm for object-oriented programs//Proceedings of the 19th IEEE International Conference on Automated Software Engineering. Linz, Austria, 2004: 2-13

[10] Görg M, Zhao J. Identifying semantic differences in AspectJ programs//Proceedings of the 18th International Symposium on Software Testing and Analysis. Chicago, USA, 2009: 25-36

[11] Loh A, Kim M. LSdiff: A program differencing tool to identify systematic structural differences//Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2. Cape Town, South Africa, 2010: 263-266

[12] Kim M, Notkin D. Discovering and representing systematic code changes//Proceedings of the 31st International Conference on Software Engineering. Vancouver, Canada, 2009: 309-319

[13] Xing Z, Xue Y, Jarzabek S. Clone Differentiator: Analyzing clones by differentiation//Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011). Lawrence, USA, 2011: 576-579

CHEN Yu-Ting, born in 1978, Ph.D., assistant professor. His research interests include software engineering, program analysis and testing, and formal methods.

YANG Wei, born in 1989, Ph. D. candidate. His research interests include software engineering and information security.

ZHAO Jian-Jun, born in 1964, Ph. D., professor, Ph. D. supervisor. His research interests include software engineering, program analysis and testing.

Background

Program differencing is a program analysis technique to compare different versions of a program and identify the program changes. It provides engineers with supports in regression testing and software maintenance. For example, identifying which parts of a program are changed can help identify tests that need to be rerun or redesigned.

Program differencing still faces strong challenges in practice. One challenge is that a purely syntactic differencing may not provide enough information for performing testing or maintenance. Another challenge is that program differencing techniques strongly depend on programing languages (e. g., C, C++, Java), and applying them to new languages usually requires significant modifications. In addition, few program differencing techniques do exist for high performance computing languages or parallel programming languages (e. g., X10, Scala, and Fortress).

In this paper, we propose an approach to identifying program differences in X10 programs. We make two contributions in this paper: an X10 program model for analysis, and a program differencing algorithm for X10. We believe that the approach can help identify both syntax and semantics changes in X10 programs, which can be further used in regression testing and software maintenance.

We have concentrated on program differencing techniques for several years and proposed some program differencing methods. To reduce the manual effort of assessing potential affected program parts during software evolution, we (Görg and Zhao) have proposed an approach to differencing aspect-oriented programs. We work out a set of well-defined signatures for elements in the AspectJ language. In accordance with these signatures, and with those existent for elements of the Java language, we investigate a set of signature patterns that can be used with the module matching algorithm. We (Zhang and Zhao) have also developed a tool, called Celadon, which automates the change impact analysis for AspectJ programs. It analyzes the source code of two AspectJ software versions, and decomposes their differences into a set of atomic changes together with their dependence relationships. The analysis result is reported in terms of impacted program parts and affected tests.

This work is supported by the National Basic Research Program (973 Program) of China (Grant No. 2015CB352203), the National Natural Science Foundation of China (Grant Nos. 91118004, 61100051, and 61272102), State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences (Grant No. SYSKF1101), and Shanghai Key Laboratory of Computer Software Testing & Evaluating (Grant No. SSTL2011_02). These projects focus on researching on techniques and methods for improving software reliability and trustworthiness. Under these projects, several papers have been published, mainly on program analysis and testing.